Tracking the QUIC Spin Bit on Tofino

Ike Kunze
kunze@comsys.rwth-aachen.de
RWTH Aachen University
Germany

Constantin Sander
sander@comsys.rwth-aachen.de
RWTH Aachen University
Germany

Klaus Wehrle
wehrle@comsys.rwth-aachen.de
RWTH Aachen University
Germany

Jan Rüth
rueth@comsys.rwth-aachen.de
RWTH Aachen University
Germany

ABSTRACT

QUIC offers security and privacy for modern web traffic by closely integrating encryption into its transport functionality. In this process, it hides transport layer information often used for network monitoring, thus obsoleting traditional measurement concepts. To still enable passive RTT estimations, QUIC introduces a dedicated measurement bit – the *spin bit*. While simple in its design, tracking the spin bit at line-rate can become challenging for software-based solutions. Dedicated hardware trackers are also unsuitable as the spin bit is not invariant and can change in the future.

Thus, this paper investigates whether P4-programmable hardware, such as the Intel Tofino, can effectively track the spin bit at line-rate. We find that the core functionality of the spin bit can be realized easily, and our prototype has an accuracy close to software-based trackers. Our prototype further protects against faulty measurements caused by reordering and prepares the data according to the needs of network operators, e.g., by classifying samples into pre-defined RTT classes. Still, distinct concepts in QUIC, such as its connection ID, are challenging with current hardware capabilities.

CCS CONCEPTS

• **Networks → Network monitoring**; **Network measurement**; **Transport protocols**.

KEYWORDS

In-Network Telemetry, RTT measurements, QUIC Spin Bit, P4

ACM Reference Format:
Ike Kunze, Constantin Sander, Klaus Wehrle, and Jan Rüth. 2021. Tracking the QUIC Spin Bit on Tofino. In *Workshop on the Evolution, Performance and Interoperability of QUIC (EPIQ'21), December 7, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3488660.3493804

1 INTRODUCTION

Network monitoring is essential for managing modern networks [17], and current approaches range from embedding measurement data into production traffic [3, 11, 14, 24, 26] to passively observing the traffic [8]. The latter option is generally seen as the go-to choice for large-scale access networks as it entails a low overhead. Corresponding techniques typically exploit the wire image [25] of production traffic to derive network KPIs providing further insights, e.g., w.r.t. SLAs, abnormal latencies, or required topology changes.

To cope with ever-increasing network sizes and speeds, network monitoring and telemetry tasks are increasingly performed directly on the data plane of programmable hardware switches [13, 18, 22]. However, these platforms come with specific constraints that render implementations not as straightforward as conventional software solutions [20]. Still, related work has recently shown that traditional TCP RTT estimations leveraging sequence numbers (SEQs) and ACKs [4, 11, 25] can be efficiently realized on such devices [6], thus enabling measurements at line-rate without deploying dedicated measurement hardware or the need for challenging software-based processing of large quantities of packets.

In contrast to TCP, QUIC has a drastically reduced wire image as it encrypts its signaling information, including its notion of SEQs and ACKs, which prevents conventional SEQ/ACK-based measurements. Recognizing this challenge, the IETF standardized a dedicated measuring bit within QUIC – the *spin bit* [16] – which is exempted from encryption and allows a passive observer to estimate RTTs. Tracking the spin bit is relatively easy in software, e.g., using Spindump [10], which can further compensate for errors caused by reordering [9]. However, as with TCP, ever-increasing data rates make hardware implementations necessary, but purpose-built hardware comes with additional cost and the spin bit is an optional, non version-independent QUIC feature, i.e., hardware investment now may soon prove to be wasted should future QUIC versions modify the spin bit. Thus, re-programmable hardware seems like a promising solution to implement the conceptually simple spin bit mechanism.

