CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Investigation of high performance configurations on the Evolved Packet Gateway

Master's thesis in Computer science and engineering

TSIGABU MEBRAHTU BIRHANU
GEORGIOS CHATZIADAM

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Master's thesis 2020

Investigation of high performance configurations on the Evolved Packet Gateway

TSIGABU MEBRAHTU BIRHANU
GEORGIOS CHATZIADAM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Investigation of high performance configurations on the Evolved Packet Gateway

TSIGABU MEBRAHTU BIRHANU
GEORGIOS CHATZIADAM

Supervisor: Romaric Duvignau, Computer Science and Engineering Department
Supervisor: Ivan Walulya, Computer Science and Engineering Department
Advisor: Patrik Nyman, Ericsson AB
Examiner: Philippas Tsigas, Computer Science and Engineering Department

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2020

Abstract

Modern servers today are based on multi-socket motherboards to increase their power and performance figures. These setups provide CPU interconnection through a high speed bus. If processes on one CPU need access to memory or devices local to another CPU, they need to traverse this bus and this adds a delay to the execution time. This is where the concept of Non-Uniform Memory Access (NUMA) presents as a solution. Every socket with its local memory is considered a node, that is locked and processes are not allowed to migrate. This means that loading instructions has low latency, but they can also access the main memory connected to the other NUMA nodes at a given penalty cost. The latest CPUs such as the EPYC series from AMD are using this concept even within the processor module, and there is no possibility to avoid taking into account NUMA aspects.

There has been a plethora of benchmarks to analyze the impact of NUMA node architecture on different processors. In this work, we have used the Packet Gateway of the Evolved Packet Core (EPC) as a test case to investigate the effectiveness of NUMA architecture on Intel processors on a virtual large-scale distributed production system with high performance requirements. On the virtualization setups, different CPU pining and deployment strategies are used, while Packet Per Second (pps) is the preferred performance indicator in systems like the Evolved Packet Gateway (EPG). We further describe and analyze different scenarios, combining CPU pinning and process placement, within the virtual machines running the EPG.

Keywords: NUMA, EPG, Computer Science, engineering, project, thesis.

Acknowledgements

We would first like to thank our examiner Philippas Tsigas and our supervisors Romaric Duvignau and Ivan Walulya of the Computer Science and Engineering Department at Chalmers University of Technology. Their door has always been open for us when we ran into a trouble spot or had questions about our work, and their support and guidance was of extreme importance for the completion of this project.

We would also like to thank our supervisor at Ericsson, Patrik Nyman, who assisted us in all stages of our research and made us feel like home by always checking in on us and being very immediate in his responses and actions. The technical experts involved in this project: Oscar Leijon, Jonas Hemlin, Patrik Hermansson and Devakumar Kannan provided critical support and are undoubtedly an important part of it.

Finally, we want to express our gratitude to our families for their continuous encouragement and support throughout our years of study and through the process of researching and completing this thesis. This accomplishment would not have been possible without them.

Tsigabu Mebrahtu Birhanu and Georgios Chatziadam, Gothenburg, January 2020

Contents

1
Introduction

The transition of current mobile broadband networks to 5G along with the variety of connected Internet of Things (IoT) devices and the increase in bandwidth available for end users, introduce a new challenge for the underlying system responsible for handling the traffic. The amount of data being transferred will only continue to grow in the future and there is an obvious need for more capable servers in order to meet this evolution. This project is an evaluation of the possible configurations of modern hardware, aiming to identify the components that could be used to optimize future mobile infrastructure systems in the pursuit of performance.

In this chapter, we want to describe the purpose of this thesis work and its significance. We shall go through some background information regarding the system that we work on, our motivation, and the overall aim of this thesis work. We provide some general information about the EPC's architecture, components and functions, while presenting the goals and impacts of the present work.

1.1 Background

EPC is part of the mobile broadband network, connecting base stations to the IP backbone and providing cellular-specific processing of traffic [1]. This framework is a critical network component that provides converged voice and data on a 4G network. In a 4G EPC, network functions are not grouped to a single network node as in a traditional or hierarchical centralized network like 2G or 3G, but they are distributed to provide connectivity where it is needed. The design of this split architecture focuses on increasing the efficiency, scalability and elasticity of the system.

One of the main components of EPC is the Packet Gateway (PG) which acts as an interface between the Radio Access Network (RAN) and other IP based packet data networks. The Evolved Packet Gateway (EPG), Ericsson's instantiation of the PG, supports Smart Services Cards (SSCs), Line cards and Switch cards. The SSCs provide modularity functions such as control handling, user packet handling, and tunneling. The line cards provide connectivity to external physical interfaces, while the switch cards provide routing, alarm management, and packet switching functionalities.

Generally, EPG provides core function of the EPC such as session management. Session management is the process of establishing and managing user sessions between the UE and an Access Point Name (APN) network. This includes activating, deactivating, and

modifying IP addresses.

1.2 Motivation

During the last decades mobile networks have been growing rapidly with the introduction of new technologies such as 5G networks, smart end devices and advanced applications. All those changes challenge mobile operators and network equipment providers to deliver the best service for their customers. The EPG is one of the main EPC components that is used for this purpose, and it is responsible to forward and control the traffic that the network can process. The upcoming increase of the traffic in the next generation of networks, entails challenging research questions, related to the ability of the current system to be capable of functioning successfully in the future. In order to identify the hardware components that could provide a certain boost to the system, we need to investigate which ones have the biggest impact and what is the potential for future improvements.

The concept investigated in this thesis work is the impact of NUMA-aware configurations and variety of CPU-pinning configurations within a virtual EPG deployment.

As the placement of main memory relative to the system cores plays a great role on the performance of the system, we have used different hardware configurations to investigate the overall packet per second processing capacity of the EPG. The packets per second is the main metric which we used as the best performance indicator of the EPG configuration. To analyze the pps processing capacity of each configuration, the EPG node uses smart cards that are installed on a Smart Service Router platform. The smart service cards use Intel-based server processors. In this thesis work [2], the effect of NUMA-aware and Uniform Memory Access (UMA) configuration is investigated on both the SSR and Commercial off-the-shelf (COTS) platforms.

1.3 Aim

Currently, a large number of requests from user equipments are processed by the EPG to connect to the internet using packet switching technology. The number of user subscriptions allowed to get the service depends on the packet processing capacity of the EPG. Considering all the other components of the EPC provide full services, the EPG is responsible for assigning IP addresses and interacting with the external packet based networks. At a time, a large number of subscriptions may rise from user equipment (UE) and more packets may be dropped if the processing capacity of the EPG is bottlenecked or if the processes are overloaded.

The packet per second processing capacity of the EPG is affected by the configuration and type of processor on the platform used for the EPG deployment. Different vendors use various processor versions with diverse processing performance and configurations. The NUMA and UMA (Uniform Memory Access) memory placement architectures are

the two main configurations of the EPG discussed in this work. These configurations affect the overall performance of the system. In this thesis work, the impact of NUMA-aware and UMA configurations of the EPG are evaluated on the SSR and COTS hardware. The main aim of this thesis is to investigate high performance configurations using Intel processors on both these platforms, and identify the components with the most significant overall effects.

1.4 Challenges

Working on such a project on a well defined system in a big enterprise, we expect to meet a number of obstacles.

- The experimentation on CPU pinning can only be executed on a virtual environment, which means testing on smaller scale hardware and comparing the statistics between the scenarios instead of actual metrics.

2
Background

This chapter provides detailed background information and a description of some components used in this project. Since EPG is one of the main components of EPC, a detailed description of EPC and its sub components are discussed in the first section. Next, EPC as a distributed system and the role of the load balancer to distribute traffic is analysed. In addition, this chapter provides some background information about some processors, NUMA concepts and our main objective of investigating the performance of all the configurations. Finally, this chapter provides a related work section that specifies the connection of this thesis work with other studies.

2.1 The Evolved Packet Core

EPC is one of the main components of the Universal Mobile Telecommunications System (UMTS) network. It is a framework for providing converged voice and data on a 4G Long-Term Evolution (LTE) network [2]. On the LTE, EPC is sandwiched between the Radio Access Network and the packet switch based external data networks. Requests coming from User Equipment (UE) to access a communication channel using packet switching and their replays are passed through the EPC. Since EPC is an all IP based network, it does not support traditional circuit switched connections that were still partially supported until 3G. The basic architecture of EPC is shown in Figure 2.1.

Figure 2.1 shows the basic architecture of the Evolved Packet System (EPS) when the User Equipment (UE) is connected to the EPC. The Evolved NodeB (eNodeB) is the base station for Long LTE radio access network. The EPC contains at least four network elements that provide different functions. The dotted line shows control signals which allow independent scaling of control and user plane functions [3]. This includes EPC components like Home Subscriber Server (HSS) and Mobility Management Entity (MME). Both the Service Gateway (SGW) and the Packet Gateway (PGW) nodes are parts of the EPG where both of them have different functions. To have a good understandings of the EPC, it is important to describe the function of each component.

2.1.1 The Home Subscriber Server

The HSS is a database that stores and controls information about user sessions and subscription related activities. It manages user data and profile information for users who

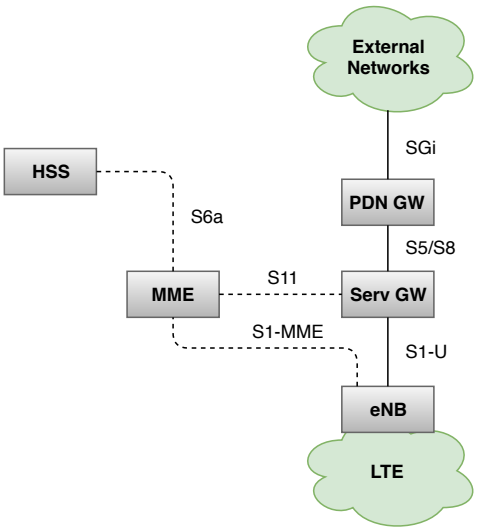


Figure 2.1: Basic EPC architecture for LTE.

are accessing over the LTE RAN [2]. It also controls other functions like session establishment, roaming, user Authentication and Access Authorization (AAA), and mutual network-terminal authentication function services.

2.1.2 The Mobility Management Entity

The MME server is responsible almost exclusively for the control plane related functions of the EPC. It handles signaling related tasks like user subscription management, session management, handovers, mobility and security of control plane functions. A handover is the process where one can get full network access while moving from one network to another and keeping the same IP connectivity. The purpose of doing handovers is to make the attachment from one network to another completely transparent for the user. Therefore, the user always stays connected no matter which network he is using.

As it is shown in Figure 2.1, the MME is linked through the S6a interface to HSS which contains all the database user subscription information [2]. The MME first collects user's subscription and data authentication information for registering in the network.

2.1.3 Evolved Packet Gateway

EPG is a gateway for the EPC to interact with the external IP-based networks. All incoming and outgoing IP packets pass through this gateway using the SGi interface. As it is shown in Figure 2.1, the SGW and PGW are connected over an interface either S5 or S8. If the user is roaming to connect to home networks, S5 interface is used to create a connection between the service gateway and the evolved packet gateway. If the user is roaming to attach to the visited LTE, S8 interface is used instead [3].

The Evolved Packet Gateway provides functions like IP address allocation, charging, packet filtering and policy-based control of user-specific IP flows. It also has a key role

in providing Quality of Service (QoS) for end-user IP services. Since an external data network is identified by an APN, the PGW will select privileged sessions to connect to the external Packet Data Network (PDN) servers on the basis of which APN the end user wants to connect to.

The Service Gateway

The SGW is the part of the EPG that deals more with session forwarding functions. It is a point of interconnection between the RAN and EPC that transports traffic from user equipment to the external network through PG. The IP packets flowing to and from the mobile devices are handled by the user plane service gateway and evolved packet gateway nodes [3]. If there is an access request that comes from the user equipment through the Evolved nodeB (eNodeB), the SGW is responsible to forward the service to the PGW node based on the privilege signal sent from MME.

Packet Gateway

The Packet Gateway (PGW) is the part of the EPG which provides connectivity to external networks and allocates an IP address to the UE. The PGW uses interfaces S5/S8 to connect towards the SGW and SGi to connect with an external network. Its job is to route, upload and download user plane data packets between the external IP networks and UE.

An important role of the PGW is performing Packet Inspection and Service Classification (PISC) by enforcing Policy and Charging Control (PCC) rules. When an event is triggered, the PGW also sends a report to Policy and Charging Rules Function (PCRF). PCRF is a software component in the node that operates at the network layer and is used to determine policy rules in multimedia networks. Some events are always reported, while others PCRF can choose to subscribe to report. Based on the triggered events, the PCRF can create, update or delete the PCC rules.

2.1.4 EPC as a Distributed System

The implementation of a single software instance of any element of the EPC, such as the MME or the SGW can become congested under heavy traffic and the hardware resources could limit the capabilities of the software. The sensible solution is splitting the incoming load between a number of identical instances of the individual component, thus deploying the entire EPC as a distributed system. Depending on the amount of traffic, it is possible to spawn new duplicates of EPC components as clusters. This elasticity is proven more capable to handle fluctuating demands, without wasting resources [4].

