A Method for the Stateful Data Plane Algorithm State Synchronization in the Network Processing Unit

Yaroslav Kuzmin
Lomonosov Moscow State University
Moscow, Russia
yaroslav_konst@lvk.cs.msu.ru

Dmitry Volkanov
Lomonosov Moscow State University
Moscow, Russia
volkanov@asvk.cs.msu.ru

Yulia Skobtsova
Lomonosov Moscow State University
Moscow, Russia
xenerizes@lvk.cs.msu.ru

***Abstract*—This work presents a network processing unit based on specialized computational cores that is used for packet processing in network devices (e.g. in network switches).**

Nowadays stateful data-plane algorithms are developing in software-defined networks. The idea of stateful data-plane algorithms is to move a part of control information from control plane to data plane. But these algorithms require hardware support because they need resources for state handling.

This work presents the network processing unit architecture modifications that allow to use stateful data-plane algorithms that require state synchronization between the NPU processing pipelines.

***Index Terms*—software-defined networks, network processing unit, stateful packet processing, network protocols.**

I. Introduction

Nowadays software-defined networks (SDN) are being developed [7]. The main principle of SDN technology is placing conrol functions in separate server called controller. Control functions are moved from network devices to the controller.

The main functional element of a network device is the network processing unit (NPU). NPU is a specialized integrated curcuit that is used for packet processing in network devices.

Nowadays programmable NPU are being developed. NPU of this type allow to load new packet processing programs and define new data transfer protocols [1].

Packet processing in programmable NPU is done according to the packet processing program that implements data-plane algorithm. One class of such algorithms is a class of stateful data-plane algorithms. Stateful data-plane algorithms are used in data processing centers' networks and in telecommunication providers' networks [3]. The state of data-plane algorithm is a set of changeable variables, keeping their values on moving to next packet processing. The examples of such algorithms are load balancing with consistency [5], port knocking algotithm [2], failure recovery algorithm [5]. The main feature of stateful data-plane algorithm is the ability to introduce dependency of the process of packet processing on the properties of packets, processed by this NPU before. With the development of SDN and programmable NPU the task of implementing stateful data-plane algorithms on programmable NPU appears.

If NPU does not have state handling support, the state will be stored on the controller. But the state can change depending on the properties of packets, processed by the NPU. If the controller stores the state, the network device will access the controller for every packet that will lead to network device perfomance reduction, packet loss and network services work failures. So, implementing the hardware support mechanism for stateful data-plane algorithm state storage in NPU becomes actual.

The architecture of the considered NPU RuNPU does not support stateful data-plane algorithms. Thus, this work proposes the RuNPU architecture modifications that allow to use stateful data-plane algorithms with synchronization between the ports of the network device.

II. Formulation of the problem

It is necessary to propose the RuNPU architecture modifications that will allow to support stateful data-plane algorithms with state synchronization between NPU ports.

RuNPU architecture has the following features:

1) Each NPU port has its processing pipeline.
2) Processing pipelines are not connected to each other and operate in parallel.
3) Packet processing time on one pipeline stage is limited to 250 ticks.

Data-plane algorithm state is represented as a set of variables. The following symbols are introduced:

1) n is a number of pipelines in the NPU.
2) l is a number of stages in each pipeline.
3) q is a number of state variables available for each pipeline stage.
4) $P = \{P_1, \dots, P_n\}$ is a set of the NPU's pipelines.
5) D_{ij} is a j-th stage of i-th pipeline.
6) $S_{ijk}(t)$ is a value of k-th state variable of the stage D_{ij} in the time moment t.
7) $S_{ij}(t) = \{S_{ij1}(t), \dots, S_{ijq}(t)\}$ is a state of the stage D_{ij} in the time moment t.
8) $S_i(t) = \{S_{i1}(t), \dots, S_{il}(t)\}$ is a state of the i-th pipeline stage in the time moment t.

