Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 7, No. 4, November 2016), Pages: 95-107
www.jacr.iausari.ac.ir

Packet Reordering Algorithms in Multi-core Network Processors

Abolfazl Akbari✉, Mohammad Karim Sohrabi, Pourya Khodabandeh
Department of Computer Engineering, Semnan Branch, Islamic Azad University, Semnan, Iran
akbari1761@yahoo.com; amir_sohraby@yahoo.com; pouryakhodabandeh@gmail.com

Received: 2016/08/28; Accepted: 2016/10/16

Abstract

Usually in the form of parallel processing with parallel multi-processor systems are designed .For this reason irregular flow may occur with packages. When you're tired and it all comes out in his NP (Network Processors) system, Reset depending on network performance is negatively affected and we delayed To do this, The design of processors working in parallel packet transmission by avoiding any packet reordering occur, If reordering, this time delay is minimal. Several algorithms have been proposed in recent years. In this paper, we will describe some of the fundamental concepts involved in NP architecture and algorithms packet reordering in recent years, which causes delays are minimized packet reordering, the evaluation will be compared.

Keywords: *Network Processors, Packet Reordering, Parallel Architectures*

1. Introduction

In new networks line speeds are rising and with it too the processing power needed for packet processing. Between all feasible methods Network Processors oblige the best tradeoff among implementation and flexibility. To handle the processing load all mercantile Network Processors pursue a multi-core solution [1] for parallel packet processing.

These NPs are generically single-chip multiprocessors with high-implementation I/O components. They include several ordinary processor cores which are optimized for exerting packets along with a control processor, which handles higher level functions. A network processor is generally set on a physical port of a router. Packet processing tasks are maked on the network processor before the packets are transited on through the router switching piece and through the next network link.

Notice in Figure 1. Packets P1, P2, P3, P4 of the same flow attain at the receive buffer (RBUF) of the network processor in order. Packets are assigned to threads in various micro engines in the following way: P1, P2, P3, P4 are assigned to ME1-T1 (Micro engine1-Thread1), ME2-T1, ME3-T1 and ME4-T1respectively. Now packet P1, being processed by ME1-T1, can get delayed with respect to P2, P3, P4. This can occur owing to different reasons, e.g., processing of other threads, or pending memory requests in DRAM FIFO. So the thread ME1-T1 completes the processing of P1 only after ME2-T1, ME3-T1, and ME4-T1 have processed their respective packets. So packet P1 is delayed with respect to P2, P3, and P4 and is transmitted only after P2, P3, and P4 have

been forwarded. This may result in a retransmission of P1. In figure 1 show how the synchronous processing of packets can efficacy the ordering of packets [2].

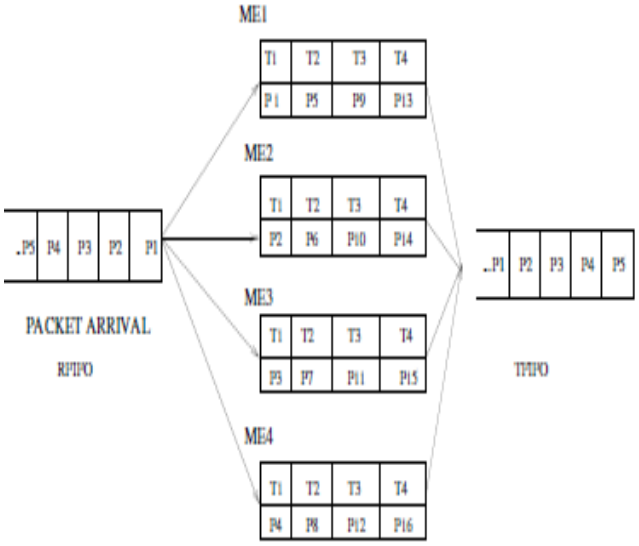

Figure 1. Packet Reordering in Network Processors.

Reset packets can adversely affect the overall network performance is, Consider ip (IP source and destination address, source and destination port number and transport layer protocol number) packet-based protocol is TPC that Packages are sensitive to reset. If you have irregular packages this leads to the remobilization. Efforts to reduce packaging as much as possible reset mode. If packets are out-of-order, this may lead to retransmission since late packets may be seen as being lost. Evaluations on an IXP2400 [3] have shown, that the retransmission rate in a 10-hop network due to packet reordering is 10% and more if there are no order ensuring mechanisms. This can result in a significant network performance reduction of up to 60% in packet throughput [4].

