TimerTasks: Towards Time-driven Execution in Programmable Dataplanes

Raj Joshi
rajjoshi@comp.nus.edu.sg
National University of Singapore

Ben Leong
benleong@comp.nus.edu.sg
National University of Singapore

Mun Choon Chan
chanmc@comp.nus.edu.sg
National University of Singapore

CCS CONCEPTS

• **Networks → Programmable networks**.

1 INTRODUCTION

Current programmable dataplanes provide an event-driven execution model based on the match-action paradigm [4]. An *action* is triggered by the arrival of a packet as defined by the match specification. Any logic that is programmed executes only when there is a packet event, and no action will be triggered otherwise. This behavior is expected since programmable dataplanes were primarily designed to realize reconfigurable packet forwarding in the hardware [3].

In recent years, programmable dataplanes are being used to implement advanced applications not earlier envisioned, such as in-network caching [8, 9], consensus protocols [7, 15] and others. The implementation for many of these systems require an action to be taken even in the absence of a packet arrival. For example, in the Raft consensus protocol [10], a missing *heartbeat* from the leader is expected to trigger a leader election after a timeout. Existing programmable dataplane architectures however do not support such time-driven semantics.

In this poster, we first describe how time-driven execution in the dataplane can help protocols and applications offload new functionality to the dataplane (§1.1). We then propose a new abstraction called *TimerTasks* that allows us to express the required time-driven semantics in an extended P4 language syntax (§2). We implement the TimerTask primitive on top of existing event-driven hardware (§2.1) and demonstrate its utility via three novel applications (§3). Finally, we discuss the limitations and future work (§4).

1.1 Case for time-driven execution

In this section, we illustrate how time-driven execution can enable the implementation of new functionality in the dataplane.

Fast link failure detection. Mechanisms such as BFD [5] are limited to sub-second link failure detection times with existing control plane implementation. A dataplane implementation of such a mechanism can do better, but it is not straightforward with the current event-driven execution model. This is because such a protocol requires action to be taken after a *timeout*, in the *absence* of a packet. Time-driven execution in the dataplane will allow actions to be taken based on aggressive timeouts (order of μs), thereby enabling orders of magnitude smaller detection times. While link failure detection is a network-specific example, it represents an entire class of application-level heartbeat protocols that are used in distributed computing frameworks (e.g. Spark [13], and Heron [6]).

Avoid aging metrics, refresh them! In systems such as CONGA [1] and DistCache [9], switches communicate certain metrics to each other by piggybacking them on top of regular traffic. The switch that receives the metrics needs to *age* them since they may not remain up-to-date when there is insufficient regular traffic. With time-driven execution, it would be possible to *refresh* the metrics instead of aging them. In particular, the switch that is the source of a metric can explicitly send the updated metric to the recipient switch even when there is no regular traffic between them.

High-resolution network measurements. High-resolution network measurements are required to understand the behavior of today's high-speed networks. Zhang et al. implemented a measurement framework that could poll counters from the switch ASIC every 25 μs using the switch CPU [14]. However, this interval was found to be too coarse to reliably measure the duration of link utilization bursts (cf. Fig. 3 in [14]). With time-driven execution in the dataplane, the counters can be retrieved at regular and much smaller intervals.

2 TIMERTASK ABSTRACTION

After studying the above use-cases for time-driven execution, we propose the following execution semantic: *if a condition (predicate) remains true for a timeout interval, then execute certain action.* This semantic can also express the periodic execution of an action by setting the condition to be always true.

TimerTask. We formalize the above semantic as a high-level abstraction called *TimerTask*. The TimerTask abstraction is designed as an extension to the P4 language. It has three parts: (i) a Reset-Match specification, (ii) a timeout value, and (iii) an action. The Reset-Match specification is similar to the P4 match specification, the timeout value is a non-zero positive constant, and the action is a standard P4 action. For a given TimerTask, the system runtime implements a timer with value equal to the timeout. Unless reset preemptively, the timer counts down to zero and then repeats. If a packet matches the Reset-Match specification (which we call a "hit"), the timer is reset. But if no packet matches the Reset-Match specification for the timeout interval, the timer expires and the corresponding action is executed. Listing 1 shows a sample TimerTask for implementing link failure detection. Semantically, this TimerTask expresses the logic that if there is no packet from an adjacent switch (identified by the ingress_port) for 3 μs, then send a ping request to that switch. Like multiple entries for a P4

SIGCOMM Posters and Demos '19, August 19–23, 2019, Beijing, China
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6886-5/19/08…$15.00
https://doi.org/10.1145/3342280.3342310

```
timertask link_failure_detection{
    resetmatch = {
        standard_metadata.ingress_port: exact;
    }
    timeout = 3;  // in microseconds
    action = send_ping_request;
}
```

Listing 1: TimerTask for Link Failure Detection

match-action table, each TimerTask can have multiple "instances" for different value pairs of Reset-Match and action data.

2.1 Implementing TimerTasks

TimerTasks can be implemented as a new hardware extension to a programmable switch machine model such as Banzai [12]. In this poster, we demonstrate how they can also be implemented on top of existing programmable hardware so as to be readily deployable. The main challenge is in implementing time-driven execution on top of existing event-driven hardware.

Inspired by the Linux operating system, our key idea is to have a periodic-event framework that uses periodic-events to orchestrate time-driven execution. Periodic-events are achieved by generating periodic packets (called *dataplane ticks*). The actual implementation of dataplane ticks depends on the target hardware.