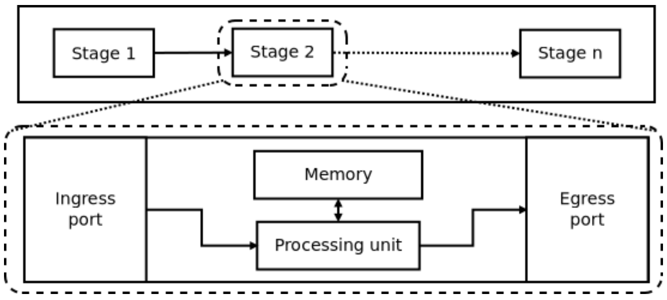

Fig. 1. NPU pipeline scheme.

Time moments $t_m, m = 0, 1, 2, \ldots$ are considered. They correspond to the moment of the next instruction execution beginning. The following situation is considered: during the instruction $i-1$ execution the state of the x-th pipeline has changed. It means that $\exists y, \exists k : S_{xyk}(t_{i-1}) \neq S_{xyk}(t_i)$. It is necessary to propose state synchronization system that will allow other pipelines to have an access to an updated state variable $\exists a \in N \cup \{0\} : \forall p \in \{1, 2, \ldots, n\} \Rightarrow S_{pyk}(t_{i+a}) = S_{xyk}(t_i)$ with minimum a. And the proposed modifications must take into consideraton the RuNPU architecture features and limits.

III. RuNPU architecture description

During the processing the packet header and metadata are moving through every stage of the pipeline (Fig. 1). Every stage processes the packet header for fixed number of ticks. Now this value is equal to 250 ticks. The stage consists of RISC processing core and memory device containing packet processing program. Packet header processing on every stage runs according to the following scheme. First of all, the packet header and metadata are loaded into the pipeline stage memory. After it, the program loaded into the stage memory device is executed. When the program finishes its work, the packet header and metadata are transferred to the next pipeline stage and new packet header and metadata are loaded into the current pipeline stage memory.

This architecture has two aspects that do not allow to use stateful data-plane algorithms. Firstly, pipelines do not have memory devices for the algorithm state. Pipeline stage memory device contains only program microcode, packet header and metadata. Secondly, NPU architecture contains a set of pipelines that work in parallel and do not have any connections to each other. It means that stages of different pipelines can not exchange data to update state in all pipelines. So, this NPU architecture does not allow to use stateful data-plane algorithms.

IV. Related work

The analysis of existing methods of memory synchronization in multicore systems was done. The overview is carried out according to the following criteria:

1) Synchronization delay, ticks (the paramerer a value).
2) Memory access time, ticks.
3) A possibility of long-time blocking.

A. Flexible Multiprocessor Locking Protocol

Flexible multiprocessor locking protocol (FMLP) is a mutex based synchronization algorithm for real time systems [4]. In this approach shared memory resources are protected using mutexes.

System operation time is divided into two phases: reading phase and writing phase. To get an access to a shared memory the process must acquire the appropriate mutex for reading or for writing depending on the access type. All memory reading operations start in the beginning of the phase and the phase is not changed until all requests in the current phase are done. Writing operations are done in the writing phase in FIFO order.

B. Combining Mechanism

The work [6] proposes the following approach: the memory is a set of memory devices. Devices and processing units are connected using a layered packet switched network.

The network has the following properties:

1) The network is nonovertaking. It means that if two messages are sent from one node in some order and arrive later at some other node then they arrive in the same order as they were sent.
2) A reply message is sent back using the same path as the request message.

The memory requests are Read-Modify-Write (RMW) operations. An RMW operation is equivalent to the execution of the following function:

```
function RMW(X, f)
begin
temp := X;
X := f(X);
return temp;
end
```

This operation applies the transformation f to the value X stored in memory and returns the old X value to the processing unit.

The combining mechanism is an approach to handle parallel requests to the same memory location. A memory request has the form $(id, addr, f)$, where id as a unique request identifier, $addr$ is an address of the memory location and f is an identifier of the transformation function. When two requests to the seme address are received on the same network switch they are combined into single request.