Read this article is as follows :In section 2 to interpretation explains the different parts of NP .In Section 3 we referred to last several procedure In Section 4 we do a comparison between them and to offer Section 5 concludes our discussion.

2. NP Architecture Fundamentals

The following section will describe some of the fundamental concepts involved in NP architecture.

2.1. Data Path Versus Control Path

Packets are operated upon at wire speed, meaning that if 25 M packets enter every second, 25Mpackets must leave during the same interval. A network processor configuration is typically portrayed as a duality of two entities: the core network processor takes care of the simple tasks; the host processor handles exceptional packets and management routines. Since the vast majority of packets require very little processing, the general-purpose nature of the processing power devoted to deal with exceptional cases is better fitted outside of the actual NP. Hence the division into a fast

path and a slow path. Most packets follow the fast path through the NP, a “dumb” but fast path. Some packets, however, are classified by the NP as being too difficult or obscure to handle properly and they are thus sent over to the host.

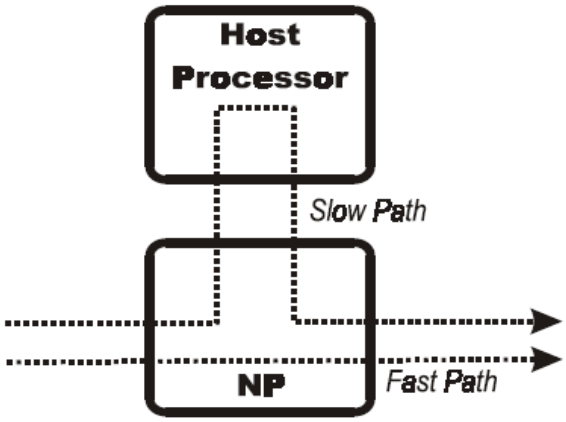

Figure 2. The fast and the slow path of a router with a host and an NP.

2.1.1. The Host Processor

The host processor, or host computer, is a general purpose processor (GPP) and handles the processing of packets too demanding for the fast path. It is also responsible for providing the fast path NP with proper information for its functioning. For example, the next-hop address is fetched for every packet handled, and that usually takes place in a look-up engine of some form. From the NP’s point of view, the look-up table contains data to be read and used; but the NP never performs any write operations to the look-up table. It does not have that option, it is the prerogative of the host. The look-up table data is based on control packets sent between the routers. The host performs a routing algorithm based on the figures it receives measuring the load/maximum data rate/availability of the lines connected. It can be described as if the host is responsible for gaining knowledge about the environment, the world it is placed in. It then presents an image of the network to the NP in form of the routing table. Compared to the workload of the NP, the host has to deal with more complex operations that occur far less frequently. The processing of packets occur every couple of nanoseconds; routing tables are updated at far more distant intervals, on the order of minutes. A GPP may be located on the same chip as the fast path, or it may have a chip of its own. Many commercial NPs have an on-chip GPP: C5 has its XP, and the IXP1200 has a Intel Strong Arm core to handle the general tasks. The configuration of placing a e.g. PowerPC in conjunction with the device is not uncommon. That way, management is facilitated, as a more user-friendly interface to the NP may be provided, along with suitable developmental tools.

2.1.2. The Host-Np Interface

As a distinction is made between a fast and a slow packet path; where exactly to draw the line is the fundamental issue in NP design. In the broadest sense, network processors are by no means a new phenomenon. There was always a computer in charge of routing tables and exceptional cases, and the packet forwarding was done by an application-specific integrated circuit (ASIC). ASICs have the advantage that they can be made almost arbitrarily fast, since they are designed for one purpose, and thus do not have to

pay, in terms of performance, for unnecessary flexibility. The fast path goes through the ASIC, and the slow path through the host. Network processors, however, try to introduce programmability in the fast path without compromising speed too much. That can be done by identifying the basic protocol processing operations that are performed on every packet, implement hardware support for their execution, and thus focus on making the fast path – fast. The more intelligence incorporated into the fast path, the greater the risk that the design will not be able to handle wire speed. The tradeoff is speed for flexibility. If the NP is fast but dumb, it is the nature of the data traffic that determines the success of the design. Keep in mind that it is in part networking requirements that has created the need for NPs. If many packets require plenty of processing, the dumb NP will have to send these packets on to the GPP. The bottleneck will then appear in either the GPP performance or the NP-GPP interface bandwidth. The more flexibility, programmability, can be placed in the fast path NP, the lower the chance of having to send packets through the slow path.