In Figure 2.2 we show the structure of the clusters with various instances of the same EPC components. The load balancer is responsible for distributing the traffic between the cluster's members and the synchronization of the duplicates is achieved either by copying their status in others, or by using shared storage [4].

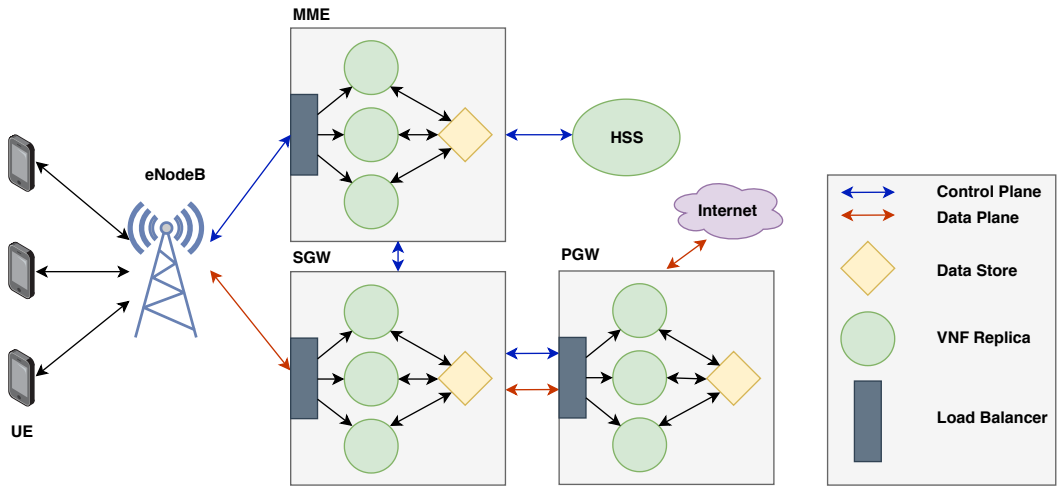


Figure 2.2: Distributed EPC [4].

In a 5G network, EPG has **Control Plane** (CP) and **User plane** (UP) functions that are flexible to deploy and can scale the CP and UP functions independently. The CP makes decisions about where traffic is to be sent, from the underlying data plane that forwards the payload traffic to the selected destination. It deals more with session management, alarm management and routing information. On the other hand, the UP deals more with packet processing, routing and inspection functions. On 3G and 4G network, the CP and UP functions are merged together in the EPG.

2.2 Non-Uniform Memory Access

NUMA is an architecture that is widely used in high-end servers and computing systems today due to its performance and scalability [5]. Multiprocessor systems have introduced challenges for compilers and run-time systems when it comes to shared memory and its contention[1].

Whenever there is conflict over access to shared resources as memory, disk or cache, buses or external network devices, we are facing contention. A resource experiencing ongoing contention can be described as "oversubscribed". Processors are currently so capable that they require directly attached memory on their socket, because remote access from another socket leads to additional latency overhead and contention of the Quick-Path Interconnect (QPI) which sits between the sockets. Since 2017 in Xeon Skylake-SP platforms, the QPI has been replaced by the Ultra Path Interconnect (UPI).

A basic NUMA architecture is shown in Figure 2.3, where each physical core is allowed to access the memory that is connected to it. Every core inside the NUMA node has its own cache memory. Different processors have different cache placement and access level strategy. Most NUMA nodes today have Level1 (L1), Level2 (L2) and Level3 (L3)

[1]Contention: Competition for resources.

caches where L1 cache is only allowed to be accessed by one core, L2 is accessed by two neighboring cores and L3 can be accessed by all the cores in the same NUMA node.

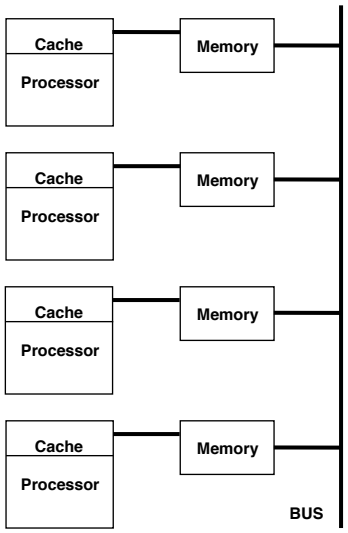


Figure 2.3: Basic NUMA Architecture.

It is important to properly place data in order to increase the bandwidth and minimize the memory latency [6] of each NUMA node. The two most important points for managing the performance of NUMA shared memory architecture are *processor affinity* and *data placement* [6]. In *processor affinity*, each process is restricted to be executed under a specific number of nearest CPUs, and for the *data placement*, each process is assigned to access a memory location connected to the NUMA node where the process is pined.

Generally, different operating systems have different ways of managing NUMA architecture, but there are many strategies used by the different operating systems to manage different NUMA configurations. Some of them are described below:

- **Heuristic memory placement of applications**
  In this approach, if the operating system is numa-aware, it is possible to enable and disable the configurations during compile time with a kernel parameter. Here, the operating system determines the memory characteristics from the firmware and it can adjust its internal operation to match to the memory configuration. This approach tries to place applications inside their local node and the memory from this local node is to be preferred as default storage. If possible, all memory requested by a process will be allocated from the local node to avoid the use of context switching.

- **Special NUMA configuration for applications**
  This approach provides configuration option for applications to change the default assumptions of memory placement policy by the operator. In this approach, it is possible to establish NUMA configuration policy for all applications using command line tools without modifying the code.

- **Ignore the difference**
  This is an initial approach which allows software and the operating system to run without any modification to the original configuration. Since this approach treats everything as equal regarding performance between configurations, the operating system is not aware of any nodes. Therefore, the performance is not optimal and will likely be different each time the application runs since the configuration will change on boot-up.

2.3 Related Work

Since performance of a system is mainly affected by latency, bandwidth and available of cores, a large amount of research has been invested in comparing and analyzing performance. Awasthi et al. [7], Cho and Jin [8] and Dybdahl and Stenstrom[9], have discussed optimizing data placement in last-level shared Non-Uniform Cache Access (NUCA) caches.

Awasthi et al. [10], developed a relevant approach to manage data placement on memory system with multiple Memory Controllers (MC). Their placement strategy is incorporated more on the queuing delay at the MC, the DRAM access latency and the communication distance and latency between the core and MC. From the methodologies they have used to investigate best data placement, they have got an efficient thread's data placement by modifying the default operating system's frame allocation algorithm. From this placement, they found 6.5% improvement when pages are assigned by first touch data placement algorithm and 8.9% when pages are allowed to migrate across memory controllers. This approach differs from ours in the way that, they are concerned more about the frame allocation algorithm, while ours impacts on network performance in large distributed systems.

As many processes in the same processor share a common cache, the issues of memory management and process mapping are becoming critical [11], [12]. Molka et al. [13] investigated a benchmark for main memory and cache identification to find out the fundamental performance property for both Intel and AMD x86_64 architectures in terms of latency and bandwidth. For both architectures they used a NUMA memory layout. Based on their identification, although the size of the L2 cache for the AMD architecture is large, it gave almost the same performance result with L3 cache bandwidth that scales better with the core count on the Intel system. The transfer rate between the socket in Intel architecture is also four times better than the transfer rate between the two dies in the AMD architecture [13].

Oltan Majo and Thomas R. Gross.[11] show that, if the allocation of physical memory and the structure of memory system is not managed well, the operating system will fail to obtain good performance on the system. From their point of view, if the memory allocation in the system is balanced, then local scheduling provides large performance benefits. If the memory allocation configuration of the system is not balanced, then map-

ping given by the maximum-local scheme needs to be modified, otherwise it is known to cause performance degradation even with relative to default scheduling. Therefore, if the distraction of system memory is not fair to all the processors, then mapping processes can lead to severe cache contention.

Hackenberg et al. [14] also did a comparison on Cache Architectures and Coherency Protocols on x86-64 Mediocre Symmetric Multiprocessing (SMP) Systems. This benchmark is done to get an effective in-depth comparison for the multilevel memory subsystem of dual-socket SMP systems based on the quad-core processors AMD Opteron 2384 (Shanghai) and Intel Xeon X5570 (Nehalem) [14]. To the best of the authors' result, the AMD's cache coherency protocol provides the expected performance advantage over the Intel's Nehalem processor for accessing modified cache lines for remote processor.

Blagodurov et al. [15], presented Contention Management on Multicore Systems. The main focus of their work was to investigate why contention-aware schedulers that are targeted to work on UMA are failing to work in NUMA, and to find an algorithm that can work on NUMA as scheduling contention-aware control. Based on the author's experimental result, one reason why contention-aware schedulers fail to work in NUMA, is that if one process that is computing for Last Level Cache (LLC) is migrated from one core to other core in different NUMA node, the process will still compute to get access for memory controller with the previous processes that were in the same core. The algorithm they have devised to solve this problem is Distributed Intensity NUMA Online (DINO) which prevents thread migration, or if the tread is migrated, the memory of this thread should also be migrated to the memory where this thread's core is connected. The evaluation of their algorithm shows that, moving thread's memory to the location where the thread is migrated is not a sufficient solution, but it is better to prevent unnecessary migrations.

Qazi et al. [1] proposed a new architecture for EPC called PEPC. On their implementation, they used the Net-Bricks platform which allows to run multiple PEPC slices within the same process. Their results show a throughput improvement 3-7 times higher than a comparable software EPCs that is implemented in the industry, and 10 times higher throughput than a popular open-source implementation. This paper work relates to ours in the way that, its main objective is also to increase the processing performance of the EPG but, our work focuses more on the EPG instead of EPC.

In general, all the above articles are related to our work, due to their focus on the impact of memory management, CPU pinning and NUMA architectures. This thesis focuses more on investigating the impact of the NUMA concept and various configurations on high-performance virtually deployed distributed systems, such as EPG, which is the test case for our experiments and benchmarks. The scalability of the Packet Gateway will be put to the test during the upcoming 5G era more than the other EPC components, and it is essential to identify the factors that could affect its performance.

3
Methodology

This chapter focuses on hardware and software methodologies used to evaluate the packet processing capacity of each NUMA configuration. The NUMA-aware and deactivated NUMA-awareness configurations on the Intel processors, and the different vEPG deployments with more CPU pinning scenarios are discussed in detail.

We proceed with system level testing to find the packet-per-second processing capacity of the EPG. System level testing is a testing technique that is used to determine whether the integrated and complete software satisfies the system requirements or not. The purpose of this test is to evaluate the system's compliance with the specified requirements [16]. In our case, the test case used to test the EPG at system level is a payload test case. We want to evaluate the packet per second processing capacity of the EPG at system level with one payload test case chosen from the available pool. Taking this as a baseline, we run a test for a NUMA-aware and a NUMA-unaware configuration on the SSR and virtual EPG (vEPG).

3.1 Studied Hardware

In this project the main processor product we used to run the EPG with different configurations is from the Intel Xeon series. These processors are installed on the smart service cards that are insulated under the SSR platform.

3.1.1 Intel Skylake

Skylake is the code name given for the project implemented in the 6th generation of Intel Core micro-architecture that delivers a record level of performance and battery life in many computing cases [17]. It is designed to meet a demanding set of requirements for various power performance points. Skylake also introduces a new technology called Intel Software Guard Extensions (IntelSGX), in which application developers can create secure code to encrypt memory so that no one can not modify or disclose it. The Skylake memory solution has an efficient and flexible system memory controller. This controller enables the processor to use Skylake System-on-Chip (SoC) on multilevel platforms using different Double Data Rate (DDR) technology.

Skylake's fabric is an extended development of the successful ring topology that is introduced in the Sandy Bridge generation [17]. It has a built-in last-level cache, that is designed to provide high memory bandwidth from different memory source. Introduc-

ing eDRAM-based memory-side cache is a main significant change in Skylake's memory hierarchy.

3.2 Traffic Modeling using the Dallas Tool

Dallas is a distributed system tool, developed by Ericsson, that can easily be scaled up to meet different load testing requirements. Dallas can simulate up to millions of subscribers, and simulate UEs and the radio network in the packet core network testing. From the system testing point of view, Dallas can be used for stability testing by running traffic for a long time, robustness testing by running different traffic models and by performing different types of failures, and capacity testing by running specific traffic models to measure the performance of the system. In this thesis work, Dallas is used as a capacity or payload testing tool.

Before sending any traffic, Dallas first sends a signal to the node to get information about the memory and CPU utilization of the node. Then, it starts sending traffic to the node based on the status of the latter. In Figure 3.1, the Dallas testing platform sends a command to generate traffic that is forwarded to EPG starting from a small number of subscribers. The command sent to the node contains the **number of sessions** and **rate** as main parameters. From the total number of sessions, rate number of session are sent in every second. Then, the **waiting time** of Dallas before sampling the payload processing capacity of the EPG is calculated as **sessions/rate + 5** seconds. If the waiting time expires, Dallas starts the sampling process which counts the number of packets processed and measures the CPU utilization of the EPG. If the CPU load has not reached its peak point, Dallas increases the traffic repeatedly with batches on new subscriptions until the EPG reaches its peak CPU utilization.

