2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)

Asynchronous Extern Functions in Programmable Software Data Planes

1st Dániel Horpácsi
Eötvös Loránd University
Budapest, Hungary
danielh@elte.hu

2nd Sándor Laki
Eötvös Loránd University
Budapest, Hungary
vopraai@elte.hu

3rd Péter Vörös
Eötvös Loránd University
Budapest, Hungary
matej@elte.hu

4th Máté Tejfel
Eötvös Loránd University
Budapest, Hungary
lakis@elte.hu

5th Gergely Pongrácz
Ericsson Research
Budapest, Hungary
gergely.pongracz@ericsson.com

6th László Molnár
Ericsson Research
Budapest, Hungary
laszlo.molnar@ericsson.com

***Abstract*—Target-independent packet processing languages support diverse hardware and software targets by generalizing over the set of primitive operations (extern-functions) available on the target. In P4, the language specification does not specify whether the invocation of an extern function is synchronous or asynchronous — supposedly synchronous by default. However, in some use cases, it makes more sense to invoke such functions in an asynchronous way and let the thread keep processing packets while the extern operation is being performed by a dedicated resource or accelerator device. In this paper, we propose a method for transparent description and efficient implementation of asynchronous extern function calls in P4-programmable software data planes. Our DPDK-based early prototype relies on the concept of coroutines used for saving packet contexts and manual switching between them. The overhead of the proposed solution is analyzed with a packet encryption case study.**

***Index Terms*—P4, Asynchronous packet processing, Software data plane**

I. Introduction

P4 [1] is a target- and protocol-independent packet processing language which enables high-level description of packet handling algorithms. P4 programs can run as software on general-purpose processors or can control a dedicated network hardware. This paper focuses on software data planes that play a key role in modern telecommunication systems, insomuch that in data centers most packets passing through virtual machines are processed by software switches.

Existing solutions usually carry out the packet processing entirely on CPU cores, according to a run to completion execution model, leaving room for further optimization regarding scheduling and offloading. The server computers hosting these software data planes can be equipped with hardware accelerators (e.g. Intel QAT, AMD CCP, CAVIUM Octeon) and other specific computational resources (such as GPUs or application-specific processors), which can take their part in the packet processing pipeline by offloading specific tasks to them. Also, for better usage of limited CPU resources, a partition of CPU cores can be dedicated to execute specific parts of the pipeline, e.g., external functions (extern functions) like encryption/decryption, compression or checksum calculation. Such pipeline elements, or external functions, may appear in the middle of the control flow, therefore efficient offloading of these external functions requires the ability of asynchronous function invocation.

This paper discusses a proof-of-concept implementation of asynchronous extern functions in the P4-programmable software data plane called T4P4S [2], using the concept of coroutines.

The authors thank the support of the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013) and Ericsson Hungary Ltd.

II. Asynchronous Extern Functions

Concept. The P4 language handles the diversity of targets by making itself extensible via so-called extern objects and extern functions. Externs represent functionality in the architecture model that is implemented by the given target (either in software or hardware). Depending on the nature of the functionality, invocations of an extern function may be implemented to be synchronous or asynchronous. In case of asynchronous invocation the extern function is executed on a dedicated resource (e.g., in a hardware accelerator card, co-processor or just a dedicated thread) in a separate context and the packet processing thread is not blocked for the function execution, it can keep handling other packets while the operation is being performed. This option pays off if the operation is complex enough making it worth handed over to a dedicated resource despite the costs of the transmission and context switch. Suppose that a P4 control invokes an architecture-provided encryption functionality (e.g., accelerated by a co-processor) whose execution takes notable CPU time in which it should not block the fast path. In this case the asynchronous invocation is the good choice.

When executing extern functions asynchronously, the control flow exits the pipeline, the function gets processed by a separate unit, and on completion the control flow is directed back to the point where the extern call happened. From

978-1-7281-4387-3/19/$31.00 © 2019 IEEE

the packet handling point of view, the processing block is suspended on the extern call and is resumed after the function's return. To support this, when the execution reaches the extern call, the packet context is saved and the packet with the context information is forwarded to an input queue of the dedicated processing unit (different thread or hardware unit). All concurrently processed packets need a separate context storing local variables, metadata, raw and parsed packet representations, etc. Note that other packets should not affect the context of the offloaded, asynchronously processed packet. After the extern call has returned, the packet is forwarded back to the processing thread with its packet context through a buffer. The context is restored for the packet and the execution of the original control block is continued right after the asynchronous extern call. Note that in many cases before context saving or after context loading additional preparation steps may be needed, e.g. preparing the packet representation for encryption or parsing additional headers after decryption. Though the support of asynchronous invocation increases the complexity of the P4 compiler and the generated software data plane, the benefits are obvious when it comes to offloading specific tasks to hardware accelerators.

