BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

Stateful Packet Processing In High-speed Network Devices Described In P4

Pavel Kohout

cesnet

Abstract
Research and development in area of network technologies allow to increase speed of network traffic up to 100 Gbps meanwhile requirements for its security and an easy administration stay the same. A process of collecting network traffic statistics is important part in the defense of a network infrastructures but its performing is difficult in a high-speed network environment. Nowadays, a P4 language becomes powerful tool for the network administrators thanks to the platform independence and its ability to describe whole packet processing pipeline. The aim of this work is to extend existing stateless solution developed at CESNET association target to FPGA platform by support of stateful processing at speed 100 Gbps. This paper describes the designed system architecture for stateful processing realization in P4 described device respecting requirements for its resources or rate. Performance testing has shown that device is capable of achieving the target throughput of 100 Gbps for limited number of used stateful memory requests in context of a table or an user action.

Keywords: P4 — Stateful packet processing — FPGA

Supplementary Material: Repository with P4 sources

*xkohou15@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

With increasing numbers of internet users collecting of network traffic statistics is an indispendable part of network management and its security. Stateful packet processing is appropriate way for collecting information about received traffic and it allows to create complex applications that behave differently based on the history of received traffic to detect some specific network incidents. For satisfaction of network administrator's demands such devices have to be capable of calculating hundreds of gigabits of statistics per second with ability of the functionality reconfiguration.

Nowadays, P4 language is one of the available solutions for describing the devices mentioned above. Platform independence allows developers to choose the target platform according to the required features. One of P4 solutions was created by CESNET. Current stateless implementation of P4 language is capable to reach 100 Gbps device rate using FPGA (Field Programmable Gateway Array) technology. In order to enable stateful processing, it is necessary to extend the current solution by support of stateful processing provided by P4 at 100 Gbps rate with atomic execution of operations working with stateful memories.

Barefoot Tofino [1] is another existing implementation of fully P4 programmable and high-performance networking solutions. It is built using a Protocol Independent Switch Architecture. Stateful processing is

supported but available stateful memory is constrained to reach the highest performance.

The result of this article is about extending the current P4 implementation by supporting stateful memories. The implementation of registers and counters P4 primitives is offered in two modes. The first one respects the resource limits on FPGA chips and therefore this memory is allocated in BRAM (Block RAM). Second mode requires more chip resources and implements memory in the register arrays allowing to make read or write operation in one clock period. To ensure atomicity of execution of accessing stateful memories a lock mechanism was developed. This mechanism guarantees atomicity of operation for each use case described in P4 using stateful memories.

The implementation realizes stateful processing described in P4 language and current solution is capable of reaching 100 Gbps device rate for some use cases. These use cases correspond to just one request to stateful memory of registers and counters in context of one user action. In case of more requests the device rate is decreased and an user is informed about reachable device rate. BRAM register configuration saves chips resources effectively and the atomicity of execution is guarenteed by lock mechanism.

2. P4 language

P4 (Programming Protocol-independent Packet Processors) is an open source, high level, platform-agnostic language for programming protocol independent processors. Its main purpose is to provide a way to define packet processing functionality of network devices respecting possibility of field reconfiguration, protocol and target platform independence.

Syntax and structure of P4 is described by standards. Nowadays $P4_{16}$ and $P4_{14}$ standards are available and they are able to describe five aspects needed for definition of packet processing device as it is described in [2]. These five aspects are: header formats, packet parsing, control program, table and action specification. **Header formats** describe protocol headers recognized by device. **Packet parser** defines order of protocol headers parsing in form of state machine. **Table specification** allows to define how the extracted header fields are matched in possibly multiple lookup tables meanwhile **action specification** defines a set and order of actions that may be executed for packets. In the end, a **control program** puts all previous parts together, defining control flow among the tables.

$P4_{16}$ is an younger standard of P4 language and user is not restricted by the set of usable actions. A desired functionality is realized by an *extern* construction.

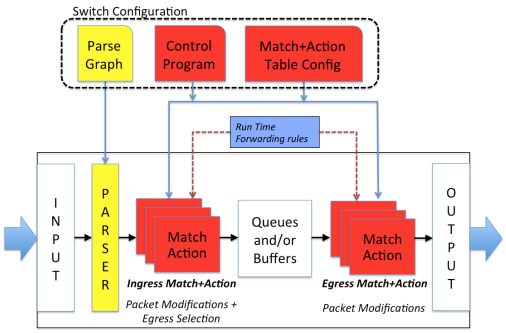

Figure 1. Model of P4 device (taken from [3])

The *extern* object describes a set of methods that are implemented by an object. Implementation of these methods is provided by user.