Finally, it calculates the average Packets Per Second (pps) handled by the EPG, which is our main metric, and stores the results in a log file. The reason we use pps as our main metric is because, the system is mostly loaded by handling packet headers. There is no Deep Packet Inspection (DPI) in the process and the system is more than capable to cope with large packets, so the size of the packets is not our main concern.

Dallas will stop sending traffic if the following conditions are not met:

- **Average packet per second loss**:
  During the execution of the test case, a required drop for each test case is set to some constant value. This value is used to compare with the drop ratio which is calculated as packets/1million on the test script. If the drop ratio is greater than the required threshold, Dallas stops sending traffic, otherwise it can continue sending traffic with following iterations by keeping the other conditions such as number of iterations, peak CPU and number of bearer connections (stable).

- **Number of iterations**:
  The maximum number of iterations Dallas can send as traffic to the node, is set to 11 iterations. This means that if there is not any crash or failure of the node to pass the other constraints, Dallas can increase the injection rate up to 11 times with different number of sessions to the node, and then terminate the connection.

- **Peak CPU**:
  The maximum value of the peak CPU is set to 100%. If the CPU reaches 100% as an average between all cores in any iteration, Dallas stops sending traffic to the node and collects the final result for that iteration.

- **Bearer connections**
  Bearer connections refers to the number of sessions that are created safely or deleted if the number of sessions are greater than the quantity set by the tool. If the node fails to pass one of these conditions, Dallas tries for the second time by sending traffic again while lowering the number of sessions to some a number which is multiplied with a constant multiplier that is initially set on the test script. If the node still fails to pass the above conditions, it stops sending traffic and outputs the result to the log file at that iteration. If the node fulfills all the conditions, Dallas continues sending traffic to the node by multiplying the number of sessions by the constant multiplier 1.02 for SSR or 1.1 for the vEPG.

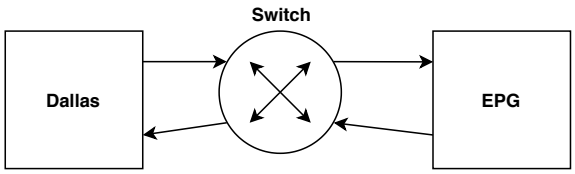


Figure 3.1: Traffic flow between Dallas and EPG.

3.3 Baseline Configuration (NUMA-aware)

The default configuration of the EPG is NUMA-aware. Processes are allowed to access specific memory locations connected directly to the NUMA node they are running on. Based on their application, the processes are classified in three groups. We call them a-processes, b-processes and c-processes. A-processes are used to forward the packets, b-processes to distribute the coming packets to a-processes, and c-processes are used to control the communication between them. Figures 3.2 and 3.3 show a NUMA-aware configuration of the EPG on SSC1 and SSC3 cards respectively.

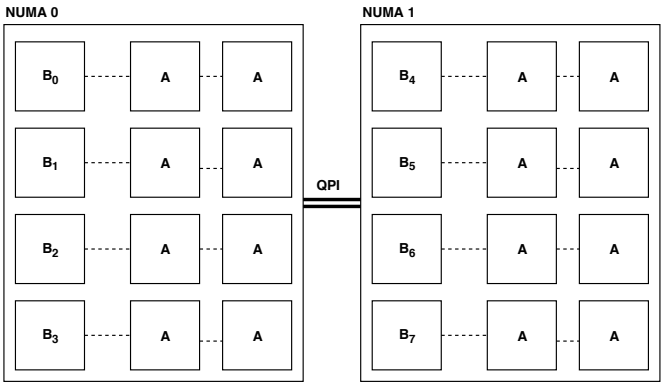


Figure 3.2: NUMA-aware EPG Configuration for SSC1.

The SSC3 card architecture follows the same concept in a 4-socket motherboard. As shown in figure 3.3 we have one NUMA node for every socket and it contains 4 b-processes. Every CPU model has 14 physical cores and 28 threads, for a total of 112 processes across the whole motherboard. The operating system installed on both the SSC1 and SSC3 cards is x86_64 architecture GNU/Linux. Both these cards are installed on the Smart Service Router (SSR) platform.

The b-processes can only distribute packets to the a-processes sharing the same NUMA node. In the SSR platform, b-processes have dedicated groups of a-processes under their command, and by default are not allowed to use a-processes belonging to another group. This means that for the SSC1 card, there is only one b-process per 5 a-processes in a group as it shown in Figure 3.2. This b-process is allowed to send the packets to these a-processes. In the SSC3 card, there are two b-processes and 9 a-processes in a group, in which the two b-processes can distribute the packets among the 9 a-processes. In the vEPG, this feature does not exist and the only bound is the node.

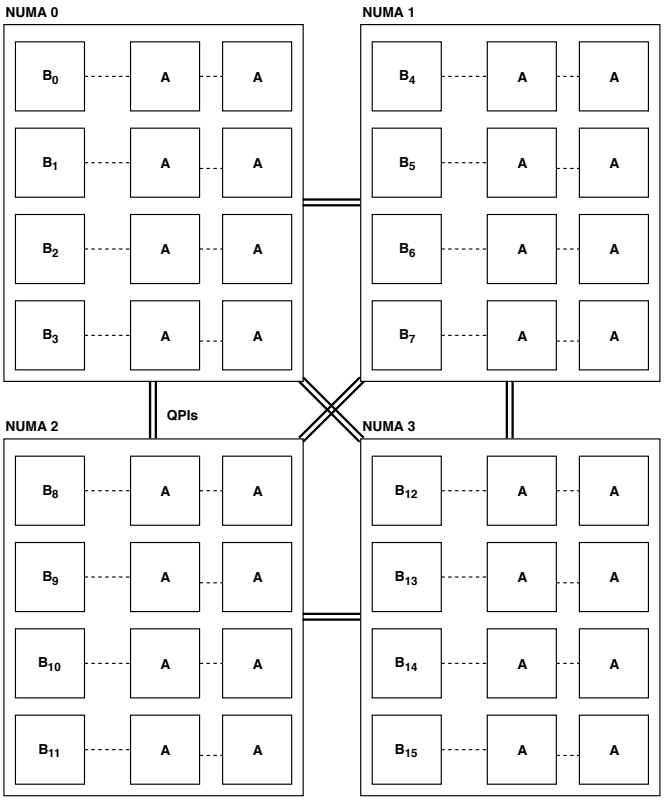


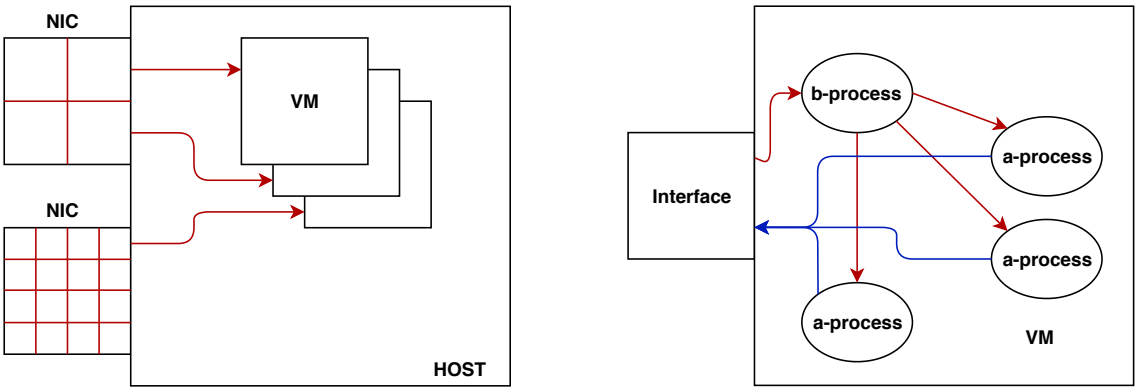
Figure 3.3: NUMA-aware EPG Configuration for SSC3.

For the case of Uniform Memory Access (UMA) configuration, the NUMA-awareness for both the SSC1 shown in Figure 3.2 and SSC3 shown in Figure 3.3 is deactivated. This means b-processes are free to send the packets to any of the a-processes on the card and there is no dedicated memory location for specific cores. It is the responsibility of the scheduler to assign which process should use which memory region.

3.4 Virtualization

Virtualization is the process of creating a virtual version of an entity, including but not limited to virtual computing, storage and networking resources. As the SSR platform is hard coded for a specific CPU pinning configuration, enabling and disabling the CPU pinning on the EPG source code doesn't bring any change. To investigate different results from different CPU pinning strategies, virtualization is the preferred option as it gives us the freedom to pin CPUs to different NUMA nodes and group them. To virtualize the EPG, Virtual I/O (VIRTIO) and Single Root I/O (SRIO) interfaces are used in both Intel servers.

As mentioned previously, our work continued on the vEPG, which can only be deployed on SSC1 cards. This means that we proceed with a smaller scale platform but with a dedicated node for our experiments only. We deployed on one SSC card only, so that we can manipulate the CPU pinning configuration.



(a) VM deployment in host. **(b)** VM function.

Figure 3.4: V-EPG in physical hosts.

Figure 3.4a shows, the way EPG Virtual Machine (VMs) are deployed in a host. A VM is an emulation of a computer system where a physical computer can be partitioned into several software based VMs. A VM can run its own OS and applications, contain specific number of CPU cores, dedicated RAM, hard disk and Network Interface Card (NIC). The physical Network Interface c-processess (NICs) can be split into smaller virtual ones, and dedicate each of them on one VM. Currently it is possible to simulate up to 64 NICs on one physical. Figure 3.4b shows a simplified way, the role of the b-processes and a-processes in the VM. The b-processes receive the tasks (packets) and forward them to the a-processes for processing. Currently each a-process is pinned to a specific core and the amount of available a-processes is equally distributed to them. For every b-process, there is a specific pool of a-processes that are allowed to access the memory connected to them, and this is how they form a NUMA node in SSC1 and SSC3.

On the SSC1 we have 32 vCPUs available and on the SSC3 there are 112 vCPUs, but not all of them are allocated by the a-b-c-processes. Since we cannot alter the number of free vCPUs on SSC1 and SSC3 on SSR, we will do that in the vEPG where we have the freedom to change the NUMA placement. On the vEPG, we will consider different CPU pinning, so that the b-processes are assigned to a core automatically. We will try a larger number of b-processes in a NUMA node in order to evaluate if those processes are a bottleneck while distributing the packets to the a-processes, and a larger number of a-processes by utilizing the free vCPUs. Eventually the main purpose of all this tests on configurations is to identify which parameters have the biggest impact on performance and why.

3.4.1 vEPG deployment using VIRTIO

The deployment of vEPG in Cloud Execution Environment (CEE) with OpenStack for a lab using VIRTIO interface is shown in Figure 3.5. OpenStack is a cloud computing operating system that is used to deploy virtual machines and other instances to control different tasks for building and managing public and private cloud-computing platforms [18]. The OpenStack is responsible for controlling the visualization process. The VM that is used to deploy vEPG has 48 vCPUs with two sockets and one NUMA node on

each socket. It is x86 architecture with GNU/Linux hyper-threaded enabled. Since there are only 48 vCPUs on the host machine, the total number of vCPUs on all the VMs combined, should not exceed that number.

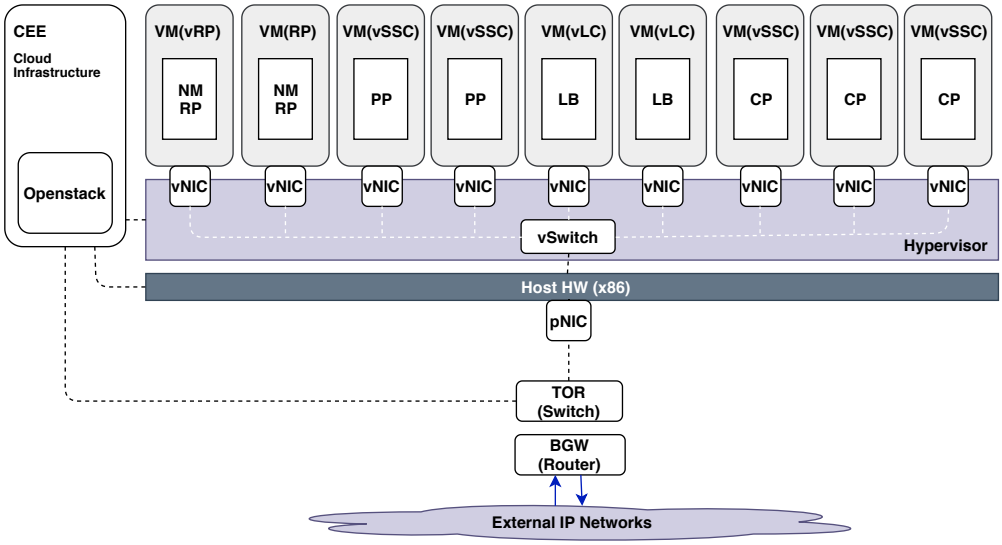


Figure 3.5: vEPG virtualization using VIRTIO interface.