This paper thus explores how the spin bit mechanism maps to a commodity programmable switch, the Intel Tofino. We identify several challenges raised by the combined characteristics of programmable data planes, QUIC, and the spin bit and discuss and present potential solutions. Specifically, we start with a naïve spin bit tracking mechanism that we complement with several additional processing steps. In particular, we test two mechanisms for protecting against reordered spin bit flanks and explore different means to export the measurements from the data plane: *periodically* and *event-driven*. We further approximate RTT averages and show how to utilize operator-defined RTT classes to classify measurements already on the data plane. Our evaluations compare the accuracy of RTT estimations by our P4-based spin bit tracker [19]

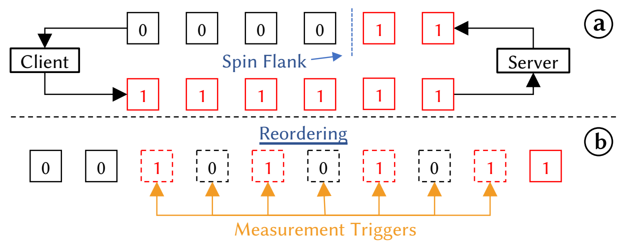


Figure 1: ⓐ: The client of a connection always *flips* the spin bit; the server *reflects* it. ⓑ: Reordering near the spin signal flanks can cause faulty measurements.

to ground-truth estimates provided by Spindump, revealing that a pure spin bit implementation is possible on Tofino. Similarly, our more evolved concepts prove to be effective, although at the cost of higher resource consumption.

Structure. In Sec. 2, we first concisely introduce fundamental aspects of programmable networking hardware. Sec. 3 then presents the design and implementation of our P4-based spin bit tracker, whose performance we evaluate in Sec. 4. Finally, we discuss related work (Sec. 5) and limitations (Sec. 6) of our tracker before concluding the paper.

2 P4-PROGRAMMABLE HARDWARE

The current generation of programmable networking hardware is usually split into an ASIC-based data plane that processes packets at line-rate and a slower CPU-based control plane that configures and manages said data plane. The data plane follows a pipeline-based model, which allows for a certain degree of programmability, e.g., using P4 [5].

Enclosed by programmable parsers and de-parsers that first extract packet information and later re-assemble the packets, the heart of the pipeline consists of a fixed number of configurable stages, each containing so-called Match-Action Units (MAUs) to which all desired computations are mapped. More specifically, the MAUs define *matching* logic on SRAM or TCAM to perform key-based lookups while the *action* part allows for a small number of simple arithmetic and logical operations. One current state-of-the-art programmable switch, the Intel Tofino, e.g., enables multiplications involving powers of two with statically defined multiplicands.

So-called *registers* are memory constructs that can be read and written from within the data plane. However, due to the pipeline model, these registers and all other memory can only be accessed within the associated MAUs, i.e., once in the pipeline (*single-access rule*), which considerably limits read-update-write concepts on state variables. Tofino enables read-update-write functionality in the form of small microprograms. Overall, the nature of these pipelines that are designed for very high-speed processing of packets with latencies that are both low and fixed makes programming, in comparison to much more flexible but slow CPU-based processing, more difficult. In the following, we present how we still implement a basic spin bit tracking mechanism as well as additional heuristics on Tofino.

3 TRACKING THE SPIN BIT ON TOFINO

The spin bit is QUIC's answer to the call for explicit measurement support by Allman et al. [1] and exposes a measurable square wave signal onto a QUIC flow enabling RTT measurements. For this, the client always *flips* the spin bit it received while the server *reflects* it, as shown in Fig. 1 ⓐ. An observer can then determine the flow's RTT by measuring the time between consecutive spin bit flips.

Based on this simple mechanism, we have implemented a P4-based spin bit tracker [19] in the Ingress of an Intel Tofino switch. Fig. 2 illustrates our overall pipeline, which centers around the core spin bit tracking (②) and RTT calculation (③) mechanisms. In particular, we first have to *identify a flow* as the spin bit is always set on a per-flow basis (①). We then study three variants to detect a spin phase-change, mainly owed to the spin bit's susceptibility to reordering (②). After computing the RTT (③), we investigate how additional post-processing concepts found in software-based observers, such as averaging (④) or range-based filtering (⑤), can be realized on programmable switches to reduce the amount of information that needs to be communicated to the network operator. Finally, we explore different ways of extracting the measured RTTs from the data plane (⑥).

In the following, we describe the design of the different components and the challenges we faced when realizing them, starting with the flow identification.

3.1 Flow Identification – ①

The spin bit is a per-flow signal, i.e., we must be able to differentiate between flows. QUIC offers variable-length connection IDs (CIDs) that identify flows even when IPs change (e.g., on NAT rebinding). The CID is negotiated in the handshake using long header packets that denote the sender's and receiver's CID, each preceded by its respective length [16].