It is done in the following way: the messages are $(id_1, addr, f)$ and $(id_2, addr, g)$. They have the same memory address and conflict. Combining them into single request is done according to the following steps:

1) The switch saves id_1, id_2 and f and forwards the message $(id_1, addr, f \circ g)$.
2) When a reply message (id_1, val) reaches the switch, the saved information is retrieved by matching the ids. The message (id_1, val) is forwarded as a reply to to the first request and a message $(id_2, f(val))$ is forwarded as a reply to the second request.

TABLE I
ALGORITHM OVERVIEW RESULT

Algorithm	a value	Memory access time (read / write), ticks	Possibility of long-time blocking
FMLP	Depends on the blocking duration	1 / 1	+
Combining mechanism	1	Number of switches on the route to memory device	-
Write-update algorithm	1	1 / 2	-

C. Write-update algorithm

In the work [8] various cache coherent algorithms are overviewed. One of them is write-update cache coherence policy. According to this policy, if some processing unit writes data to the memory it is updated in the same cache blocks in other processing units' cache.

It can be applied to the NPU in the following way: each processing unit has its memory device for the packet processing algorithm state. Memory devices located at pipeline stages with equal depths are connected with a shared bus. When a processing unit reads data from the state memory the value is taken from the local memory device. When some processing unit writes data to the state memory it is written to the local memory device and updated in all other memory devices using the bus.

D. Result

The problem of the FMLP algorithm is that phases do not have fixed length, so this approach can lead to request locking for an unestimated period of time what can lead to pipeline errors because it has only 250 ticks to process a packet header. Two other approaches do not allow such behavior, so their applicability and exact properties will be evaluated during the experimental research. The overview results are shown in the table I.

V. PROPOSED NPU ARCHITECTURE MODIFICATIONS

A. Memory synchronization via the shared bus

In this approach the following modifications are proposed: memory devices containing the algorithm state are added to each pipeline stage. Memory devices on pipeline stages with equal depths are connected using a shared bus. The bus is used to synchronize data in the memory devices. When a memory cell in some memory device is updated, the new value and memory cell address are sent to other memory devices via the bus.

There are two operations available for the processing unit: read a value from a memory cell and write a value to a memory cell. When the value is read it is taken from the memory device that is on the same pipeline stage. When the value is written to the memory it is written to the memory device on the same pipeline stages and then is sent to appropriate memory devices in other pipelines.

B. Combibing mechanism

In this approach memory devices and processing units are connected using a packet switched network-on-chip with special switches that allow to combine memory requests. Memory requests have a Read-Modify-Write form. Memory request has a form of a tuple $(id, addr, f)$ where id is a unique request identifier, $addr$ is a memory cell address and f is a memory operation identifier. The response consists of two values: request identifier and the value that was in the memory cell before the memory operation was done.

This approach requires memory cells and network switches to have specialized arithmetic units to perform memory operations and memory request combinations.

VI. RUNPU SIMULATION MODEL

The RuNPU simulation model is used for the experimental research. It is written in Python programming language and allows to evaluate various NPU parameters such as pipeline throughput and power consumption. The simulation model input consists of two parts: pcap files with test packets and the packet processing program written in the assembly language. Each ingress port has separate pcap file with a sequence of packets. The output consists of the statistics and pcap files with packets that were sent via the NPU egress ports.

The simulation model consists of a number of the modules that are responsible for NPU modules work simulation. These modules are:

1) Main application module. This module is responsible for other modules initialization and configuration.
2) Pipeline module performs pipeline initialization and controls all pipeline components. There are 24 pipelines in the RuNPU simulation model.
3) InFIFO module reads the network packets from the pcap file that corresponds to the NPU ingress port.
4) OutFIFO module writes processed packets to the pcap file that correponds to the NPU egress port.
5) DE (Decision Engine) module represents the NPU pipeline stage and is responsible for packet header processing according to the packet processing program.
6) PacketMem module stores the packet bodies while packet headers are being processed in the pipelines.