The deployment process starts by generating a Heat Orchestration Template (HOT) file using a python script HOT file generator. Orchestration is the process of creating one or more virtual machine at a time. Next the image is downloaded using the Virtual Deployment Package (VDP) to the glance from any EPG build. Glance is component that provide services to the OpenStack. Using this service, a user can register, discover, and retrieve virtual machine images for use in the OpenStack environment. The images that are deployed using the OpenStack image service can be stored in different locations like OpenStack object storage, file system and other distributed file systems [18]. Then, the flavors are created and the HOT template is executed to generate and create the Openstack resources. Flavors are defined on the configuration file to set the vCPU, memory, and storage capacity of the virtual machines. During the deployment process, vCPUs, Disk and main memory of the VMs are created based on the definition of the flavors on the configuration file.

The deployment of vEPG from the default configuration file is shown in Figure 3.5. Here, nine VMs are created with 2 of them used for Payload Processing (PP), 3 of them used as CP, 2 of them as Route Processing (RP) and two of them as a Line Cards (LCs). Line cards are used to send the out-going packet from the EPG to the PDN servers or the in-going packets from the EPG to the UE devices. The RPs are used to manage and facilitate the communication between the VMs. The default deployment configuration, assigns 6vCPUs for the control and user plane VMs. In each of the user plane vCPUs, one of them is used as a-process, one as a b-process and 4 of the are left for the background process of the VM.

Deploying vEPG using VIRTIO interface is simple and flexible to change the VM con-

figuration. It is possible to change the number of CPs or UPs from one role to another and to pin the CPU to different NUMA nodes. But, since the hypervisor process creates a Virtual Switch (vSwitch) from the Top Of Rack (TOR) physical switch, it slows the traffic that is forwarded to the b-processes. The bandwidth of the vSwitch is limited to process a maximum of 10 Gbps, and the traffic that passes through this vSwitch can not overload the a-processes of the VMs to reach their max CPU utilization. Even if this virtualization is is not recommended to test EPG at a system level, we decided and proceeded with testing our different configurations by using 50% average CPU utilization as the reference point, by continuing to send the same number of sessions and rate for all the configurations.

Since our main metric is packets per second for comparing the performance of each configuration based on CPU utilization, we selected a test case that works with fixed CPU utilization. By fixed CPU utilization, we mean that Dallas stops sending traffic to the node if its CPU utilization reaches between +2% or -2% of the specified utilization percentage. The test case we use was designed to work with a fixed CPU utilization of 27% using 2CPs and 2UPs with some constant initial sessions. This means that Dallas stops sending traffic if the CPU utilization of the node reaches a value between 25% and 29% in any of the iterations.

Therefore, since the default test script configuration of this test case does not match with our requirement, we changed all the parameters of the test script and the conditions to match our desired 50% fixed CPU utilization.

3.4.2 vEPG deployment with 8vCPUs UP on each NUMA node

For our customized default configuration with 8vCPUs VMs, the setup is presented in Figure 3.6. Every square represents one vCPU on the VMs. The reason we chose earlier to proceed with 8vCPU VMs is because of the number of a-processes in smaller scale deployments. As we mentioned before, 6vCPU deployment UP instances have one b-processes and one a-processes deployed on each socket, which does not allow splitting a-processes of a single user plane between NUMA nodes.

If we check the properties of the host, we are presented with the NUMA nodes and the IDs of the processes they include. This gives us the ability to map the location of every thread. We can see the instances deployed and the vCPUs they contain, and also identify the b-process and a-process threads. In this test case, we identify where every VM is running and manually change the IDs of the processes it uses to customize the topology. By changing all the vCPUs a VM uses, we are able to practically move it wherever we want in the system. We are basically telling the VM which vCPUs to use. This is going to be our new baseline for all the rest of the configurations. Every VM is deployed on a specific NUMA node (socket) with 1 CP and 1 UP on socket 0, and 1 CP and 1 UP on socket 1.

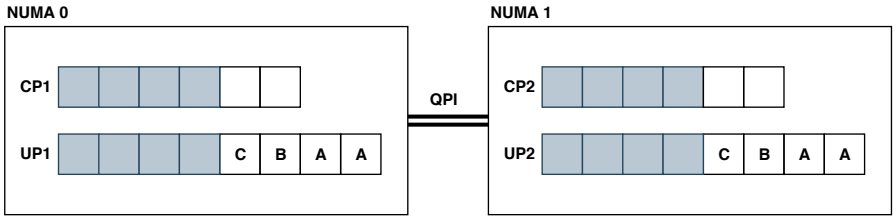


Figure 3.6: vEPD deployment with 8vCPUs on UPs (result 4.3).

The user plane VMs have 1 c-process, 1 b-process and 2 a-processes. The c-process is a sibling of the first UP a-processes and the b-processes is a sibling of the second UP a-processes. This deployment achieved better performance because the VMs with a-processes do not exchange data between sockets and work independently.

CPU Pinning Scenarios

Following the test on NUMA-awareness in the system, we want to experiment with the deployment of the Virtual Machines. The virtualization allows us to experiment with CPU pinning, something that was not possible in the SSR platforms as it is hard coded. The performance degradation in NUMA-unaware systems is mainly a result of processes requesting memory from non-local sockets, thus having to traverse the QPI bus. This method adds latency to the run-time of the process.

There are a variety of combinations we could test with CPU pinning, but we will proceed with the following scenarios for this configuration, since according to our research these introduce the most major diversities in the system and performance.

Pinning a-processes to another NUMA-node

In this configuration, the a-processes of each user plane VMs are separated from the rest of the vCPUs and are pinned to the other NUMA node. By default, one a-process of each UP is a sibling of c-process and the other a-process is also a sibling of the b-process. On this scenario, both the a-processes for both the user plane VMs are migrated to the other NUMA node as shown in Figure 3.7.

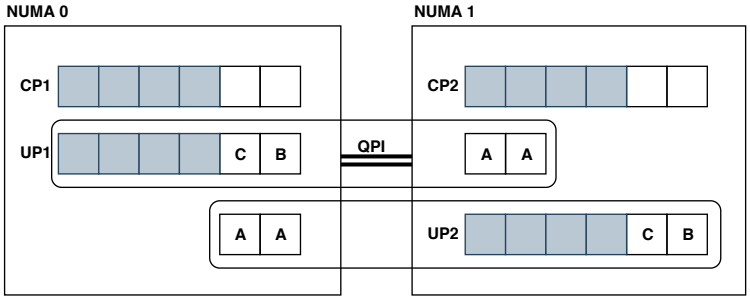


Figure 3.7: A-processes on separate NUMA node (result 4.4).

The purpose of this scenario is to evaluate the configuration of two a-processes when they are configured as sibling processes, by separating them from the rest of the vCPUs of the VM. Since the b-processes of each user plane VM is in a different NUMA node than the a-processes, communication implies traversing the QPI to distribute the packets to the a-processes, and this may cause a performance degradation with respect to the baseline configuration in section 3.4.2. Even if the two a-processes are siblings, the result of this configuration may result in high performance degradation compared to the baseline configuration, as the latency to distribute the packets is worse than the baseline configurations.

Pinning b-processes and one a-processes to other NUMA-node

In the configuration shown in Figure 3.8, we want to test how forcing the b-processes and one a-processes of the UP to work with the QPI bottleneck will affect the final results. On this scenario, the sibling b-processes and one a-processes of each UP VMs are separated and pinned to the other NUMA node from the rest of the vCPUs. Therefore, the latency between the the b-processes and one a-processes remains the same as the default configuration since they are pinned as sibling process to the other NUMA node, but the latency between the b-processes and the other a-processes as well as the latency between the a-processes will increase as they are in different NUMA nodes.

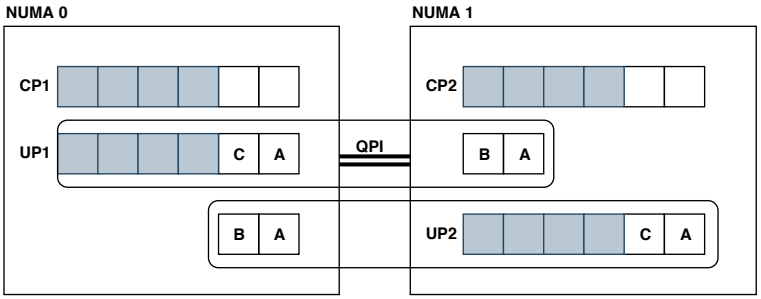


Figure 3.8: One b-process and one a-process on separate NUMA node (result 4.4).

The task of the UPs is the most CPU-intensive in the system and we expect a significant amount of negative impact and packet loss and the overall performance of this conjugation may be worse than all the pinning scenarios and the default configuration in section 3.4.2

vEPG deployment with 3CP and 2UP VMs

Figure 3.9 is used to describe the VM setups for the default 6vCPU configuration. When the EPG is deployed on a virtual machine, the default configuration on the 48core host creates 5 VMs, where 3 of them are Control Plane and 2 them User Plane instances. Each VM though has 6 vCPUs available as shown in Figure 3.9. In a 6core User Plane VMs, one vCPU is used as a-process, one vCPU as c-process and b-process and the rest are used for line cards.

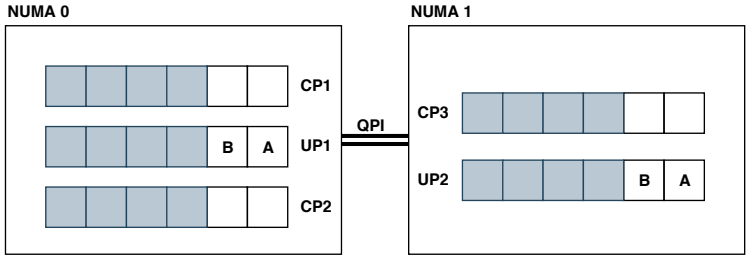


Figure 3.9: vEPG deployment with 3CPs and 2UPs (result 4.5).

On this deployment, two control plane and one user plane VMs are deployed on the first NUMA node and one control plane and one user plane VMs are deployed on the second NUMA node. This deployment is not efficient based on the usage of vCPUs, memory and a hard disk of the host machine. Since control plane VMs are not that much overloaded to give routing information for the user plane VMs, deploying three CPs on one NUMA node is almost a waste of resources.

vEPG deployment with 2CP and 3UP VMs

Figure 3.10, shows a vEPG deployment with 2 control plane and 3 user plane VMs with 6 vCPUs each. This deployment is the modified version of the 3CP and 2UP deployment configuration. There are two user plane and one control plane VMs on the first NUMA node and one control plane and two user plane VMs on the second NUMA node.

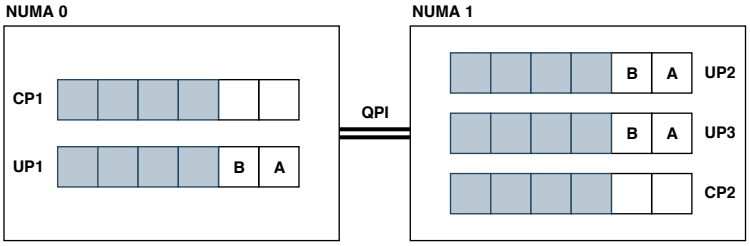


Figure 3.10: vEPG deployment with 2CPs and 3UPs (result 4.6).

The main objective of this deployment is, to compare with the default 6vCPU configuration, and we found that this deployment is better than the one in section 3.4.2, due to this deployment having more user plane VMs than control plane VMs. Since packet forwarding and processing burden is more on the user plane VMs than on the control plane VMs, having more user plane VM is a good option to avoid overloading the b-processes and a-processes.

3.4.3 vEPG deployment with 2CP and 2UP VMs

For this deployment, the configuration file is changed to deploy two control plane VMs in the first NUMA node with 6vCPUs each and two user plane VMs on the second NUMA node with 8vCPUs each as shown in Figure 3.11. In this deployment, we want to separate the Control Plane VMs from the User Plane. The communication between the control plane and user plane implies traversing the QPI but the the communication between user

planes does not traverse the QPI as they are on the same NUMA node. Since communication between control plane and user plane VMs is more intensive than communication between user plane VMs, this configuration impacts the performance negatively more than the default configuration in section 3.4.2.

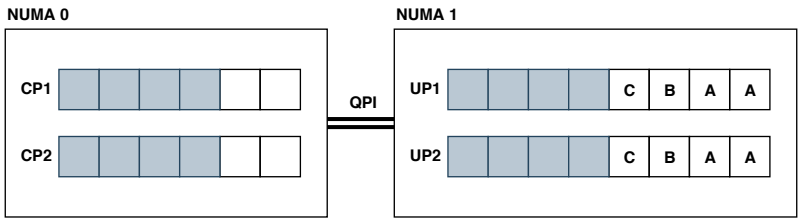


Figure 3.11: vEPG deployment by separating CP and UP VMs (result 4.7).

On this configuration, there are a total of two b-processes and four a-processes on the same NUMA node. Even if it is easy for the line cards to distribute the packets to the b-processes on the same NUMA node and there is less latency between the user plane VMs, the signaling communication between the control plane VMs and the user VMs are more costy. This configuration showed slight performance degradation compared to the default configuration, since all signals for communication traverse the QPI.