While the CID is not encrypted, using it for identifying flows on a switch is still challenging. We first require some mapping to CID length to extract the CID from short header packets (containing the spin bit) as this header, unfortunately, lacks a length field [16]. This mapping can either be manually configured or extracted from long header packets during the handshake. Since a server may change the initially client-chosen destination CID, one needs to update subsequently observed CID length fields. We prototyped the CID length extraction from long header packets to show its feasibility but are currently not using it and instead set up the mapping manually. However, having two variable-length fields (20 byte max. length.) is too taxing on the Tofino's deparser, yet, representing each possible length by one specific field does work, although this has a heavy cost on the deparser, too.

For the actual CID extraction in short header packets, we can speculatively parse 20 bytes of CID. We then match on the five-tuple[1] to find the length of the CID. If the length is non-zero, we extract the corresponding subset from the 20 bytes as the actual CID. The CID (or the five-tuple for zero-length CIDs) is then hashed as a flow ID that identifies our spin bit measurements. While we did implement this on Tofino, the sequential dependencies, together with our spin bit measurements and post-processing steps, in sum,

[1]Depending on mobility or NAT rebinding at which side, just one endpoint's IP and port may be more suitable.

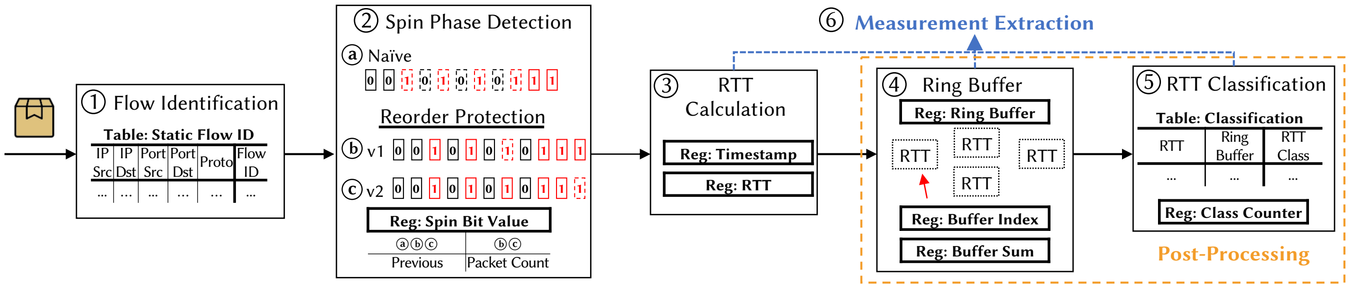


Figure 2: A five-step spin bit tracking pipeline. After an initial flow identification (①) and a spin phase detection (②), our pipeline calculates the current RTT (③). It further keeps track of the last N measurements in a ring buffer (④) and can classify the current RTT based on those last measurements (⑤).

exhaust the available stages. Our prototype currently focuses on the measurement-related capabilities and maps flows only by five-tuple as depicted in Fig. 2–*Table: Static Flow ID*.

Next, after identifying a flow, the spin bit tracker has to correctly detect spin bit phase transitions.

3.2 Spin Phase Detection – ②

The spin bit is known to be susceptible to packet reordering [9] as out-of-order packets near the flanks of the square wave signal will cause invalid RTT readings. For example, in Fig. 1 ⓑ, each packet within the affected spin flank (dotted packets) produces faulty estimates. Simply triggering a new measurement once the spin bit changes as done in our naïve variant (ⓐ) might thus lead to invalid results in times of reordering. While reorder protections can already detect and discard many erroneous measurements [9], they rely on performing the measurement first, then applying heuristics, and finally rejecting invalid measurements. Such concepts are not directly possible in the data plane due to the single-access rule. While workarounds using packet recirculation are possible, they add non-negligible overhead. We thus experiment with two approaches to actively protect against reordering. They are inspired by the reorder protection proposed for the square bit, an additional measurement extension that enables packet loss measurements [7].

Reorder Protection v1 – ⓑ. Our first approach directly follows the protection of the square bit. A phase transition, and thus a measurement, is only triggered as soon as N packets of the new flank have been registered, e.g., after the third 1-bit in Fig. 2 for $N = 3$. However, in scenarios with significant reordering around a flank or if the spin bit is greased, i.e., set to arbitrary values when the spin bit functionality is not enabled, this simple form of protection might still yield many invalid measurements.