2.2. The Processing Element (PE)

Virtually all network processors are multiprocessors, meaning that they are not built as one huge RISC processor, but rather several small ones working in parallel, co-operating, on the same silicon chip. The basic programmable unit in the network processor is henceforth referred to as the Processing Element, PE. The PE may be clustered in groups of two or more, forming more complex "Processing Elements" at a higher level of system design. They may also work in conjunction with other small programmable units optimized for some task, but all these are in some sence PEs; the defining characteristic of a PE in this context is a programmable unit not containing other units that are programmable by the user. From the architectural perspective, design issues need to be confronted at two or more levels. First there is the design of the PE itself. It needs a basic instruction set, memory and a data path suitable for operating in a multi-PE environment. Second, there is the question of how to arrange the PEs. Are all to be of the same type – general purpose? Or should some of them feature specialized instructions for data manipulation, while others focus on rapid load-store

Operations? How should the data path from the ingress to the egress be conducted through the PEs? Today, quantitative data of the benefits of PE architecture and configurations is poor.

There are some studies done, e.g. the project at the University of Washington [5], but the wide spectrum of approaches seen in commercial NPs suggests that there is plenty of room for further research.

2.2.1. PE Architecture

The architecture of the PE depends very much of the environment it is intended to operate in. Terms like NISC (network instruction set computer), PISC (packet instruction set computer), DCP (digital channel processor), ACP (Active Communications Processor) etc. are used to describe PEs in different architectures. NPs differ in terms of number of PEs and consequently also in terms of PE complexity. Intel IXP1200 has six PEs, all capable of sustaining two simultaneous threads. Motorola C-5 has 16, but in the design available as intellectual property from Clear Speed Inc., hundreds of simple 8-bit PEs may be included.

2.2.2. PE Configurations

The PEs may be grouped into functional blocks, or be independent. It is logical that the more functionality incorporated into one PE, the more able it is to work on its own. However, given that the overall incentive of the NP architect is to gain high performance by maximizing utilization of the on-chip logic, whether large and potent multi-threaded PEs or small and simple PEs should be used is not a trivial issue. The best configuration is one where idle processing time is kept low, and the number of packets that can process in parallel is maximized.

2.2.3. Co-processors and Hardware Accelerators

A hardware accelerator is any state machine that operates independently of the network processor PEs and that can be called upon as a functional unit. If the hardware accelerator is in itself programmable, it is called a co-processor. Hardware accelerators are used to speed up certain simple tasks that are done at a regular basis. Examples of accelerators can be CRC calculation, routing table look-up engines etc.

2.3. Exploiting Parallelism

Parallelism is the term used to describe how tasks may take place independent of each other. Consider for example an algorithm that reads a series of numbers and squares each of them. That task contains parallelism in the sense that the result is insensitive to the order the numbers are read; if there are enough arithmetic resources in our computing hardware, several numbers may be squared simultaneously. If the algorithm is altered to not just square the numbers, but to add the square of the first number in the series to the next number before squaring, the parallelism is from the algorithm. Each operation need to be finished before the next one can be begun. And this is the case regardless of the number of squaring elements

Available in the processor. Instruction-level parallelism, ILP, is a well-known phenomenon that modern processors are designed to exploit. Thread-level parallelism, TLP, is a relatively new concepts that is beginning to emerge in commercial processors; it exploits the independence existing between different programs, threads of execution that execute on the same processor. Packet-level parallelism, PLP, finally, is an even higher level of parallelism existing in the data flow in NPs. When computer architecture takes advantage of inherent parallelism, wherever found, performance increases are possible. Parallelism at all these three levels are necessary to examine to fully maximize performance.