3.4.4 vEPG deployment with 2CP and 1UP VMs

This deployment is almost the same as the deployment in section 3.4.3 as shown in Figure 3.12. This configuration creates 2 control plane VMs on the the first NUMA node with 6vCPUs each and one user plane on the other NUMA node with 16vCPUs. The user plane has 16vCPUs out of which 2 of them are b-processes, 9 of them a-processes and one c-process. This deployment saves the memory and hard disk of the host machine that was allocated for the second user plane by the flavors as in section 3.4.3.

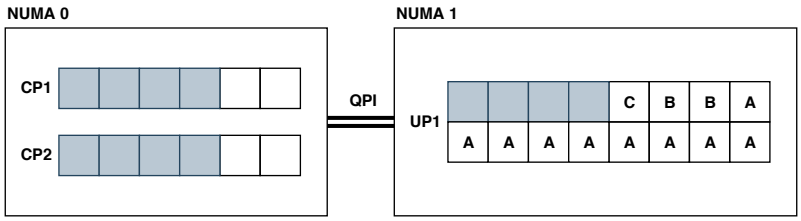


Figure 3.12: vEPG deployment with 2CPs and 1UP (result 4.8).

The objective of this configuration is to evaluate the EPG by deploying one user plane to use most of the vCPUs as a-processes and compare with the default configuration that deploys two user planes. On the default configuration in section 3.4.2 and in section 3.4.3, out of the total 16vCPUs on the user plane VMs, only 4 vCPUs are assigned as a-processes and 4 vCPUs as line cards. But, this deployment assigns 9 vCPUs as a-processes as it only uses 4 vCPUs for line cards. This configuration performed better than the default configuration since there are 5 more a-processes than the 2 user plane

VMs deployment, but there might be more packet drops as the signaling communication between the control plane and user plane VMs traverses the QPI.

3.4.5 vEPG deployment with 1CP and 2UPs VMs

This configuration deploys one control plane with 12vCPUs on the first NUMA node and two user planes with 16vCPUs on the second NUMA node as shown in Figure 3.13. Each user plane VM has 1 c-process, 1 b-process and 2 a-processes. On this configuration one control plane is used to send the routing information for both user plane instances. This configuration saves memory and hard disk on the host machine that was allocated for the other control plane VM during the 2 control plane configurations.

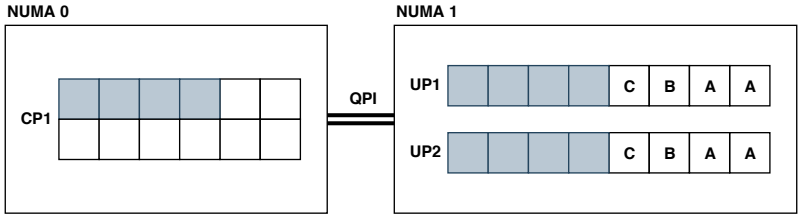


Figure 3.13: vEPG deployment with 1CP and 2UPs (result 4.9).

The objective of this configuration is to evaluate the EPG with a single control plane instance. Even if there are 12vCPUs on the control plane VM, the performance of the EPG may not increase by adding more vCPUs to a control plane, since these VMs are not overloaded by signaling packets. In this configuration it is not efficient to control both user plane instances using one control plane. Therefore, the CPU utilization of this configuration is much worse than the baseline configuration, as there may be more packet drops on the cards due to routing errors.

3.4.6 vEPG deployment with 1CP and 1UP VMs

This configuration deploys the control plane VM with 12vCPUs in the first NUMA node and the user plane VM with 16vCPUs in the second NUMA node as shown in Figure 3.14. This configuration saves almost 50% of the memory and hard disk that was allocated for the control plane and user plane VMs during the baseline configuration. The user plane has 16vCPUs out of which 2 are b-processes, 9 of them are a-processes and one c-process as in the Figure 3.4.6.

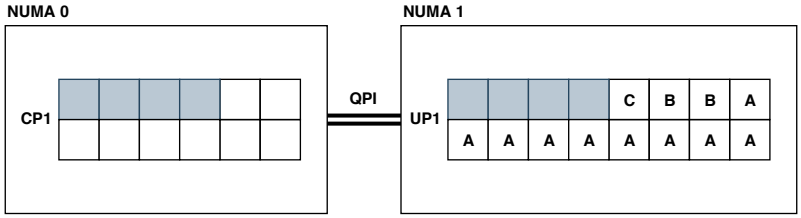


Figure 3.14: vEPG deployment with 1CP and 1UP (result 4.10).

This deployment is almost the same as the deployment in section 3.4.4, but this configuration saves the memory and hard disk of the host machine that was allocated for the other control plane VM on the baseline configuration, since it deploys with only one control plane. Since the signaling communication is only between the one control plane and one user plane VMs, the result is almost the same as the configuration in section 3.4.4.

4
Results

In this chapter, the performance assessment of the different configurations discussed in the methodology chapter is presented in depth. First the results for the baseline configurations on the SSR platform with Intel processors for both NUMA-aware and deactivated NUMA-awareness configurations are discussed. Next, the results for the different configurations that deploy EPG virtually (vSSR) are explained.

Finally, there is a discussion section based on the comparison of the results of all configurations. For the purpose of non-disclosure of sensitive information, all the actual packet-per-second values processed by each configuration are not revealed. On the y-axis percentage is used to compare the result of each configuration in relation to the baseline configuration on all the tables.

4.1 Evaluation

In this thesis work, the average CPU utilization is the main metric of comparing performance between different configurations. A configuration with lower CPU utilization is considered more efficient for the same load. On the SSR, 100% is the target CPU Utilization to identify and evaluate the impact of NUMA-aware and NUMA-unaware configurations on the EPG. On the virtualized deployments, 50% is taken as a target average CPU utilization, even for the baseline configuration. The average CPU utilization of each configuration on the final iteration is compared with the average CPU utilization of the baseline on that iteration for the same number of sessions. Then, the average CPU utilization difference between the baseline and any configuration is considered as performance improvement or degradation.

During the Dallas simulations, extensive and detailed logs are generated for the performance of each configuration tested. Comparing between these outputs is based on the main metrics provided by the tool and the ones that have been used in this study are listed bellow:

Sessions

As mentioned previously, the amount of sessions is directly connected to the number of simulated users connected to the system. There is a stable ratio between sessions and number of packages simulated, thus it cannot be revealed. Their number is increased in every iteration to achieve the desired load (sessions).

Packets per Second

The main metric of every simulation is the amount of packets the gateway can handle before its breaking point is reached.

Throughput

EPG's throughput allows further investigation based on its bandwidth and can add great value on a stress test. Since the size of packets in the simulations varies and because the EPG is more than capable of withstanding loads higher than the ones achieved in our tests, we present this result without relying on it for performance evaluation. Focusing on processing the package headers was of greater importance than the package context, and the system bottleneck we are studying.

CPU Utilization

At any given step of a simulation the percentage of CPU utilization is known. It is a reliable representation of the efficiency of the configuration and reveals the real bottleneck of the system in every occasion. Two fields relative to CPU utilization are presented in the result tables. The average CPU reveals the utilization percentage on a stable state for that given load, while peak CPU is the value documented on the breaking point.

Packets dropped per million

It is essential for the deployed EPG to meet a high robustness and reliability level. In order to maintain QoS the threshold for acceptable packet drop is kept low. This is the reason that this number is also taken into account for the success or failure of a test. Dallas has the ability to take measurements on both sides of the gateway because it acts both as the sender and receiver of the payload. Keeping record of the sent and received traffic allows for easy calculation of the amount of discarded packets.

4.2 Baseline Configuration on SSR

The test results for NUMA-aware configurations on SSC3 cards when the CPU reaches its peak point, are presented in Table 4.1. On the SSR platform, every configuration is tested twice to verify its stability. As shown in Table 4.1, an amount of X packets is processed in the final iteration of the first test. The number of sessions sent to the node on both tests is identical. The same for the second test, the EPG processed 0.09% more packets per second over the first test. For the NUMA-aware configuration, Dallas initially sent 2,500,000 sessions to the EPG node on both tests. At this point, the peak CPU of the node was 83% for both tests. For every following iteration, Dallas multiplies the number of sessions sent to the node by a factor of 1.02 until the node fails to pass the conditions discussed in section 3.2.

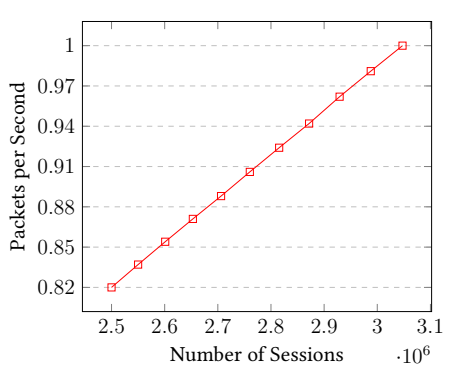
Figure 4.1 shows the test result for the NUMA aware configuration on the SSC3 cards. The relationship between session and pps is plotted on 4.1a, where the number of sessions are shown on the x-axis as an independent variable and pps on the y-axis with

relative values. Even if the packets per second are hidden to not reveal the real data on the y-axis, it is clear that the processing capacity of this configuration is increasing as the number of sessions increases.

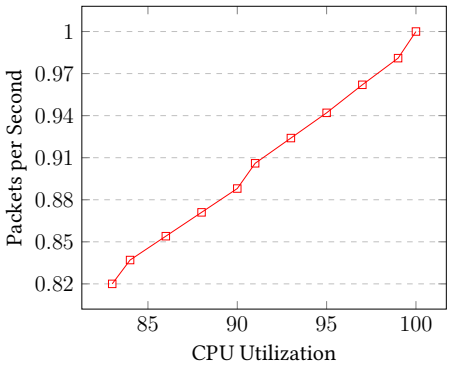
Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
2987700	baseline	baseline	90%	99%	baseline	11
2987700	+0.092%	+0.07%	90%	99%	+50%	11

Table 4.1: NUMA-Awareness processing result using Intel processor on SSCs.

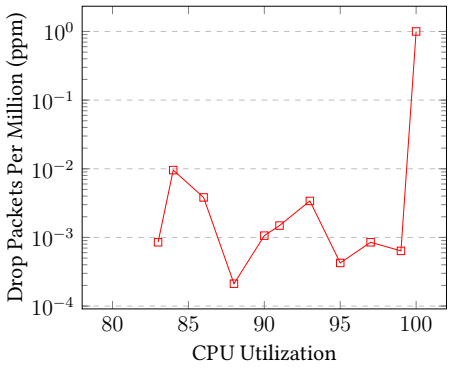
In every iteration, Dallas collects the average number of pps, average number of dropped packets per million (ppm) and peak CPU utilization of each iteration. If the node passes the conditions discussed on the Methodology Chapter section 3.2, Dallas continues to send traffic to the node by multiplying the number of sessions with the constant multiplier 1.02 as shown in Figure 4.1a and Figure 4.1b, until the node fails to pass the conditions. For the test results shown in Figure 4.1c, the drop ratio is almost stable before the peak CPU reaches its peak point, but the dropped ppm raises rapidly when the number of sessions is increased to a number that makes the CPU reach 100%. At this iteration, Dallas stopped sending traffic to the node since the a-processes are overloaded and the drop ratio is increased dramatically. Finally, all the processing capacity results of the EPG when the CPU of the node reaches its peak point are stored in a log file.



(a) pps with number of sessions.



(b) pps with CPU utilization.



(c) Drop ppm with CPU utilization.

Figure 4.1: Baseline configuration results on SSR.

To determine the packet-per-second processing capacity of the EPG without NUMA-awareness, we have deactivated the NUMA-aware configuration of the EPG. As the results show in Table 4.2, the peak CPU reaches 100% in the first iteration for both tests. When Dallas sends traffic to the node with initial number of sessions at 2.7M, the peak CPU and the average CPU utilization on both tests reaches 100% automatically. This means, both the b-processes and a-processes are overloaded and there are more packets lost per million as shown in Table 4.2. Dallas sends traffic with a lower number of sessions less than the previous iteration to give a second chance to the node, but still the peak CPU is 100%. Therefore, if the node failed to process the required packets per second, Dallas stopped sending traffic, as the node drops a high number of packets per second. For the second deactivated NUMA-awareness test, almost the same result is produced as shown in Table 4.2

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
2382400	baseline	baseline	100%	100%	baseline	2
2382400	-0.13%	-0.17%	100%	100%	+24%	2

Table 4.2: Deactivated NUMA-Awareness processing result using Intel processor.

4.3 Virtualization

As discussed in section 3.4, the virtualization which deploys the EPG in a virtual environment and the different results for the vEPG using different configurations are presented here. The bar chart in Figure 4.2 shows the relationship between number of sessions and packet-per-second processing capacity of the node for the different configurations. The blue and red color bar charts represent the number of sessions and packet-per-second processing capacity of each configuration on the last iteration. Dallas sends a fixed number of packets per second in one session with the same number of sessions for all the configurations.