Reorder Protection v2 – ⓒ. Our second approach tries to resolve these shortcomings by requiring the same number of packets but in direct succession, putting a stronger requirement on the phase-change. Depending on the threshold value, this scheme should filter greased spin bits and also most invalid measurements in case of actual reordering.

Implementation. We implement each of the three phase-change detection approaches using a single register that tracks the current value of the spin bit (Fig. 2–*Reg:Spin Bit Value*). However, the reorder protection variants require more memory as they do not only track the spin bit (left) but also the number of seen packets (right). In our prototype, we deploy all three concepts in parallel to allow for easy switching between the different modes. Upon detecting a phase-change, the subsequent RTT calculation is triggered.

3.3 RTT Calculation – ③

We calculate the RTT using two registers, one tracking the timestamp of the previous spin bit flip (Fig. 2–*Reg: Timestamp*), the other storing the calculated RTT (Fig. 2–*Reg: RTT*). For measuring time, Tofino provides a 48-bit timestamp with nanosecond resolution. For simplicity and to save resources, our current spin bit tracker prototype uses a well-chosen 16-bit slice of this switch-local timestamp to derive a timestamp with near-millisecond resolution (one timestep $\approx$ 1.049 ms) that wraps around after roughly 69 seconds; we plan to evaluate the suitability of the chosen slice in future work (see Sec. 6). Using the previously described mechanism, this core component of the spin bit measurement functionality nicely maps to Tofino. In the following, we show how these simple measurements can be leveraged for additional post-processing steps, such as determining average RTTs or classifying measurements.

3.4 Post-Processing

Network operators might not be interested in the exact RTTs of specific flows but rather in a rough distribution of the occurring RTTs or whether the RTT of flows fluctuates. Consequently, there is potential in early post-processing on the data plane to reduce the amount of information that needs to be communicated between the data and the control plane.

RTT Averaging (Ring Buffer) – ④. For classification tasks, it is essential to maintain statistics on the already seen RTT values. For instance, software-based RTT tracking tools, such as Spindump, use a moving average of the current values. However, such techniques are not easy to implement on the data plane, as, e.g., required divisions are typically not supported. To still capture some form of average RTT, we implement a simple ring buffer on the data plane that stores the last N RTT values of a flow (Fig. 2–*Reg: Ring Buffer*). We further store the sum over all ring buffer entries for a particular flow in another register and update it whenever we exchange entries of the ring buffer (Fig. 2–*Reg: Buffer Sum*). A third register tracks the read/write position of the ring buffer (Fig. 2–*Reg: Buffer Index*). Knowing N, the control plane can calculate the

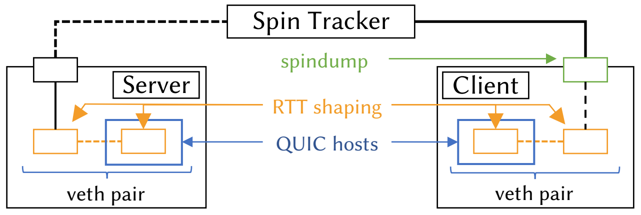


Figure 3: One Tofino switch running our spin tracker interconnects two QUIC end-hosts.

mean RTT of a flow without having to react on every new result or to keep state.

RTT classification – ⑤. While explicit information on the current RTTs is important, we believe that network operators are more interested in the general traffic characteristics and the stability of their network's performance. Consequently, we propose deploying a simple form of RTT classification already in the data plane that builds upon our ring buffer solution. In our concept, the control plane can configure different RTT ranges, and the data plane then accurately tracks which of these ranges are hit by the flows currently under study. This information could, e.g., be used to create RTT histograms. Our prototype uses a simple range setting with three classes, one designed to capture greased spin bits and reorderings (< 5ms), one intended to capture stable measurements ($+/-10\%$ *meanRTT*), and one containing the regions in between and above, thus capturing unstable measurements. We leave finding an optimal configuration for these RTT classes to future work. The RTT classification deploys one MAU using range matches of the current RTT and the ring buffer RTT to classify the current RTT (Fig. 2–*Table: Classification*). An additional register then tracks how often each of the classes has been hit (Fig. 2–*Reg: Class Counter*).