2.3.1. Instruction-Level Parallelism

ILP exists whenever the machine instructions that make up a program are insensitive to the order in which they are executed. If dependencies exist, then one instruction has to wait for another to be completed. If not, they may be executed simultaneously. This is what happens in a multi-scalar processor. Dependencies are automatically detected, and ILP is exploited whenever it is encountered in the stream of instructions.

2.3.2. Thread-Level Parallelism

TLP exists between different threads of execution, i.e. different programs. There are several ways to make a processor execute more than one program at the same time. The basic method of time sharing splits a time interval into pieces during which the processor executes instructions from each of the programs. For example, during 10 ms

program A gets all the attention; for the next 10 ms program B takes over etc. Time sharing is an example of underexploited parallelism. Due to dependencies in the program code currently in execution, the processor cannot utilize all functional units at all Moments. Still, there is no way to put those functional units to work with code from another Program thread that by definition has no dependencies with the original program. That is why the existing parallelism remains unexploited.

Multi-threading (MT) is a way of dealing with this problem. It lets a multi-scalar processor execute several threads at every instant. There are different forms of multi-threading. In fine grained MT, a thread is alone with the processor until it has to wait for some function with high latency to respond, such as a memory access. When such a stall occurs, the processor switches contexts and keeps on working with code from another thread. Simultaneous MT issues multiple instructions at every clock cycle; instructions from several threads.

To summarize, to gain high performance, it is essential to avoid leaving any part of the processor idle during execution. Thus there are methods such as pipelining and multithreading. To exploit parallelism means identifying theoretical independencies in program flow and implementing hardware support for the independent pieces of code, operations or part of operations to utilize the functional units of the processor.

2.3.3. Packet-Level Parallelism

Packet-level parallelism, PLP, is a description of the NP workload. It is not impossible to have a program in the NP operating on data from multiple packets (thereby creating dependencies as the order of packet processing is not irrelevant), but is highly uncommon. In the general case, each packet is treated independently. The case of PLP is different from ILP and TLP in so far that it is based on the data rather than the instruction flow. Since no NP can be made fast enough to finish working on one packet before the next one comes, they exploit PLP by working on many packets simultaneously. Consider a processor with a fixed number of functional units FU, some of which have very high latency. This can be a simplified model of an NP. A packet enters and starts being processed upon by one thread of execution. The utilization of the FUs is limited by instruction dependencies within the thread, and by the latency involved in the high-delay FUs. Another packets enters before the first one is done with. Processing the second packet can be done completely without involving the first one. FUs that are idle due to dependencies in packet number one may be put to work through packet number two. It is possible to regard PLP as a variant of TLP in which each packet is given a thread of its own. More than one thread may operate on every single packet, e.g. if the packet processing is done at several levels in the protocol stack, and parallelism exists between packets as well, as they are, to repeat, treated independently. The question that seeks an answer is this: Are there occasions when FUs might be idle even when MT is applied, and can PLP be exploited to solve that problem? The question is still open, but it is already clear that MT solves the problem of idleness due to instruction-level dependencies. However, the problem of latency is not that easily tamed. A memory access may take more time than can be purposefully used by other threads. All in all, processing many packets in parallel is simply a way of increasing the number of threads that may execute in parallel, be it on a single processor or on parallel PEs.

2.4. Parallel or Pipelined Processing

Commercial NPs all contain a number of PEs with a certain extent of multithreading; two fundamentally different approaches exist when it comes to the arrangement of the PEs. They can be ordered either to work in parallel or in a large pipeline.

2.4.1. Parallel Packet Processing

Parallel processing in this context refers to the configuration where the entire processing of a packet is done by the same PE or group of PEs. Several PEs work in parallel on their particular packets, but there is no functional division being made between the PEs. Basic requirements for parallel processing include having a number of PEs high enough to allow for multiple packet processing. Second, the PEs themselves need to be sufficiently "general purpose" for packet processing to be able to perform all steps without unnecessary delay.