For each proposed approach the simulation model was modified. For the synchronization method based on shared bus the following modules were added: memory devices for the data-plane algorithm state and the shared bus modules. For

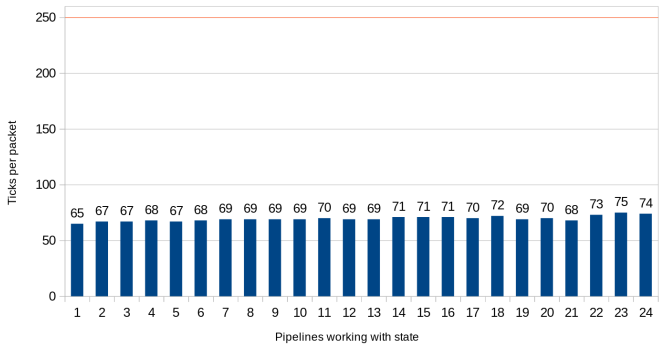

Fig. 2. Shared bus approach evaluation results.

the the combining mechanism approach the network module, network switch modules and memory devices for the algorithm state were added to the simulation model.

VII. Experimental research

For the experimental research a program that implements flowlet switching algorithm was written. This algorithm is used to balance packet flows on transport layer and requires state synchronization between the NPU pipelines. Pcap files with test packets were generated. Test packet sequences contain a number of packet groups. Each group consists of packets that belong to a single transport flow.

During the experimnetal research the test packets were processed by the simulation model and the required statistics was collected. The experimental research consists of a series of the NPU simulation model runs with a different number of pipelines working with the state. It allows to evaluate how the number of pipelines actively working with state affects the time required to process a packet header.

The experimental research results for the shared bus approach was finished (Fig. 2). It shows that the number of ticks required to process a packet header does not exceed 250 ticks.

VIII. Future work

In the future it is planned to finish an experimental research for the combining mechanism approach and determine what approaches are applicable.

IX. Conclusion

This work proposes the RuNPU architecture modifications that allow to synchronize stateful data-plane algorithm state between the NPU processing pipelines. An overview of the existing methods was done and two approaches were selected for further implementation and experimental research. A simulation model and test data for the experimental research were prepared.

X. Acknowledgements

This work is partilly supported by the Russian Foundation for Basic Research under grant 19-07-01076.

References

[1] Bezzubtsev S.O., Vasin V.V., Volkanov D.Yu., Zhailauova S.R., Miroshnik V.A., Skobtsova Y.A., Smeliansky R.L. ”An Approach to the Construction of a Network Processing Unit.” Modeling and Analysis of Information Systems 26.1 (2019): 39-62. (In Russ.)

[2] Bianchi, Giuseppe, et al. ”OpenState: programming platform-independent stateful openflow applications inside the switch.” ACM SIGCOMM Computer Communication Review 44.2 (2014): 44-51.

[3] Bifulco, Roberto, and Gábor Rétvári. ”A survey on the programmable data plane: Abstractions, architectures, and open problems.” 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR). IEEE, 2018.

[4] Brandenburg, Björn B., and James H. Anderson. ”Reader-writer synchronization for shared-memory multiprocessor real-time systems.” 2009 21st Euromicro Conference on Real-Time Systems. IEEE, 2009.

[5] Cascone, Carmelo, et al. ”Traffic management applications for stateful SDN data plane.” 2015 Fourth European Workshop on Software Defined Networks. IEEE, 2015.

[6] Kruskal, Clyde P., Larry Rudolph, and Marc Snir. ”Efficient synchronization of multiprocessors with shared memory.” ACM Transactions on Programming Languages and Systems (TOPLAS) 10.4 (1988): 579-601.

[7] Smeliansky, R. L. ”Software-defined networks.” Open systems 9 (2012): 15-26. (In Russ.)

[8] Stenstrom, Per. ”A survey of cache coherence schemes for multiprocessors.” Computer 23.6 (1990): 12-24.