3.5 Measurement Report Intervals – ⑥

In contrast to TCP measurements, there is at most one measurement per RTT for the spin bit. Thus, there are different options for notifying the control plane of new measurements.

One option is leveraging the switch-local control plane to read out the data plane registers in fixed intervals (*periodic*). While this allows for well-controlled read intervals, it might be challenging to find the best readout interval for each situation that ensures a truthful representation of the signal.

As a more dynamic alternative, we let the data plane send out one dedicated packet containing the latest measurement results whenever a new spin cycle has been detected (*event-based*). For this, we clone the packet that triggered the measurement and then modify the clone to contain the RTT estimations before redirecting it to the control plane. This approach is more robust to changing flow dynamics yet requires additional data-to-control plane bandwidth.

4 EVALUATION

We evaluate the accuracy and capabilities of our prototype in a testbed where two end-hosts are interconnected by an Intel Tofino switch running the spin bit tracker, as illustrated in Fig. 3. We shape delay using *tc netem* on the interconnecting links of the veth pairs

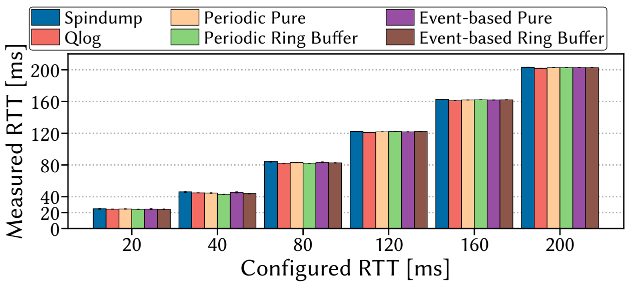


Figure 4: Measured mean RTTs for different configured RTTs and a download of 2 MB.

on each end-host and configure a steady bandwidth of 10 Mbps[2] on the client's ingress interface. For measuring RTTs, we create QUIC traffic using aioquic [21] at the QUIC hosts (blue) and capture the output of our tracker, the qlog [23] information of aioquic, and the traffic at the client's ingress interface, which we subsequently analyze using Spindump [10]. While the Spindump results provide a groundtruth regarding the accuracy of plain spin bit measurements, the qlog data serves as a groundtruth for the actual RTT. To accurately compare the timestamps of the different involved entities, we use PTP [15] to synchronize the clocks before each measurement run. For each setting, we perform 30 measurement runs and report mean values with 99% confidence intervals.

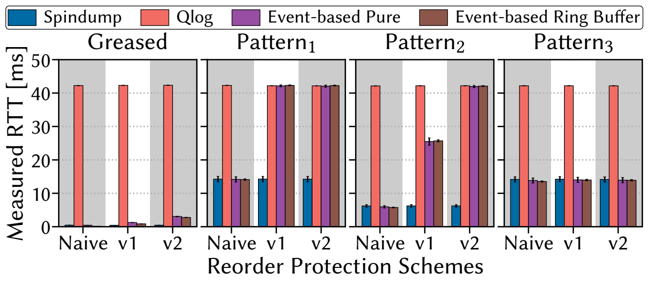
4.1 Web Traffic Performance

We first investigate how well our spin tracker can track the RTT of standard web traffic. For this, we configure different RTTs and let our client request different-sized files using HTTP/3 from the server. Fig. 4 shows the measured mean RTT across all measurements when downloading a 2 MB file.

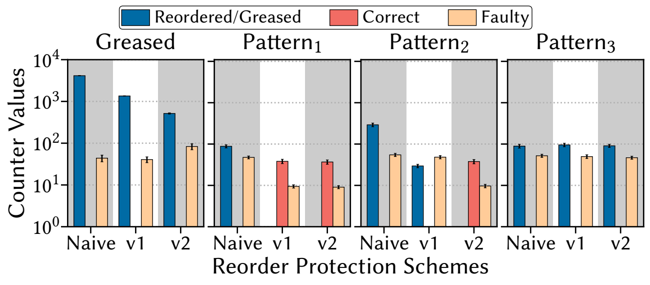
The individual (*pure*) readings of the *periodic* and *event-based* solutions are very close to the *Qlog* and *Spindump* estimates, showing the general feasibility of performing spin bit measurements on the data plane. The *ring buffer* shows steady results, proving that RTT approximations using a ring buffer on the data plane with subsequent average calculations on the control plane are feasible.