For every configuration, 3.2 x 10^5 sessions are sent in the first iteration but each configuration processed different number of pps. For every configuration, the number of sessions for the next iteration is multiplied by a constant number 1.1. As it is shown in the Figure 4.2, there are 7 configurations which are named in the x-axis. Each name represents one of the configuration we discussed in the methodology chapter. The name "baseline" refers to the baseline configuration we have discussed in section 3.4.2. The name "3cp2up" refers to the configuration discussed in section 3.4.2 with 3 control plane and 2 user plane VMs. The name "2cp2up" refers to the configuration discussed in section 3.4.3 that deploys the two control plane VMs in the same NUMA node and the two user plane VMs in the second NUMA node. The names "2cp1up", "1cp2up", "1cp1up", and "3cp2up" refers to the configurations discussed in section 3.4.4, section 3.4.5, section 3.4.6, and section 4.5 respectively. The names in Figure 4.3 and Figure 4.4 also refer to the different configurations discussed in the methodology chapter.

Looking at the results in Figure 4.3, we observe that each configuration processed different amount of packets per second when the average CPU utilization of each config-

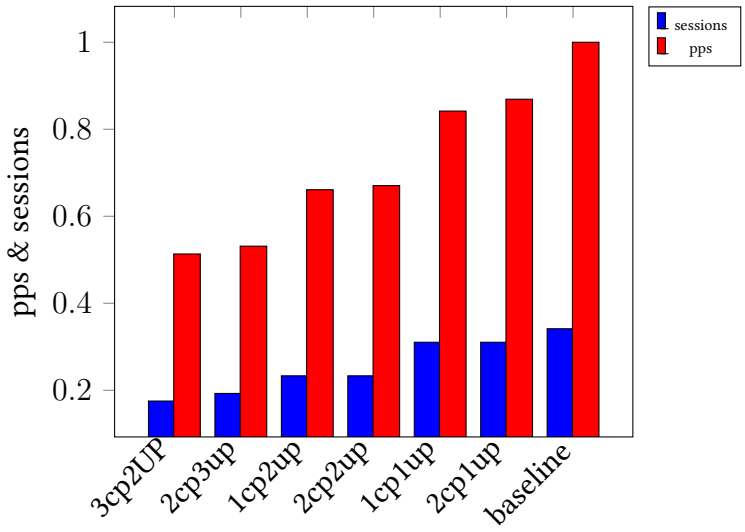


Figure 4.2: Maximum packets per second results for all configuration.

uration reaches its breaking point. The result of each configuration is plotted with different shape and color. As shown in Figure 4.3 with the square shape and red color, the baseline configuration reaches an average CPU utilization of 20.2% in the first iteration and 49.3% in the last iteration. In Figure 4.3, the plot for the 2cp1up and 1cp1up configurations overlapped since both these configurations reached the same average CPU utilization on their first and last iterations, as it is shown in the chart in Figure 4.2. But, the configuration with 2cp1up processed more packet than the 1cp1up configuration. The same case with the 2cp2up and 1cp2up configurations, as they are overlapped on the plot, since both these configuration reach almost the same average CPU utilization on their first and last iterations.

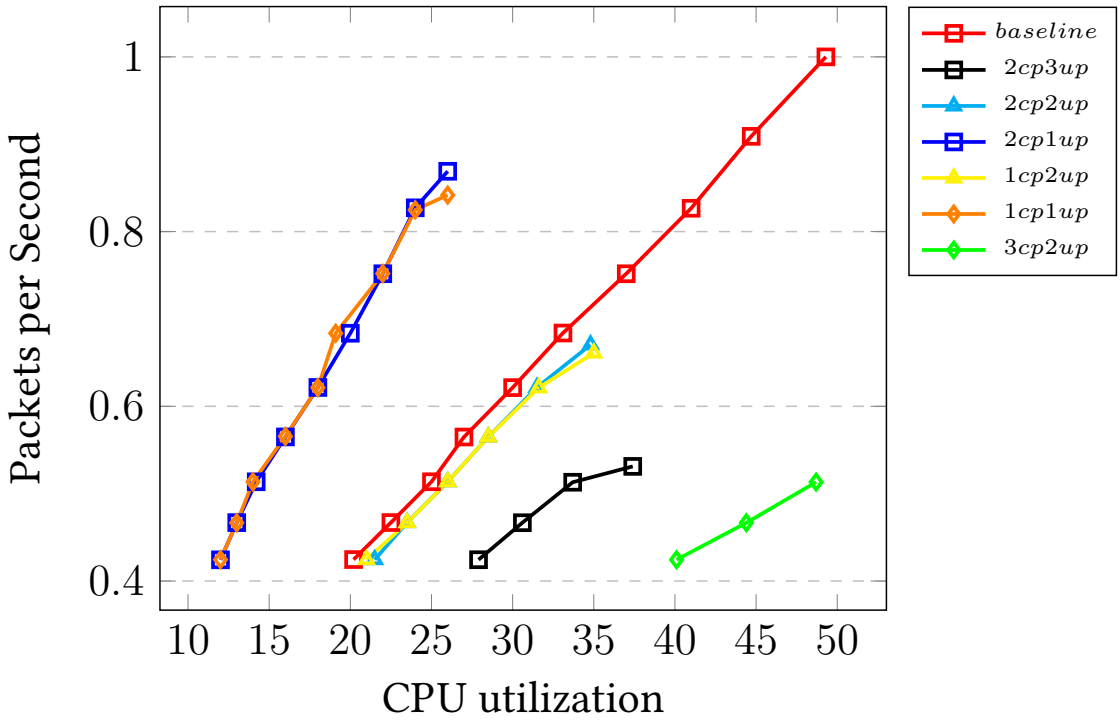


Figure 4.3: Packets per second with CPU utilization.

The relationship between average CPU utilization and packets per million dropped are shown in Figure 4.4. The drops are either on the vSwitch or on the different VM cards. Since the bandwidth of the vSwitch is 10Gbps, there are more packet drops on the vSwitch if the bandwidth of the EPG is greater than 8Gbps. There are also more packet drops on the VM cards if the number of sessions sent to the node and the placement of VMs is not configured correctly. As shown in Figure 4.4, there is a threshold level that limits drops packet per million a node can drop to continue to the next iteration. The threshold is set to 2000 packets per million. If a configuration drops more than 2000 packets per million in one iteration, Dallas stops sending traffic to the node and this drop is recorded as the final step of this configuration. The packet-per-second processing capacity, average CPU utilization and total packet drops per million of each configuration are discussed in details in the following sections.

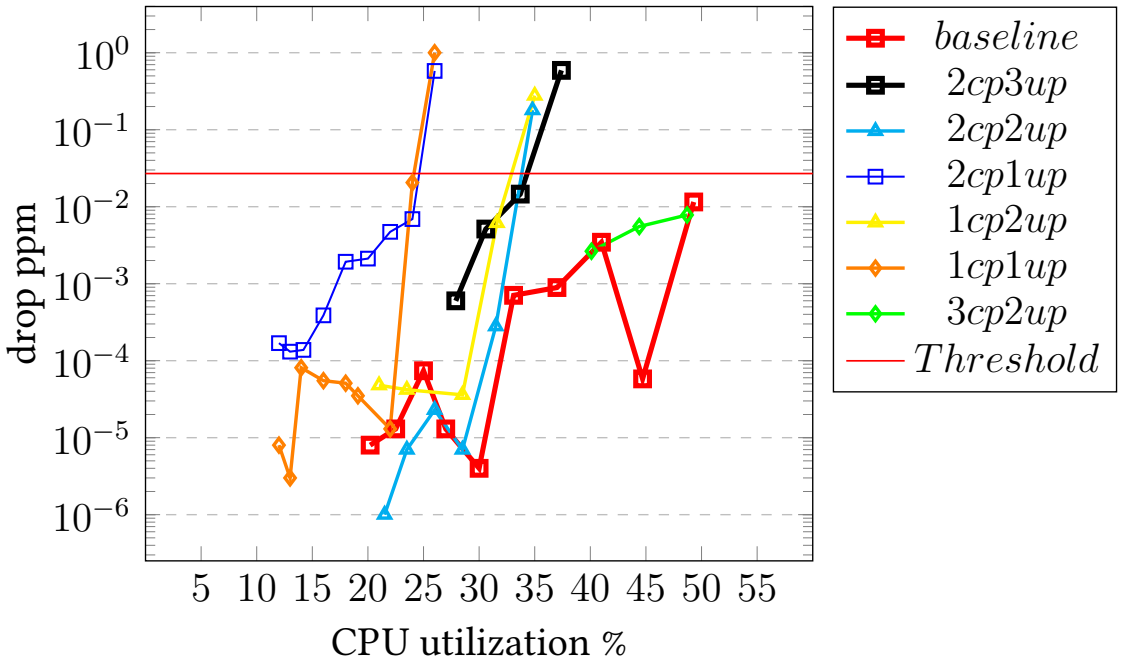


Figure 4.4: drop packets per million vs CPU utilization.

4.3.1 vEPG deployment with 8vCPU UP on each NUMA node

The result for the deployment presented in section 3.4.2 with a total of 8vCPUs in each user plane VM is shown in Table 4.3. Since memory and vCPUs of each VM are defined in the configuration file during the deployment process, deactivating the NUMA-aware configuration in virtualization does not affect the performance of the EPG. To check this condition, we run a test by deactivating the NUMA-aware configuration of the EPG, and the result is almost the same as shown in Table 4.3 for both the NUMA and UMA configurations. Therefore, since the rest of the CPU pinning scenarios and different configurations of the virtualization are tested based on the NUMA-aware configurations of the EPG, we have taken the NUMA-aware result shown in Table 4.3 as our baseline.

	Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
NUMA	754600	baseline	baseline	49.3%	55%	baseline	10
UMA	754600	-0.13%	-0.17%	48.5%	54%	+24.04%	10

Table 4.3: Results for scenario 3.6.

The packet-per-second processing capacity and number of sessions on the last iteration of this configuration is shown in the bar chart Figure 4.2 with the name baseline. This baseline configuration processed some amount of packets when the number of session are 754600. The relationship between CPU utilization and packet-per-second processing capacity of the node on this configuration is shown in Figure 4.3. In Figure 4.3, the pps processing of the node increases linearly as the average CPU utilization increases. When the average CPU utilization of the EPG reaches the target 50% CPU utilization as shown

in Figure 4.3 with the name "baseline", Dallas stopped sending traffic to the node. Figure 4.4 with square shape and red color, shows the total packet-per-million drops of the node in each iteration. For the first five iterations, the packet loss was very small and the node was stable, but when the average CPU utilization of the CPU becomes greater than 40%, the drop ratio increases to high packet loss. This is because, the vSwitch we have discussed in section 3.4 is becoming a bottleneck. Therefore, when the bandwidth of the node approaches the 8Gbps mark, there are more packets dropped on the vSwitch.

CPU Pinning Scenarios

On the baseline configuration, two pinning scenarios are tested. The results for the pinning scenarios discussed in section 3.8 and 3.7 are shown in Table 4.4. For pinning one b-process and one a-process to another NUMA node, the average CPU utilization reaches the target CPU when the number of sessions reached 515,400 in the 6^{th} iteration. For the second scenario pinning the two a-processes to the other NUMA node as sibling processes, the average CPU of the node reaches the target CPU when the number of sessions is 468,500 in the 5^{th} iteration. Pinning of one b-process and one a-process on another NUMA node processed more packets than pinning two a-processes. This shows that it is better to pair one b-process and one a-process as sibling processes so that the latency to send the packet to the a-process is relatively small and they might use the same cache memory.

	Sessions	**pps**	**Gbps**	**avg CP**	**peak CPU**	**drop ppm**	**iter.**
1A&1B	515400	-0.03%	-0.12%	49.4%	53%	+91.03%	6
2A	468500	-0.02%	baseline	48.1%	51%	baseline	5

Table 4.4: Results for scenarios 3.8 and 3.7 respectively.

4.3.2 vEPG deployment with 3CP and 2UP VMs

For the deployment we discussed in section 3.4.2, the result is shown in Table 4.5. This configuration stopped at iteration 3 when the average CPU utilization of the node reached the target CPU 50%. The average CPU utilization of this configuration reaches the target CPU early in iteration 3, since there is a small number of b-processes and a-processes as discussed in section 3.4.2. This deployment processed 0.07 percent less packets than the baseline configuration as shown in Table 4.5, when the target CPU is set to 50% .

Sessions	**pps**	**Gbps**	**avg CPU**	**peak CPU**	**drop ppm**	**iter.**
387200	-0.02%	baseline	48.7%	50%	+38506%	3

Table 4.5: Results for scenario 3.9.

The pps processing capacity and sessions of this configuration on the last iteration is shown in the bar chart Figure 4.2 with the name 3cp2up. The average CPU utilization of

this configuration reaches 40.1% early.The packet-per-second processing of the configuration increases almost linearly as the average CPU utilization increases, until it reaches the target CPU shown in Figure 4.3 with triangle shape and green color. The drop ratio also increases for every next iteration as shown in Figure 4.4 with the triangle shape and green color. Since there are only two a-processes in the user plane VMs, they are overloaded as the number of sessions increases in every iteration and the drop ratio also increases.