$P4_{14}$ is a second older standard for which this work is reserved and therefore it will be described in more detail. A core of $P4_{14}$ standard is abstract model determining the order of individual parts of P4 devices and it is highlighted in the figure 1. The first module parser is used for filling header fields with the incoming network data. These data fields are passed to the ingress Match+Action pipeline where packet is modified according to desired functionality. Also queuing mechanism is present to implement data distribution based on configuration from ingress Match+Action pipeline. Furthermore an egress Match+Action pipeline is executed and data from protocol headers fields are transformed back to the form of a network packet. The modification of header fields is performed in user actions. These actions are composed of primitive actions. Primitive actions are a set of functions provided by $P4_{14}$ standard for realization of the user desired functionality.

Stateful processing in P4

The P4 provides counters, meters and registers for maintaining state for longer than one packet. Together they are called stateful memories. An individual counter, meter or register is referred as a cell. Stateful memories are organized into named arrays of cells. A cell is referenced by its array name and index. The stateful memories in P4 can be global – they can be referenced by any action of any table or to be static – bound to one table instance. P4 also allows direct access to memories. In case of a direct mode, memory cell is allocated for each entry of table, which cell is bounded to.

Counters are stateful objects used for calculating the number of certain phenomenon occurrences. They determinate the number of proceeded packets

or bytes matched to desired table entry according to a type of declared counter. A *bytes* type counter gets incremented by the length in bytes meanwhile a *packets* type gets incremented by just one. A *packets_and_bytes* type counter is a combination of both previous counter types and it is composed of two subcounters, bytes and packets counter type. The counter is incremented whenever *count* primitive action is executed (in case of static counters) or a packet enters to table (in case of direct counters).

Meters are stateful object that measures the data rate, either in packets or bytes per second. The result of meters execution is on of three colors: green, yellow and red. Specific algorithm for meter implementation is not given by P4 specification and it is left up to the user. Available meter types *bytes* and *packets* are same as counter types mentioned above. The execution of requests to direct or static meters is also the same as in case of counters. Primitive action working with meters is *execute_meter*.

Registers are stateful memories that can be read and written in actions and they can be used in the most general way. Both, direct and static registers are accessed only through primitive actions *register_read* and *register_write*.

3. Mapping of P4 statistics to FPGA

A realization of large data storage in the FPGA is a complicated quest because of limited chip resources. Using of block RAM (BRAM) is the most common way but the delay between request to memory and the result is critical in a design applying pipelined processing with requirements for work with actual data. Therefore a data storage realization in the distributed RAM is also provided to minimize a delay. The choice of desired implementation is left to the user.

A core of the implementation is built around register component. Its design provides an instantiation of chosen memory realization and it creates the circuits needed for a correct communication with software program. A bus communicating with software is capable of transferring 32 bit width data in one message but a bit width of register cells is variable. Therefore, a special array of 32 bit width registers is created to hold the currently transmitted data over the bus. The array length is determined by bit width of register cells and it is calculated according to the equitation 1.

$$arrLength = \lceil cellWidth/32 \rceil \qquad (1)$$

When a request for stateful data of device appears read data are stored in array and 32 bit width messages starting at $arr\_item_{[0]}$ are sent over the bus to software iterating over all array items. In case of writing data to a stateful memory of device, array is filled by data transmitted over the bus and write signal to cell is generated when last message is received by the array.

The P4 language structure allows more stateful objects arrays bound to one table. To reduce an usage of chip resources an *envelope* component was developed. This component enables to connect register module with more registers/counters end points. The final width of cell is determine by the widest stateful object of desired type. The end point represents a call of primitive action working with stateful memory or request of direct counter.

Meter implementation is based on RFC 2697 [4], that serves as reference provided by [3]. This reference requires more complex architecture, therefore meter design differs from previous ones. Meters component is designed to be capable of calculating packet color for each clock period. This feature is reached by breaking meter to many subcomponents as shown in the figure 2. Meter component is composed of two *Bucket_pipeliner* components. The aim of meter is calculation of output color based on received flags from *Bucket_pipeliner* components and determination of a bucket to update. *Bucket_pipeliner* wraps a functionality of one meter.*Val_selector* detects multiple requests to one address in a row and it makes a sum of these value to secure data consistency in further processing. *Inc_array* component keeps values of last bucket actualization. These values are used for calculation of value to increment a target bucket. To secure data consistency *bucket_cache* component was implemented. It keeps dual information for one request to memory.