However, when experimenting with different bottleneck queue sizes and bandwidths (not shown), we found that the oversampling of our interval-based variant (5 ms readout interval) can lead to an RTT underestimation when small queues build up. In particular, the induced queuing delay is only eventually picked up by the spin bit, while the interval-based readout oversamples readings in the transition periods, thus causing an underestimation of the RTT. Consequently, selecting suitable readout intervals, especially for multi-flow settings with different latencies, is essential and non-trivial as too high intervals will cause measurements to be missed. At the same time, too short intervals might oversample the available measurements and cause an untruthful representation of the RTTs.

[2]This choice is sufficient for our evaluation as our prototype does not use recirculation. The measurements thus scale with the RTT and not with the bandwidth.



(a) Measured mean RTTs



(b) Mean RTT classification counter values

Figure 5: Results for four spin bit patterns, an RTT of 40 ms, and a download of 10 MByte.

4.2 Reorder Protection & RTT Classification

We next study how we can use the reorder protection mechanisms and the RTT classification to account for greased spin bits and packet reordering occurring at the spin bit flanks, both causing invalid measurements. To focus our evaluation, we artificially create four distinct traffic patterns targeting a reordering threshold of three packets. We leave investigating an optimal sizing of the reordering threshold for future work.

Spin Bit Patterns. *Greased* sets the spin bit randomly to create a random spin bit sequence. In contrast, three reordering patterns produce fixed sequences. *Pattern*$_1$ inserts one duplicate packet of the old flank after an actual spin flank which causes two invalid readings in short succession. *Pattern*$_2$ repeats this process for the first three packets of the new flank, thus triggering six invalid measurements to circumvent reorder protection v1. Finally, *Pattern*$_3$ transmits a sequence of three packets of the old flank as soon as there were three packets of the new flank, thus disabling reorder protection v2. The first two patterns represent random reordering, while the latter could, e.g., be caused by faulty load balancing.

Results. Fig. 5 shows (a) the measured mean RTT of our event-based approach (*pure* and *ring buffer*) in comparison to end-host estimates (*Qlog*) and *Spindump*, as well as (b) the mean counter values of our RTT classification tables, denoting how often a class was hit, in a setting with an RTT of 40 ms and a download size of 10 MB when inducing the four spin bit patterns.

The end-hosts automatically filter the reordering and produce realistic latency estimations in all settings. In contrast, when not using reorder protection, the Spindump ground truth and our spin tracker expectedly underestimate the actual RTT for all patterns. However, our RTT classification is able to classify these measurements as faulty, which can be seen in the high values of the corresponding counters (reordered/greased and faulty). No readings are classified as correct because this requires a stable RTT average which is not the case for these frequent reorderings.

Reorder protection v1 effectively protects against *Pattern*$_1$. It further yields more reasonable results for *Pattern*$_2$ compared to no protection as it filters many invalid edges and only produces one invalid reading for each spin flank. Consequently, its RTT estimates are roughly half of the real RTT.

Reorder protection v2 protects against *Pattern*$_1$ and *Pattern*$_2$ as it puts the tightest requirements on a flank. Most measurements are classified as *stable*, mixed with a few faulty readings, which can be explained by the queue build-up already noted in Sec. 4.1. Yet, v2 does not cope with *Pattern*$_3$, i.e., sequences of reordered packets exceeding its threshold.

Finally, none of our schemes fully protect against greasing, although they filter many invalid readings and detect all faulty measurements accordingly.

Takeaway. *Our results show that it is possible to i) filter invalid edges from the outset or ii) at least identify invalid/unstable measurements already on the data plane, thus showing potential for post-processing according to the needs of network operators. Depending on the expected level of reordering, our proposed protection schemes can be configured with different thresholds for tailored protection. Similarly, the RTT classes are configurable to the expected latency ranges and offer a very focused view of the network characteristics.*

To achieve optimal cost-efficiency, our approach should be deployed on hardware already used for standard networking functionality. We thus finally analyze the resource use of our prototype.

4.3 Resource Footprint

To judge the practicability of our approach, we finally dissect the relative resource use of the different components of our prototype in a configuration that is able to track up to 10 000 flows with a ring buffer size of 4 and 8 different RTT classes.