4.3.3 vEPG deployment with 2CP and 3UP VMs

The following Table 4.6, contains the results for the vEPG deployment with 2 control plane and 3 user plane VMs. As we have discussed in section 3.4.2, the main objective of this configuration is to increase the b-processes and a-processes by deploying 3 user plane VMs instead of 2 user plane VM in section 3.4.2. This deployment stopped early at iteration 4 and processed more packets than the deployment in section 3.4.2, but this configuration processed less packets per second than the baseline configuration by 5.59% as shown in Table 4.6. Finally, this configuration stopped in the $4th$ iteration without reaching the target CPU when the average CPU utilization reached 37.4%, which is 27.80% performance degradation from the baseline configuration. The relationship between the average CPU utilization and packets per second of this configuration is shown in Figure 4.3 with the square shape and black color.

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
425900	-5.95%	-6%	37.4%	40%	+4374440%	4

Table 4.6: Results for scenario 3.10.

On the $4th$ iteration, Dallas stopped sending traffic to the node since there are more packets dropped as shown in Figure 4.4 with square shape and black color. On this configuration, there are more packet drops as the number of sessions is increasing in every iteration, and the 2 control plane VMs are not able to route all the packets, because the a-processes are overloaded. The packets per second processed and the number of sessions of this configuration is shown in the bar chart shown in Figure 4.2 with the name 2cp3up. The pps processing capacity of the node was increasing linearly as the number of sessions increases. But, the pps processing capacity of the node fails to be linear when the number of sessions are 3.9 x 10^5 as more packets are dropped during that iteration.

4.3.4 vEPG deployment with 2CP and 2UP VMs

The result for the vEPG deployment we discussed in section 3.4.3 with both the user plane VMs on the first NUMA node and the control plane VMs on the second NUMA node is shown in Table 4.7. The main objective of this configuration is to see the impact of separating the control VMs from the user plane VMs between the NUMA nodes. The packets per second processing capacity and number of session on the last iteration

of this configuration is shown in the bar chart shown in Figure 4.2. This configuration processed 1.97 percent less packets than the baseline configuration when the number of sessions is 515,400 on the last iteration.

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
515400	-1.97%	-2.03%	34.8%	39%	+25321%	6

Table 4.7: Result for scenario 3.11.

The relationship between average CPU utilization and packet-per-second processing capacity of this configuration is shown in Figure 4.3 with the name "2cp2up". The average CPU utilization of this configuration reaches 12% in the first iteration and 34.8% in the last iteration. As shown in Figure 4.2, this configuration has almost the same CPU utilization with the 1cp2up configuration. The packet-per-million drop count of this configuration was below the threshold before the average CPU utilization reaches 31.5% as it shown in Figure 4.4, but this configuration drops almost 25321% more packets per million than the baseline configuration when the number of sessions is 515,400.

4.3.5 vEPG deployment with 2CP and 1UP VMs

Table 4.8, shows the result at the $9th$ iteration for the vEPG deployment with two control plane VMs on the first NUMA node and one user plane VM on the second NUMA node as we discussed in section 3.4.4. This deployment processed 4.93% less packets per second than the baseline configuration when the CPU utilization of this configuration reaches 26% in the last iteration. Since this deployment has more a-processes than the deployment in section 3.4.3 and has more control plane VMs than the user plane VMs, it processes more packets than the configuration in section 3.4.3.

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
686000	-4.39%	-2.29%	26%	28%	9868%	9

Table 4.8: Result for scenario 3.12.

The packet-per-second processing capacity and number of sessions of this configuration on the last iteration is shown on the bar chart in Figure 4.2 with the name "2cp1up". This configuration processed some packets when the number of sessions is 686,000 on the last iteration. The packet-per-second processing capacity of the node increases as the average CPU utilization increases as in the Figure 4.3. This configuration reaches 12% average CPU utilization when the number of sessions is 320,000 in the first iteration and 26% when the number of sessions is 686,000 in the last iteration. As shown in Figure 4.4, the dropped packets per million of this configuration was running below the threshold level until iteration 7, but it raises above the threshold level in the $8th$ iteration. The pps processing capacity of this configuration is saved as the last iteration since no more iterations are performed.

4.3.6 vEPG deployment with 1CP and 2UP VMs

The result of the vEPG deployment with one control plane on the first NUMA node and two control plane VMs on the second NUMA node is shown in Table 4.9. This configuration stopped in the $6th$ iteration at an average CPU utilization of 35% when the number of sessions is 515,400. The result of this configuration is almost the same as the result of the configuration we discussed in section 3.4.3, but this configuration uses only one user plane VM with more a-processes and b-processes instead of two UPs as in the configuration on section 3.4.3.

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
515400	-3.34%	-3.43%	35%	38%	+39158%	6

Table 4.9: Result for scenario 3.13.

The bar chart in Figure 4.2 with the name "1cp2up", shows the pps processing capacity and number of sessions of this configuration. This configuration processed 3.34% less packets than the baseline configuration when the number of sessions is 515,400 in the last iteration. The pps processing capacity of this configuration increases as the average CPU utilization increases as shown in Figure 4.3 with the name "1cp2up". The packets per million dropped in this configuration are below the threshold level until it reaches iteration 5. At iteration 6, the pps processed by the node raise above the threshold level as shown in Figure 4.4 with a name "1cp2up". The result of this configuration shows that having two CP VMs with two UP VMs like the configuration we discussed in section 3.4.3, is the same as having one CP VM for two UP VMs in this configuration. These two configurations have almost the same pps processing capacity with the same average CPU utilization as shown in Figure 4.3 with the names "2cp2up" and "1cu2up".

4.3.7 vEPG deployment with 1CP and 1UP VMs

For the configuration discussed in section 3.4.6, the result is shown in Table 4.10. This configuration processed 15.82% less packets at an average CPU utilization of 26% when the number of sessions is 686,000 in the last iteration. This configuration drops 8577% more packets per million. The packets per second processed on the node increase almost linearly until the 8^{th} iteration is reached, but falling rapidly after iteration 8 as there is high packet drop in the 9^{th} iteration.

Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	iter.
686000	-15.82%	-14.79%	26%	28%	+8577%	9

Table 4.10: Result for scenario 3.14.

The packets per second processed and the number of sessions in the last iteration of this configuration is shown in the bar chart in Figure 4.2 with the name "1cp1up". The average

CPU utilization of this configuration reaches 12% in the first iteration and 26% in the last iteration. As shown in Figure 4.3, the average CPU utilization of this configuration is almost overlapping with the 2cp1up configuration as both of them reached the same average CPU utilization in almost all of the iterations. This configuration dropped more packets than the 2cp1up configuration as shown in Figure 4.4.

4.4 Discussion

As discussed in section 2.2, the difference in average CPU utilization between the NUMA-aware and deactivated NUMA-aware configuration on the SSC3 (SSR platform) is not determined as the peak CPU of the node for the deactivated NUMA-aware configuration reached 100% early. To compare the CPU load of the baseline configurations with the deactivated NUMA-aware configuration, we made a test by lowering number of sessions for both configurations. We lowered the number of sessions to 1.5M for the first iteration, which means that Dallas starts sending 1,500,000 sessions to the node on the first iteration and multiply it by a factor of 1.02 for the next iteration until the peak CPU reaches 100%, or stop when it runs for 11 iterations without failure.

For this number of sessions, the node is stable for both the NUMA-aware and deactivated NUMA-aware configurations. Both configurations run for 11 iterations without failure as shown in Table 4.11. From Table 4.11, it is clear 3450% more packets per million are dropped on the deactivated NUMA-aware configuration than the NUMA aware-configuration. This shows that even if the b-processes and a-processes of the node are able to handle this number of sessions as NUMA-aware configurations, there may exist packet loss due to cache misses as the b-processes have no specified a-processes to distribute the packets like the NUMA-aware configuration. The CPU utilization of the node for both configuration increases almost linearly as the number of session increases in Figure 4.5.

	Sessions	**pps**	**Gbps**	**avg CPU**	**peak CPU**	**drop ppm**	**iter.**
NUMA	1865100	baseline	baseline	62.8%	65%	baseline	11
UMA	1865100	0.02%	+0.12%	87%	91%	+3450%	11

Table 4.11: NUMA-aware and UMA configurations results for less number of sessions.

In the $11th$ iteration, the average CPU utilization of the NUMA-aware configuration reaches 62.8% when the number of sessions sent from Dallas is 1,865,100. At this iteration the average CPU utilization of the node for the deactivated NUMA-aware configuration reaches 87.0% for the same number of sessions as the NUMA-aware configuration. This means that the NUMA-aware configuration processed this number of sessions at an average CPU utilization of 62.8% while the deactivated NUMA-aware configuration processed the same amount at an average CPU utilization of 87%. We observed performance degradation of almost 24.2% on the node when the NUMA-awareness is deactivated.

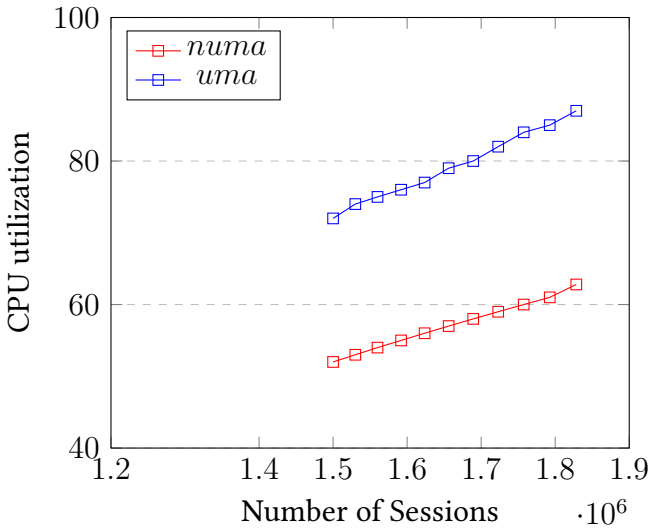


Figure 4.5: packets per second with number of sessions.

Virtualization

Since we presented the results of each configuration on virtualization in section 4.3, the results for the different configurations are discussed and compared in this section.

The result for the baseline and all other configurations are shown in Table 4.12. The baseline runs for 10 iterations and is our reference for the rest of the configurations. The result of every configuration on its breaking point is compared with the baseline on that iteration with the same number of sessions.

In the Table 4.12, the column perf(+/-) is used to indicate the performance improvement or degradation in reference with the average CPU utilization of the baseline configuration. The CPU utilization is compared with the iteration at which the configuration stopped. For example, for the configuration 3cp2up, it stopped at iteration number 3, the CPU utilization of this configuration is compared with the baseline configuration at iteration 3. The column sessions, specify the number of sessions sent on each iteration. The number of sessions sent is equal for all the configurations on each iteration. The "iter" column, indicates the iteration at which the node failed to process the required packets, meaning the iteration at which Dallas stopped sending traffic to the node because of the node failing to pass one of the condition discussed in section 3.2 or may have reached the target CPU.

As discussed in section 4.3.1, deactivating NUMA-awareness configuration on virtualization does not affect the performance of the node. Therefore the CPU utilization difference between the NUMA-aware and deactivated NUMA-aware configurations is 0.8%, which is almost the same result, except there are more packet drops in the NUMA-aware

configuration than in the deactivated NUMA-aware.

For the configuration in which we pinned one b-process and one a-process, the average CPU utilization reaches the target CPU at iteration number 6, while the average CPU utilization of the baseline configuration reaches 33.1% for the same number of sessions at the same iteration. This shows a performance degradation of 16.4% on this configuration than the baseline configuration.

The same for the configuration in which we pinned two a-processes, the average CPU utilization reaches (48.1%) on the target CPU at iteration 5, while the CPU utilization of the baseline configuration at this iteration is 30% for the same number of sessions (468,500) (Table 4.12). Therefore, there is 18.1% performance degradation on this configuration than the baseline configuration.

Config	Sessions	pps	Gbps	avg CPU	peak CPU	drop ppm	Stops @iter.	perf (+/-)
Baseline	754600	baseline	11.49	49.3%	55%	baseline	10	
UMA	754600	-0.13%	-0.17%	48.5%	54%	+24.04%	10	+ 0.8%
baseline	515400	baseline	baseline	33.1%	37%	baseline	6	
pinn.B&1A	515400	-0.03%	-0.12%	49.4%	53%	+91.03%	6	-16.4%
baseline	468500	baseline	baseline	30.0%	34%	baseline	5	
pinn.2A	468500	-0.02%	baseline	48.1%	51%	baseline	5	-18.1%
baseline	387200	baseline	baseline	25.0%	28%	baseline	3	
3cp2up	387200	-0.02%	baseline	48.7%	50%	+38506%	3	-23.7%
baseline	425900	baseline	baseline	27%	31%	baseline	4	
2cp3up	425900	-5.95%	-6%	37.4%	40%	+4374440%	4	-10.4%
baseline	515400	baseline	baseline	33.1%	37%	baseline	6	
2cp2up	515400	-1.97%	-2.03%	34.8%	39%	+25321%	6	-1.7%
baseline	686000	baseline	baseline	44.7%	50%	baseline	9	
2cp1up	686000	-4.39%	-2.29%	26%	28%	9868%	9	+18.7%
baseline	515400	baseline	baseline	33.1%	37%	baseline	6	
1cp2up	515400	-3.34%	-3.43%	35%	38%	+39158%	6	-1.9 %
baseline	686000	baseline	baseline	44.7%	50%	baseline	9	
1cp1up	686000	-15.82%	-14.7%	26%	28%	+8577%	9	+18.7%

Table 4.12: Results for the different scenarios relative to the baseline's iteration.