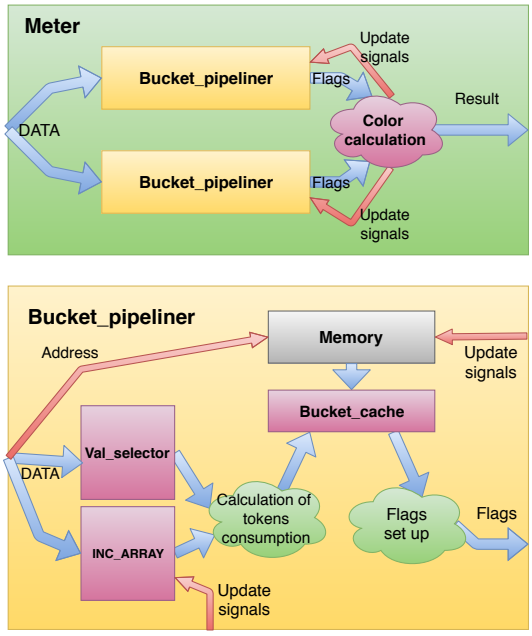

Figure 2. Design of meter component.

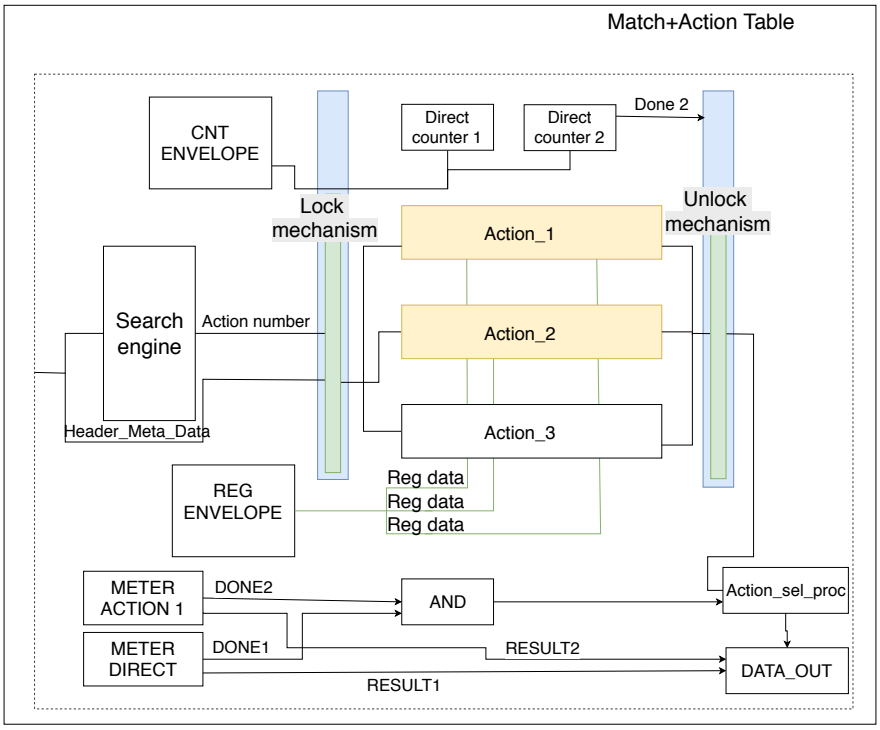

Figure 3. Mapping of statistics to P4 pipeline

First one represents read value from memory meanwhile second one keeps a calculated value which will be written back to the memory. Second value is used in case of quick address switching when the calculated value is not yet updated.

Figure 3 shows an integration of stateful memories described above to a P4 pipeline. The figure depicts a table where all kinds of stateful memories were used. Yellow colored actions indicates an usage of primitive actions working with registers. In *Action_1* two primitives to request registers were used and in *Action_2* just one. The table contains two direct counters. Direct counters are not bounded to user action but they are allocated in the context of table. An execution of direct meters and user actions is parallel. The data pipeline is locked when packet enters to the user actions. An unlock mechanism is applied when every request to stateful memory was handled. Meters are not wrapped to one component as in the case of counters or register. For each declared meter is instantiated one meter component and the result is added to data flow at the end of table context. Therefore it is not allowed to use the meter result in the primitive actions of the current table. The meter implementation is designed to maximize device performance but it is still decreased in case of multiple requests to the same static meter during one user action.