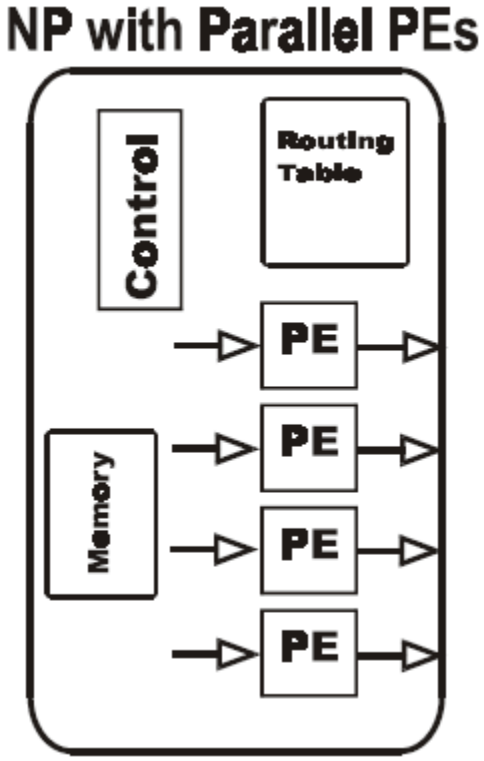

Figure 3. NP architecture based on parallel Processing Elements

2.4.2. Pipelined Packet Processing

Pipelined processing can be done when a number of steps can be identified that every packet goes through. The PEs may then be configured to do only one part each of the total work. The pipeline will be made up of independent PEs, and thus be far more complex than a RISC pipeline of functional units. When one pipeline stage is finished, it delivers the packet to the next pipeline stage and so on.

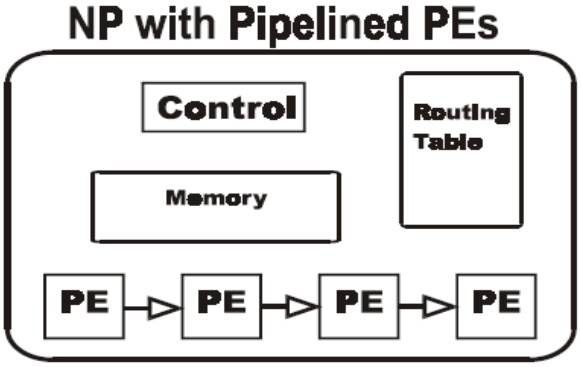

Figure 4. NP architecture based on pipelined Processing Elements

One advantage of the pipeline approach is the ability to specifically tailor the PEs for their respective tasks. Another is the high predictability in the packet stream, as packets will be processed in the same order as they enter. The ideal pipeline consists of stages that require the same time to perform, as is the case in the basic RISC. Packet processing is more complex, though, and it is difficult to calculate the latency of each stage. Conversely, the stages must be designed to make the pipeline balanced, so that the different stages take approximately the same time to complete.

2.5. On-Chip Communication

All existing NPs are multiprocessors in one way or another, having multiple PEs on a single chip. All these PEs have to operate on data that is delivered through the input data stream, and since they have to operate in parallel, they must avoid blocking each other when accessing e.g. memory buffers. Further, memory may be distributed to the different PEs, which might cause one PE to want to access the memory that belongs to another PE. If all PEs get their data off one single common data bus, that bus would easily become saturated and hence blocking. Therefore, the issue of how to arrange for on-chip communication is important. Communication architectures are characterized by their performance as well as their scalability. A crossbar provides direct access between any two nodes, but scales by the square of the number of nodes; it also occupies plenty of space, expensive silicon real estate. The octagon configuration of eight PEs in a proposed architecture by STMicroelectronics and the University of California in San Diego [6] is reportedly of much higher performance than any configuration with a common bus. The eight nodes are configured in a circle, where ever node can communicate with its two neighbors, as well as the node on the opposite side of the circle. This gives a maximum of two hops for any communication between nodes. The Octagon is also shown to be scalable to higher numbers of nodes with linear complexity in terms of connections and hops.

2.6. Memory – Buffers, Queues and Tables

When designing an NP, the question of what memory to use for what purposes is central even at the beginning. There are essentially three different types of memory to be used, if functionality is considered. These are instruction memory, packet memory and route table memory. Instruction memory can generally be kept small, as NPs so far do not have to execute very complex programs. Both PEs and control processors generally have their own instruction caches, and the entire code may well be fitted in the

same memory as the data packets, or in a special one. This is not a big issue, since it has little effect on performance in an NP. Packet memory on the other hand, the buffers where packets are written upon arrival, and read when they are forwarded; queues where the modified packets are written and eventually read when the actual forwarding takes place; that memory has to be very carefully considered. Tables are also vital for performance; very little data has to be read, but look-ups must be performed fast, with minimum delay.