The average CPU utilization of the 6vCPUs default deployment configuration with 3CP and 2UP VMs reaches the target CPU at iteration number 3 when the number of sessions is 387,200 while the average CPU utilization of the baseline configuration is at 25% for the same number of sessions. On this configuration, there are only two b-processes and two a-processes which are half of the number of b-processes and a-processes in the baseline configuration. As a result, there is 23.7% performance degradation than the baseline configuration.

The same for the configuration using 2CP and 3UP VMs with 6vCPUs each, the average CPU utilization reached 37.4% at iteration 4 while the average CPU utilization of the baseline configuration at that iteration is 27%. As discussed in section 4.3.3, this configuration stopped without reaching the target CPU utilization due to more packet drops on the control plane cards as there are less control plane VMs than user plane VMs. This configuration is better than the 3CP and 2UP deployment configuration but there is still 10.4% performance degradation compared with the baseline configuration.

The average CPU utilization of the configuration with two CP VMs on the first NUMA node and two UP VMs on the second NUMA node, as discussed in section 3.4.3 stopped at 34.8% CPU utilization without reaching the target CPU. Dallas stopped sending traffic to the node before the bandwidth of the node reaches the maximum bandwidth of the vSwitch. This shows that the vSwitch can forward traffic to the node, but due to more packet drops on the node, the test is terminated without reaching the target CPU at iteration 6.

In this iteration, the CPU utilization of the baseline configuration is 33.1% as shown in Table 4.12. Since both the control plane and user plane VMs are on different NUMA nodes, the node dropped more packets as the communication between control plane and user plane VMs is by traversing the QPI. As a result, there is 1.7% performance degradation on this configuration compared to the baseline configuration.

For the configuration with 2cp1up, the average CPU utilization reached 26% when the number of sessions are 686K at iteration 9. For the same number of sessions, the CPU utilization of the baseline configuration reached 44.7% on the same iteration. As we discussed in section 3.4.4, this configuration has 9 a-processes and 2 b-processes on one UP VM. Therefore, this configuration has better CPU utilization of 18.7% over the baseline configuration as shown in Table 4.12.

For the deployment we discussed in section 3.4.5, the average CPU utilization stopped at an average CPU utilization of 35% without reaching the target CPU. This configuration gave almost the same result as the 2cp2up configuration. Both these configuration stopped at iteration 6 when the number of sessions are 515,400. In terms of resource usage, this configuration uses half of the resources of the host machine that is used for control plane VMs on the 2cp2up configuration we discussed in section 3.4.3. This configuration has 1.9% CPU performance degradation than the baseline configuration.

The last configuration that presented better results than the baseline configuration is the deployment we discussed in section 3.4.6. The average CPU utilization of this configuration stopped at 26% when the number of sessions is 686,000 as shown in Table 4.12. For the same number of session and at the same iteration, the average CPU utilization of the baseline configuration reaches 44.7%. The result of this configuration is the same as the result of the configuration we discussed in section 4.3.5.

Summary of this chapter

On this chapter, we assessed and discussed the results of different configuration on the SSR platform and virtualization. We used average CPU utilization as our comparison metrics for all the configurations. Since CPU utilization and and packets per second count of a configuration is linearly proportional, it is known a configuration with better CPU utilization will process mare packets than a configuration with worse CPU utilization for the same number of sessions. On the SSR platform, we run a test for the NUMA-aware and deactivated NUMA-aware configuration. As we have discussed in section 4.4, there is 24.4% performance degradation on the NUMA-unaware configuration than the NUMA-aware configuration.

On the virtualization, we used 50% average CPU utilization as a target CPU utilization as the bandwidth of the vSwitch is limited to some constant value. From all the configurations, Both the 1cp1up and 2cp1up configurations showed 18.7% CPU utilization performance improvement over the baseline configuration. This configuration uses one CP VM instead of two as the 2cp1up configuration. Therefore, this configuration saves the memory and hard disk of the host machine that was used by one control plane VM on the 2cp1up configuration. But, the 2cp1up configuration processed 3.1377 percent more packets per second than the 1cp1up configuration and has 72.4119 percent less packets per million drop ratio than the 1cp1up configuration. Therefore, based on average CPU utilization the 2cp1up configuration is the most efficient configuration than all the configurations we have tested so far.

5
Conclusion

This chapter is a general conclusion of the thesis work. We divided the chapter into two sections. In the first section we conclude the work of this thesis in short, and in the future work section we discuss about a future study that could follow our concept and focus on the latest AMD server processors.

5.1 Conclusion

In this project, we have shown the impact of NUMA and UMA configurations of the EPG on the SSR routers using Intel server processors. The results presented lead to the fact that deactivating NUMA-awareness shows a 24% average CPU utilization degradation than the NUMA-aware configurations. Furthermore, we have deployed the EPG virtually to investigate the impact of different deployment and pinning scenarios on the virtualized SSR using the VIRTIO virtualization technology. On the virtualization, we have shown that deactivated NUMA-aware configurations do not affect the performance of the EPG, because the memory of the VMs is defined on the configuration file during the orchestration process.

As we have discussed in section 4.4, we have taken the configuration described in section 3.4.2 as a baseline to evaluate the rest of the configurations. As shown in Table 4.12, both the configurations with 2cp1up and 1cp1up have 18.7% CPU average utilization improvement over the baseline configuration and are the most efficient configurations of the ones tested in this study. Based on packet-per-second processing capacity and packets dropped, the configuration with 2cp1up is better than the 1cpu1up configuration, and we can state in confidence that this configuration showed higher efficiency of CPU utilization on the virtualization among all configurations.

5.2 Future Work

During this project, we experimented with Intel-processor servers and designed our test scenarios based on the diversity we could achieve and the variety of performance results we could produce. In our initial plan we also wanted to work on the newly acquired AMD EPYC series processors and the servers based on them. Since AMD EPYC has 8 NUMA nodes, it is possible to test more configuration than we did on the virtual environment. By the time of finishing this study we did not have the opportunity to achieve that goal, because of the availability of those systems and the time-plan of this work. This goal

could be achieved in a future study, because the AMD's unique architecture is of great interest regarding NUMA configurations as shown in the next section.

5.2.1 AMD EPYC

All AMD EPYC processors are composed of four eight-core Zeppelin dies. Thus, a dual socket EPYC setup is in fact a virtual octa-socket NUMA system. This topology is ideal in applications with many independently working threads such as small VMs, which is the case for the EPC. This processor is a System-on-Chip processor with up to 32 Zen cores per SoC [19] which allows "hyperthreading" concept so that each module, can provide up to 64 threads.

This processor has two sockets that are connected with a PCI-Express (PCIe). EPYC processors support both single and dual-socket motherboards, and each EPYC processor provides 8 memory channels and 128 PCIe 3.0 lanes for each socket. Each die in an EPYC module, contains four memory channels, 8 physical cores, one NUMA node and 4 PCIes as shown in Figure 5.1.

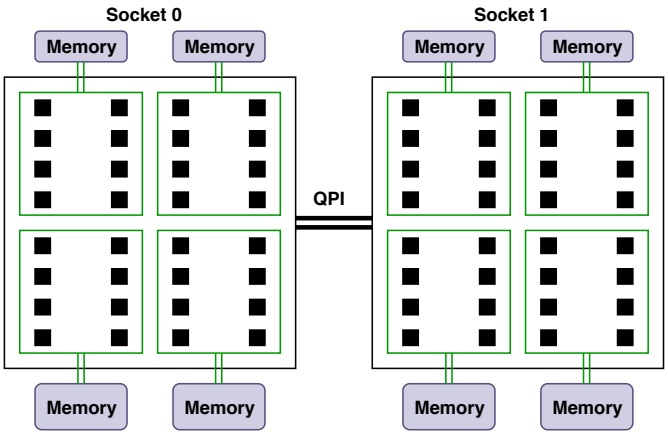


Figure 5.1: AMD EPYC architecture.

Looking at the architecture presented on the figure above, it is clear that there is a significant number of possible combinations and configurations that can be the subject of a future study. The freedom given by the virtual deployment displays potential on customized setups, and it would be really interesting to investigate the performance of these processors and the behaviour of their "internal NUMA nodes".

Bibliography

[1] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A high performance packet core for next generation cellular networks", in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, USA, 2017, pp. 348–361.

[2] P. Lescuyer and T. Lucidarme, *Evolved packet system (EPS): the LTE and SAE evolution of 3G UMTS*. John Wiley & Sons, 2008.

[3] M. Olsson, S. Rommer, C. Mulligan, S. Sultana, and L. Frid, *SAE and the Evolved Packet Core: Driving the mobile broadband revolution*. Academic Press, 2009.

[4] P. Satapathy, J. Dave, P. Naik, and M. Vutukuru, "Performance comparison of state synchronization techniques in a distributed lte epc", in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2017, pp. 1–7.

[5] Y. Liu, Y. Zhu, X. Li, Z. Ni, T. Liu, Y. Chen, and J. Wu, "Simnuma: Simulating numa-architecture multiprocessor systems efficiently", in *2013 International Conference on Parallel and Distributed Systems*, IEEE, 2013, pp. 341–348.

[6] C. Lameter *et al.*, "Numa (non-uniform memory access): An overview.", vol. 11, no. 7, p. 40, 2013.

[7] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter, "Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches", in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, IEEE, 2009, pp. 250–261.

[8] S. Cho and L. Jin, "Managing distributed, shared l2 caches through os-level page allocation", in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, IEEE, 2006, pp. 455–468.

[9] H. Dybdahl and P. Stenstrom, "An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors", in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, IEEE, 2007, pp. 2–12.

[10] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Managing data placement in memory systems with multiple memory controllers", *International Journal of Parallel Programming*, vol. 40, no. 1, pp. 57–83, 2012.

[11] Z. Majo and T. R. Gross, "Memory management in numa multicore systems: Trapped between cache contention and interconnect overhead", *SIGPLAN Not.*, vol. 46, no. 11, pp. 11–20, Jun. 2011, ISSN: 0362-1340. DOI: 10.1145/2076022.1993481. [Online]. Available: http://doi.acm.org/10.1145/2076022.1993481.

[12] K. Harzallah and K. C. Sevcik, "Evaluating memory system performance of a large-scale numa multiprocessor", in *MASCOTS*, 1994.

[13] D. Molka, D. Hackenberg, and R. Schöne, "Main memory and cache performance of intel sandy bridge and amd bulldozer", Jun. 2014. DOI: 10.1145/2618128.2618129.

[14] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore smp systems", in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2009, pp. 413–422.

[15] S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali, "A case for numa-aware contention management on multicore systems", in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ACM, 2010, pp. 557–558.

[16] M. S. Reorda, Z. Peng, and M. Violante, *System-level test and validation of hardware/software systems*. Springer Science & Business Media, 2006, vol. 17.

[17] J. Doweck, W.-F. Kao, A. K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake", *IEEE Micro*, vol. 37, no. 2, pp. 52–62, 2017.

[18] K. Jackson, C. Bunch, and E. Sigler, *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2015.

[19] X. Guo, *Best Practice Guide – AMD EPYC*, http://www.prace-ri.eu/best-practice-guide-amd-epyc/, 2018.

List of Figures

List of Tables

List of Abbreviations

AAA Authentication and Access Authorization
CEE Cloud Execution Environment
CP Control Plane
DDR Double Data Rate
DINO Distributed Intensity NUMA Online
eNodeB Evolved NodeB
EPC Evolved Packet Core
EPG Evolved Packet Gateway
EPS Evolved Packet System
HOT Heat Orchestration Template
HSS Home Subscriber Serve
IntelSGX Intel Software Guard Extensions
IP Internet Protocol
LC Line Card
LLC Last Level Cache
LTE Long-Term Evolution
MME) Mobility Management Entity
NIC Network Interface Card
NUMA Non-Uniform Memory Access
PCC Policy and Charging Control
PCIe Peripheral Component Interconnect-Express
PCRF Policy and Charging Rules Function
PG Packet Gateway
PGW PDN Gateway
PISC Packet Inspection and Service Classification
PP Payload Processing
QoS Quality of Service

QPI QuickPath Interconnect
RAN Radio Access Network
RP Route Processing
SGW Service Gateway
SoC System-on-Chip
SSC Smart Service Card
SSR Smart Service Router
UE User Equipment
UMTS Universal Mobile Telecommunications System
UP User Plane
UPI Ultra Path Interconnect
vCPUs virtual CPUs
VDP Virtual Deployment Packag
vEPG virtual EPG
VIRTIO Virtual I/O
VM Virtual Machine