4. Results

In this section achieved results will be commented. At the time of writing this article registers and counters were fully integrated to a P4 processing pipeline. Meters are still in the phase of integration but an implementation of design described above was done. To prove a validity of P4 register and counter concept an existing verification was extended and for purpose of performance testing, a list of P4 stateful use cases was created. A card COMBO-100G2Q was chosen as target device for a testing process. This card is equipped with FPGA Virtex-7 H580T chip enabling a running of complex designs.

Testing of functionality was performed using CESNET verification environment [5] and an existing verification written in SystemVerilog. As a simulation tool was used a program ModelSim. The verification sends thousands of transactions through device and it compares an output with a reference solution of behavioral model. Behavioral model is a software P4 implementation provided by P4 language Consortium. Registers were tested on five hundreds of thousands transactions. A sequence of *write* and *read* register operations accessing same memory cells was called to not defect data included in the packet.

The list of use cases contains seven P4 programs using stateful memories in the different way. This list was created to show an influence of stateful objects to device resources and performance. A core of use cases is built around three tables. Here is a list of P4

programs provided for testing:

reference.p4 It is the reference program without an usage of stateful objects with which others will be compared.

simple-reg This use case declares just one static register array with three 8 bit width cells. One *register_write* operation is called.

reg-bram It is extension of simple-reg use case by adding two direct register arrays implemented in BRAM bound to table with 512 table entries and 4 primitives operations execution in one user action.

simple-cnt One static counter array with 5 items of 32 bit width. Just one call of *count* primitive action.

cnt-direct-bram Direct counter array implemented in BRAM bound to table with 512 entries. The width is 11 bits.

reg-cnt-together Direct register array implemented in BRAM bound to table with 512 entries and cell is 16 bit width. Two counters are implemented in same table. First counter array has 5 cells of 32 bit width. The second array has 4 cells of 16 bit width. In the context of user action two calls of register and counters are executed.

cnt_dir-multi-reg This use case was created for testing of direct counters with registers. Two static counter arrays with 5, 4 items, one direct counter array and two direct registers array are declared in table with 512 entries. Another table has one static *packet_and_bytes* counter array with 444 items and register array with 3 items.

Table 1 shows resources needed by individual use case and a maximal design frequency. Target frequency of design is 200 MHz. Results show that current implementation saves chip resources efficiently and it is capable of running design at a desired frequency.

Use case	LUT([%])	BRAM([%])	f [MHz]
reference	125554(35)	219(23)	202.8
simple-reg	130671(36)	219(23)	200.1
reg-bram	131206(36)	220.5(23)	201.7
simple-cnt	131217(36)	219(23)	201.7
cnt-direct-bram	130386(36)	219.5(23)	204.8
reg-cnt-together	130930(36)	220.5(23)	203.2
cnt_dir-multi-reg	131277(36)	223.5(24)	202.9

Table 1. Table of resources for individual use cases

The performance of device is affects by a lock mechanism interrupting table processing for an amount of clock periods. The final device throughput caused by the lock mechanism can be evaluated according to an equation 2 where f is device frequency, L_i means an user action latency and $Cnt_{dir} = max(Cdir_{reqCount} - 1,0)$. $Cdir_{reqCount}$ signs for count of direct counter accesses in table from where L_i action latency was given.

$$Thr = \frac{f}{max(L_1,L_2,L_3,...L_n) + Cnt_{dir} + 1} \quad (2)$$

The user action latency is determined by primitive actions used in it. All supported primitive actions except those working with stateful memories are implemented in a combinational logic. Therefore, an user action without or with just one access to stateful memory (lasts one clock cycle) has latency 1. Otherwise, the action latency (L_i) can be evaluated according to equtations below:

$$L_i = max(R_{op}, C_{op}, 1) \quad (3)$$

$$R_{op} = |R_{wr}| + |R_r| \quad (4)$$

$$C_{op} = |C_{count}| \quad (5)$$

$|R_{wr}|, |R_r|$ *and* $|C_{count}|$ sign an amount of clock periods needed to execute all *register_write, register_read* and *count* primitive actions in user action. An amount of clock periods needed by individual primitive actions are listed in table 2.