2.6.1. Packet Buffers and Queues

It is trivially concluded that the on-chip memory bandwidth for the packet memory must be at least twice the line bandwidth: if 10 Gbit/s of packet data is written to the buffers, 10 Gbit/s is read from the same buffers, hence a total bandwidth of 20 Gbit/s. If the output queues, though logically distinct, are physically situated in the same memory, then the bandwidth would have to be four times that of the carrier line. Therefore, it seems wise to separate the output queues from the packet buffers and arrange them in a pipelined way so that they can be accessed in parallel. But the question remains: how can the high memory bandwidth be achieved?

First, it has to be decided whether to place the buffers on the chip or off the chip. On-chip memory can be made faster, and the bandwidth can be made almost arbitrarily high, since bus widths scale easier on-chip than off-chip. On the down side is that on chip memory can never be very big compared to off-chip. Second, what type of memory must be used? Fast and expensive SRAM or slow and cheap DRAM? A look at commercial solutions reveal that DRAM is what is almost always used for buffers. Still, a lot of work is going on to produce special fast DRAM for network equipment that might be closer to SRAM when it comes to performance.

One interesting solution for packet buffers is to have it fully distributed on the chip, along with the PEs that perform the processing. A central administrator must supply the different “cells” of PEs and memory with their packet data, but the distributed solution, along with a non-blocking communications model could provide a way to circumvent the memory bandwidth problem.

2.6.2. Look-Up Tables

The route table is a memory which is never written by the NP, only by the control processor or the host. The NP only has to read from it, and every read must be as fast as possible. Updates to the route table can take place at very distant intervals, relative to the normal time frame of the NP fast path. A query to the route table occurs with every packet that must be processed: a next-hop address must be established based on the destination address in the header. More complex classification for QoS requires look-ups to be performed to find out what action, what processing, must be performed on the packet, and not just a next-hop address. The look-ups must complete as fast as possible. How can that be solved? Basically there are two different approaches. Either a fast common memory such as SRAM is used, where the table entries are stored in an intelligent data structure that enables fast queries. This is usually some form of search tree, and the look-up table often comes with some extra hardware accelerator that performs the necessary algorithm to search into the tree. The other solution that is not uncommon is to use a memory type which functions similar to a hash table already at the gate level; a Content Addressable Memory, CAM. Compared to a RAM, data entries in a CAM can be accessed by searching on content in addition to address. Every

memory cell in a CAM stores therefore more information than RAM memory cell, in order to provide this additional capability. A query can be made not only with an address, but with some data, and the cell that holds that data will be read. The problem with CAMs is that they are more complicated to manufacture, requiring more transistors per cell, and therefore more expensive. RAM solutions can build on more common parts and take advantage of the existing knowledge on search algorithms to make a fast look-up table. However, it should be noted that very much due to the developments in networking equipment, the market for CAMs have increased considerably. It is estimated that a third of the semiconductor market for switches and routers is made up of CAMs [7].

CAMs can be binary or ternary. A Ternary CAM, TCAM, can store three binary values for every bit: zero, one and don't care. This extra feature enables more advanced algorithms to exploit the memory. A TCAM performance of approximately 100 million TCAM searches per second is common; this would enable OC-768 traffic at 40 Gbit/s with minimum size packets.

2.7. Considering Caching

A cache memory is used for data that is accessed often. If a small part of the entire memory stands for a high portion of the memory accesses during a certain time interval, it is wise to store that part in a smaller, faster memory. If the main memory is placed off-chip, the cache is often placed on the same chip as the processor, to minimize latencies.

In a general purpose CPU, data caches and instruction caches are most common. Each of these store instructions that are likely to be executed next, or data that is likely to be called for. The term locality is used to describe how useful it is to cache specific data or instructions. If some instructions exhibit temporal locality it means that once they are accessed, they are likely to be accessed again soon. Spatial locality of data means that if the data is accessed, some other data stored nearby will also be accessed soon. That way, it is wisest to place that data too into the cache. In short, a cache memory contains a copy of some of the data in the main memory, namely the data that is used often.