The periodic-event framework maintains timers for all the TimerTask instances using the on-chip memory. For each dataplane tick, the timers are decremented and for the expired timers, the corresponding actions are executed. The resetting of timers on a Reset-Match "hit" is achieved using match-action tables. Also, TimerTask actions can re-purpose the dataplane tick packets as *new* packets to send out. We omit further details due to space constraints.

Compiling TimerTasks. TimerTask is a new P4 "object" that can be coupled with rest of the P4 program to form the required logic. A TimerTask compiler can first add the TimerTasks into the periodic-event framework (implemented in P4) and then merge the framework with the rest of the program. The resulting P4 program can be then fed to a regular P4 compiler. In the current design, TimerTasks are specified at compile time while the instances are added during the runtime. The TimerTask compiler can also generate a TimerTask runtime layer that allows for adding and deleting TimerTask instances. Under the hood, the TimerTask runtime adds or removes entries from tables of the periodic-event framework.

Related Approaches. It is possible to implement simple periodic tasks using manually crafted match-action tables and periodic packets. However, a TimerTask provides a high-level abstraction and richer semantics such as taking an action in response to the absence of a specific packet or a condition.

3 EVALUATION

We evaluated TimerTasks in terms of their utility by conducting three proof-of-concept case studies. Our testbed consists of two Barefoot Tofino switches and two x86_64 servers connected in a linear topology via 10 Gbps links.

We used Tofino's on-chip packet generator to generate the dataplane ticks. Being on-chip, the packet generator is a reliable source of dataplane ticks as there are no issues related to link disruption and the generated packets have very low probability of getting lost before entering the pipeline. For timely execution of TimerTasks, it is important that the dataplane ticks have low-jitter. We configured Tofino's packet generator to generate packets at different intervals (which we call a "tick period") and used dataplane timestamps to measure the actual packet inter-arrival times in the pipeline. Our measurements show that the 99th percentile error in the inter-arrival times is less than 0.1% for different tick periods from 1 μs to 100 ms. In summary, on-chip packet generators can provide reliable and timely dataplane ticks.

Fast Link Failure Detection. We implemented a link failure detection protocol that uses normal traffic as an indicator of connectivity, but explicitly pings the adjacent switch when there is no packet for 3 μs (Listing 1). Our experiments show that this protocol can detect link failures within 6 μs, which is orders of magnitude faster than BFD [5].

Metric Refreshing. We implemented a novel metric refreshing (updating) mechanism to explicitly update piggybacked metrics in lieu of sufficient normal traffic to piggyback them. The benefit of such a scheme for network [1] and application load distribution [9] remains to be quantified.

High-resolution Network Measurements. We implemented a periodic TimerTask that is able to read the switch counters at 1 μs intervals providing 25 times finer resolution than the previous high-resolution measurement [14]. Unlike [14], our solution runs entirely in the dataplane and therefore avoids the overhead and unpredictability involved in polling dataplane counters from the control plane.

4 DISCUSSION AND FUTURE WORK

Our preliminary design of the periodic-event framework is simple and cannot decrement all the timers with a single dataplane tick packet. This is due to the restrictions on updating the on-chip transactional stateful memory. As a result, one dataplane tick packet is required for each TimerTask instance. This adds additional pipeline processing overhead when supporting a large number of TimerTask instances.

However, the typical network load conditions are such that there is almost always spare pipeline processing capacity available [14]. Further, pipelines are often provisioned for a few extra ports other than the front-panel ports for facilitating packet generation [2] or backplane data connections [11]. For a 100 Gbps pipeline, a single 100 Gbps extra port provides a spare headroom of up to 150 Mpps which can be used to support 150 TimerTask instances with a tick period of 1 μs without affecting the normal traffic from the front-panel ports. In practice, we do not expect a large number of TimerTasks instances with very small tick periods. For a tick period of 100 μs, a spare headroom of 150 Mpps can support 15,000 TimerTask instances.

There is still further scope to improve the periodic-event framework such that multiple TimerTask instances can be updated with a single dataplane tick packet to further scale the system. Correspondingly, the TimerTask compiler would need to become more sophisticated to combine multiple TimerTasks appropriately. In the longer term, we argue that TimerTasks should be supported natively in hardware, which we plan to study as well.

REFERENCES

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of SIGCOMM*.

[2] Barefoot Tofino 2019. https://goo.gl/cdEK1E.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM CCR* 44, 3 (2014), 87–95.

[4] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *Proceedings of SIGCOMM*.

[5] Dave Katz and Dave Ward. 2010. Bidirectional Forwarding Detection. RFC 5880.

[6] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter Heron: Stream processing at scale. In *Proceedings of SIGMOD*.

[7] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. 2016. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering. In *Proceedings of OSDI*.

[8] Jialin Li, Jacob Nelson, Xin Jin, and Dan RK Ports. 2018. *Pegasus: Load-Aware Selective Replication with an In-Network Coherence Directory*. Technical Report UW-CSE-18-12-01. University of Washington.

[9] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. DistCache: Provable load balancing for large-scale storage systems with distributed caching. In *Proceedings of FAST*.

[10] Diego Ongaro and John Ousterhout. 2014. In search of an understandable Consensus Algorithm. In *Proceedings of ATC*.

[11] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter Rising: A decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proceedings of SIGCOMM*.

[12] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet transactions: High-level programming for line-rate switches. In *Proceedings of SIGCOMM*.

[13] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. In *Proceedings of HotCloud*.

[14] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Proceedings of IMC*.

[15] Yang Zhang, Bo Han, Zhi-Li Zhang, and Vijay Gopalakrishnan. 2017. Network-assisted Raft Consensus Algorithm. In *Proceedings of SIGCOMM Posters and Demos*.