	Implementation	
Action	**Distribute RAM**	**BRAM**(output reg)
register_write	1	2(2)
register_read	1	1(2)
count	1	1(2)

Table 2. Clock cycles needed for execution of stateful primitive actions

Figure 4 shows results of performance testing on network with 100 Gbps rate. Each use case was tested for three specific network traffic. The device and the generator of traffic were configure to target whole proceed traffic to actions using access to stateful memories. Therefore, a graph in figure 4 shows device behavior for an extreme network traffic situation that is not usual in normal network traffic. For each situations is dedicated one column. First mode of traffic is a situation when each proceed packet size is 64 bytes. Thanks to its size, a network link is capable of sending more packets per seconds but this causes the reduction of throughput, as device is currently able to process only one packet per clock cycle. Device throughput of a network traffic composed from packets of 64 bytes is shown as blue column. The red column is earmarked for a network traffic of 1526 bytes size. In this case the

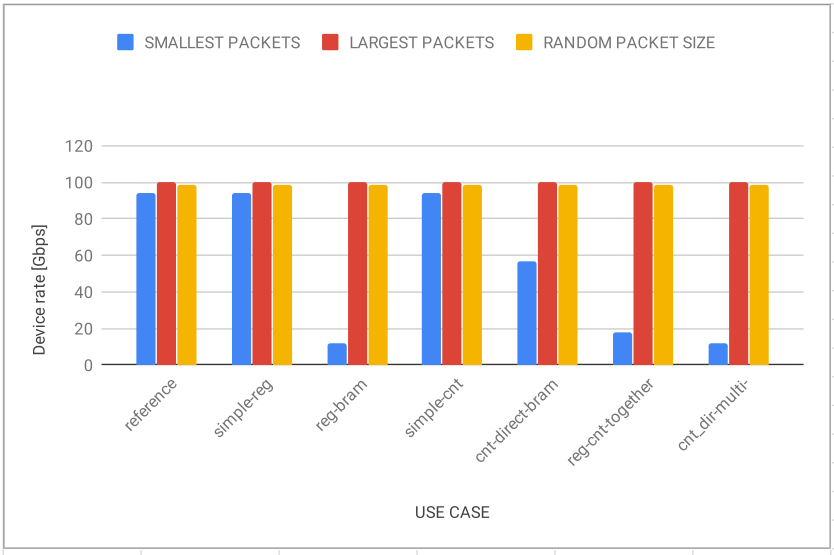

Figure 4. Device throughput of individual use cases

network link is capable of sending less packets than in previous one and the device throughput is sufficient to process all packets for each use case. The last yellow column signs a network traffic of packets from 64 to 1526 bytes size.

Results have shown that device throughput for the largest (red) and random size packets (yellow) is same for each use case. For this kind of traffic the device throughput is sufficient even if stateful memories are used. The lock mechanism affects performance in case of the network traffic composed from packets with small size. Results confirm a prediction of new device throughput derived from stateful memories usage, based on equation 2.

5. Conclusion

The subject of this paper is the realization of stateful processing in high speed network devices described in P4 language, the description of the design of components for stateful processing, an integration of these components to existing stateless solution with a security of atomic execution of stateful operations and their influence to the final device throughput.

The solution of stateful processing in P4 provided by this paper is capable of reaching to a desired device throughput for a variety of P4 use cases with stateful memories on only a packet larger length. However, to achieve better performance it is possible to divide requests to stateful memories into more tables and for a transfer between these tables a meta data can be used. The current implementation saves a chip resources efficiently.

In the near future meter's integration should be finished. Another challenge is an extension of current solution for a support of external memories to store stateful information without significant negative effect to device performance.

Acknowledgements

I would like to thank to my supervisor Doc. Ing. Jan Kořenek Ph.D. for his help with preparing this paper and a constant support during work on this project. I would also like to thank to my project leader Ing. Pavel Benáček Ph.D. for his constant enthusiasm, willingness to share his experience and his assistance with a design of components and their implementation. In the end, I would like to thank to my colleague from CESNET Ing. Radek Iša for providing verification environment.

References

[1] Barefoot Networks. Barefoot tofino. https://barefootnetworks.com/products/brief-tofino/.

[2] Pavel Benáček. *Generation of High-Speed Network Device from High-Level Description*. PhD thesis, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[3] P4 Language Consortium. $P4_{14}$ specification. https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf.

[4] J. Heinanen and R. Guerin. A Single Rate Three Color Marker. RFC 2697, RFC Editor, September 1999.

[5] R. Iša, P. Benáček, and V. Puš. Verification of generated rtl from p4 source code. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 444–445, Sep. 2018.