Cache memories are by nature small compared to main memory. A main memory of 256 Mbytes generally comes with two or three levels of caches where the cache closest to the processor is only a couple of Kbytes big. In a general-purpose machine, the main memory contains very large programs and vast amounts of raw data. If the programs to be executed are small, however, the entire program could be stored in a cache of moderate size. It is logical that cache design for a computer depends very much on the typical workload and the programs that are meant to run on the computer.

In an NP, both programs and workload is different from a general purpose machine. The programs that run on the fast path are typically small, otherwise they could not run on the fast path. The data they operate on is of two types: data streams and state information. The data stream is made up of packets, datagrams that enter and leave. The state information is the static data that the NP uses to operate on the packets; the best example is the route table, but all data that is used for classification purposes or traffic management belongs to this group. It is possible and maybe even useful to equip an NP with one or several instruction caches (for the different PEs). Caching packet (header or payload) data, however, is of little use. At least not in the present network situation, where payloads are left untouched by the network. Caching state information is a different thing though. The route table is a large data structure stored in a memory, but

some locality may be assumed. Route table caches are therefore a most effective way of enhancing NP performance, and there is research going on in this field. A route table cache exploits the locality between destination addresses in a packet stream. If a packet heading for a certain destination arrives, it is very likely to be followed by another packet heading the same way. Consider TCP connections, such as those established every time a file is being downloaded from a server. The file is transmitted in several IP packets in the TCP connection between server and destination, and though these do not necessarily travel the same way, they are nonetheless very likely to travel parts of the way through the same routers. That way, if the corresponding entry of a destination is cached when it is accessed, the next time it will not take as long to access it. Exactly how a router cache should be designed is another question, and there are several approaches: full destination address caching, destination address range caching and routing prefix caching to mention three, with variants existing on how to effectively store the tables using intelligent hashing functions.

2.7.1. Full Destination Address Caching

Full destination address caching, or host address caching, is the straightforward method of simply caching the entire destination host address in each entry in the table. It can be demonstrated [8] that for a cache of this type, the ideal block size, i.e. the size of the part of the memory that is replaced each time a miss occurs, is one single entry. This seems logical considering that neighboring addresses in a table have little in common, and that all entries in the cache should be occupied by data that have been written there by their "own merit".

2.7.2. Destination Address Range Caching

With destination address range caching, the cached table entries do not contain individual IP addresses. Instead, each entry corresponds to a certain part of the address space, and every packet heading for a destination that falls within the range of an entry will be forwarded to the corresponding output port.

2.7.3. Routing Prefix Caching

Routing prefix caching is a model where addresses are not stored at all in a table, but rather prefixes of addresses. The prefix is the part of the IP address that, in Classless Inter-Domain Routing, identifies the network as opposed to the host in that network. The term 'Classless' means that IP addresses are not grouped in classes A, B, C and D, where the length of the network identifying prefix is fixed; instead, the number of bits in the prefix can vary, and this complicates the procedure when making a look-up. A study performed at Stanford University [9] indicates that caching prefixes rather than actual address gives a router cache with higher performance. Also, by caching prefixes instead of full addresses, cache sizes can be kept smaller when the entire IP address space is exploited in the Internet.

3. Characterized Algorithms

3.1. Single Sequence Number Algorithm

Each entering packet is austerity signed with an incremented sequence number. Then, the reordering unit at the output link demise only packets with the oldest sequence number. Other packets keep waiting. Presume that a packet A arrives before a packet B,

and both are sent to different PEs (Processor Elements). Further assume that A and B belong to different flows, and that B has light processing requirements and is ready to leave quickly, while A has heavy processing requirements.[9] For instance, B only needs forwarding, while A also needs DPI. Today, B typically needs to wait for A, potentially incurring a high delay. This simple architecture is easy to implement, it can cause high reordering delays, because packets of one flow may need to wait for a long time for late packets of a different flow.

3.2. Hashed Sequence Number algorithm

In this Methods [2], describe a Hardware Re-Sequencer Unit for Network Processors. Inbound packets will be labeled in the entrance path, guarding the packet order with flow granularity. An Aggregation Unit reorders the packet flows in the egress path if needed. In contrast to most other solutions the way of the packet through the NP system is dispensable, which enlarges design freedom in terms of e.g. load balancing.