Prototype Implementation. In order to demonstrate the proposed concept, we have modified our DPDK-based open-source P4 compiler and software switch, called T4P4S [2], to provide experimental support for asynchronous invocation of extern functions in P4 programs. With the modifications, the packet processing threads can handle multiple packets concurrently. Concurrent packet processing within threads is achieved by employing asynchronous function execution by turning packet handlers into coroutines allowing us to suspend and resume packet processing at asynchronous extern calls. In our prototype implementation, coroutines are realized with the ucontext (user thread context) module in C library System V.

To illustrate the applicability of the method and to carry out performance measurements we have created a simple P4 program as a benchmarking example that extends a simple L2 forwarding example with two asynchronous extern calls `encrypt` and `decrypt` added to the v1model P4 architecture. They are responsible for encrypting and decrypting a part of the serialized packet resp., implemented by DPDK's OpenSSL-based Cryptography PMD. The extern functions are called after the original L2 forwarding block applying two exact-matching tables `smac` and `dmac`. To denote where asynchronous invocation shall be used we embedded the extern call into an annotated block in the P4 program: `@async(deparse=..., parser=...) { ... }`. Annotation parameters are used to define which deparsing method shall be applied before and which parsing method after the execution of the asynchronous extern. In our example, the original parse and deparse methods of the L2 forwarding can be applied by both asynchronous calls. Note that encryption and decryption methods require the serialization of the packet content in advance, but other extern functions may work without these steps, thus deparse and parse parameters can be left undefined.

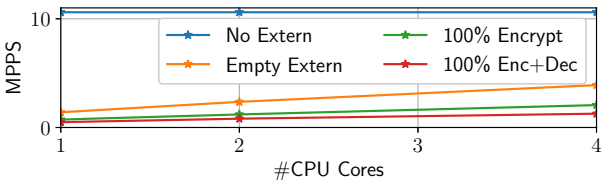

Fig. 1. Performance w/wo async. extern calls

The measurements have been carried out in our local testbed consisting of two identical nodes (AMD Ryzen 1900X 8C/16T 3.8 GHz, 128 GB RAM): a traffic generator using DPDK's Pktgen tool to generate test traffic and a P4 switch executing the proposed implementation, interconnected with two 10 Gbps links. Figure 1 depicts the overhead of asynchronous extern calls including context creation and context switching steps. The curve "No Extern" illustrates the baseline performance of a simple L2 forwarding without asynchronous calls and context management (11 MPPS). In the "Empty Extern" case, the P4 switch creates the packet contexts and after the L2 block calls an empty asynchronous extern function. The main purpose of this measurement is to quantify the overhead of context creation and switching. One can see significant performance drop compared to our baseline measurements (4 MPPS with 4 CPU cores). The "100% Encrypt" case illustrates when an asynchronous extern operation encrypting the packet payload (using OpenSSL DPDK driver) is called, while in the "100% Enc+Dec" case two asynchronous extern operations are called: first encrypting and then decrypting the payload, resulting in multiple context switching.Our main finding is that context creation of the ucontext library is very expensive, causing significant decrease in packet processing performance. After the context is created, context switching has non-significant costs. The two extern functions are currently implemented by OpenSSL whose overhead is also visible in the figure.

Discussion. Though the proposed method makes the application of asynchronous extern functions possible in P4 programs, there are a number of open issues not or only partially managed in our solution. 1) Some asynchronous operations e.g. encryption, compression may require the partial serialization of packet headers while reparsing may be needed for others like decryption or decompression. 2) In our early prototype context handling is based on the ucontext library that saves or reloads the whole stack of the generated switch program at given checkpoints, resulting in too large overhead for practical usage. 3) The proposed approach can be extend in a straightforward way to support not only asynchronous extern functions but asynchronous execution of P4 instructions or blocks of instructions (e.g. complete controls).

REFERENCES

[1] P. Bosshart et al., "P4: Programming protocol-independent packet processors," ACM SIGCOMM Computer Communication Review vol. 44, issue 3, 8795, 2014.

[2] P. Vörös et al., "T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors," In Proceedings of IEEE HPSR 2018. 2018.