3.3. Reordering Per Processing Phase Algorithm(RP^3)

In this way[10], the new scalable scheduling algorithm to maintain prioritized depending on the network have been introduced multi-core processors And shown that the algorithm can reduce latency reordering As with any load balancing algorithm is implemented The lowest implementation complexity is overhead. The algorithm uses a flow of packets arrived from the observation that all requests are processed the same Can be divided into a fixed number of stages of the logical processing phases and described. The third method is considered conceptual framework that when a network processor learns that the logical phases a suitable algorithm can consider using any of the frameworks.

4. Comparison Algorithms

In Rp^3 method, the aim is to provide scalable algorithm is to reduce the delay reordering as with any load balancing scheme is implemented. Therefore NP plan can use load balancing scheme to achieve a high output and only by using algorithms to reduce delays we will reordering. in single method ,The package arrived too late to Np And is lighter processing ,Must be a long time waiting for a package that earlier entered And processing is heavier To maintain sequence of packets that create a long delay.

In hashing method, Consider the balance of proven design while in RP^3, Load balancing scheme with an update.in hashing algorithm and single, NP Can be a sequence number to each packet without changing the overall load balancing, in hashing, Sort divided packages in multiple domains. Generating a different sequence number to consider for any number of Sort. In any domain is preserved sort or Sequence of packets. Therefore, unnecessary delay happens between currents with different requests processed (That is tangled in the same domain of the sort). In return in RP^3 method, According to the reordering in Processing phases. In this case, Request processing of all packets that are identical can be divided into equal number of processing logic courses that Processing phase is called. And the superiority of RP^3 method over hashing and single .in Rp^3 algorithm We have any need for buffer size.in this method Maintaining the entry sort packages and buffered in NP .When these algorithms with low latency is more than reordering Faster release package And requires less buffer compared to other methods In addition, the algorithm works on

descriptor. Thus, for correction if a package is removed during the process, Algorithm logically descriptor Package for sorting, in Rp^3 method Processing Cannot in logical phases Classification for many specially flows. Particular, the packet flows can be for other domains sorting by hashing header packet that the same algorithm is hashing. In which case the two methods RP^3 and hashing Similar to each other.

In Rp^3 method, reduces homogeneous traffic as more traffic with the same requirements and Number of phases is equal for all streams of packets. This limitation algorithm is RP^3 that because we can use a combination of algorithm RP^3 and hashing that it can be an effective new method and future.

5. Conclusion

In this paper, we study and survey and comparison algorithms packet reordering in network processors and understand the algorithm RP^3 for the third framework requires few assumptions, it also provides a slightly worse performance and is more complex to implement. Two algorithm Hashed and RP^3 can also be combined, by concurrently distinguishing flows based on an arbitrary hash as well as on the number of processing phases.

References

[1] Shah, N.: Understanding Network Processors. Berkley Technical Report (September 2001)

[2] M. Meitinger, R. Ohlendorf, T. Wild, and A. Herkersdorf, “A hardware packet re-sequencer unit for network processors,” in Proc. 21st Int. Conf. Archit. Comput. Syst., 2008, pp. 85–97.

[3] Govind, S., Govindarajan, R., Kuri, and J.: Packet Reordering in Network Processors. In: IPDPS 2007 (May 2007)

[4] Laor, M., Gendel, L.: The Effect of Packet Reordering in a backbone Link on Application Throughput. IEEE Network (September/October 2002)

[5] P. Crowley ET. al.: Characterizing Processor Architectures for Programmable Network Interfaces, Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, N.M., May, 2000, University of Washington,

[6] Karim, F.; Nguyen, A.; Dey, S.; Rao, R.,On-chip communication architecture for OC-768 network processors Design Automation Conference, 2001. Proceedings, 2001 Pages: 678–683.

[7]. Elektroniktidningen, no 11, 2002, p. 12

[8]. Wu, B., et al.: A Practical Packet Reordering Mechanism with Flow Granularity for Parallelism Exploiting in Network Processors. In: IPDPS 2005(2005)

[9]. P. Crowley ET. al.: Characterizing Processor Architectures for Programmable Network Interfaces, Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, N.M., May, 2000, University of Washington,

[10]. A. Shpiner, I. Keslassy, R. CohenScaling Multi-Core Network Processors without the Reordering BottleneckIEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 27, NO. 3, MARCH 2016.