As can be seen in Table 1, a **base** implementation of our pipeline (Steps ① – ③ⓐ) together with a simple L2/L3 forwarding logic is very light on resources and leaves plenty of room for other functionality. The reorder protection (③ⓑ/ⓒ) also comes with minor additional resource use, even when simultaneously adding both variants. In contrast, the ring buffer mechanism (④) requires more resources and additional pipeline stages. Still, even when deploying the RTT classification (⑤) or all components simultaneously (**Full**), the overall resource use is still modest, with ALUs being the most used component with 13.5% of the available resources. While our full spin bit tracker requires resources at all available stages due to sequential dependencies, its modest resource use shows the practicability of deploying a spin bit tracker on programmable networking hardware alongside other networking functionality. However, adding a CID-based flow identification overstretches the sequential capabilities, illustrating that the available resources can make prototypes fail even though they are conceptually possible.

Table 1: Resource footprint of different compositions of our spin tracker in % of the available resources.

Resource [%]	Base	+③ⓑ/ⓒ	+④	+④⑤	Full
Stages	41.7	41.7	58.3	100	100
SRAM	3.4	4.2	5.2	6.0	6.7
TCAM	0.0	0.0	0.0	3.5	3.5
Hash Bits	4.6	5.2	5.8	6.6	6.8
VLIW Actions	2.3	3.1	3.6	5.2	6.0
Match X-bars	2.4	2.5	3.2	3.8	3.7
ALUs	6.23	8.3	10.4	11.5	13.5

5 RELATED WORK

Tracking flow latencies on the data plane has already been studied for TCP. Ghasemi et al. [12] propose Dapper to track various TCP metrics, including the RTT, which they measure using the SEQ/ACK methodology. In contrast, Chen et al. [6] completely focus on RTT measurements and show that these can be conducted using a Tofino switch. They first store timestamps of outgoing packets in a multi-stage hash table and then match those to timestamps of incoming packets, cross-relating the packets using hashes of the five-tuple. Apostolaki et al. [2] perform RTT measurements based on the TCP handshake leveraging SYN/SYNACK packets which they map using a Counting Bloom Filter. A similar methodology could also be applied to measure the RTT of QUIC's handshake.

6 DISCUSSION

Flow Identification. While QUIC has an explicit CID, its lack of a length indicator in short header packets complicates a CID-based flow tracking as the connection establishment must be observed. Even though possible to implement, it i) involves many sequential steps that challenge a placement with further functionality on a fixed number of stages, and ii), depending on what scenarios should be covered, different keys to look up the CID lengths may be beneficial. In addition, it is up for discussion whether using CIDs is beneficial or whether the five-tuple approach is sufficient.

Flow Selection. Our prototype only measures flows that are explicitly added to a selection list. More dynamic approaches are of interest for network operators, e.g., for dynamically suggesting flows to track. Even though dynamically tracking flows is beyond the scope of this work, we prototyped a long header CID extraction to validate the feasibility.

Measurement Direction. We currently only perform one-directional measurements, i.e., track a flow passing in both directions as two separate flows. The main reason for this choice is that we cannot guarantee that we can track a bi-directional flow in all cases as one pipe of the Tofino (similar to other switches) only manages a subset of all ports. If the ingress and egress ports of a bi-directional flow are on different pipes, the registers holding the flow information are not shared between those pipes, and thus, the information is inaccessible between the two. Nevertheless, the control plane can map these flows.

Time Resolution. As already discussed in Sec. 3.3, our implementation uses a fixed 16-bit slice of the available 48-bit nanosecond resolution timestamp. The lowest time interval supported by our implementation is ≈ 1.049 ms, and there is a rollover roughly every 69 seconds. Our current prototype can only resolve one rollover between consecutive spin bit flips for which we invest additional hardware resources, i.e., we duplicate the *Spin Bit Value Register* and *Ring Buffer Register*. Given typical communication intervals, we believe a rollover of 69 seconds to be a feasible solution. However, given that Chen et al. [6] use full nanosecond resolution, we plan to evaluate if the selected 16-bit slice is sufficient.

Reordering Threshold. Our current prototype uses a reordering threshold of three. While configurable, choosing a value is not straightforward as there might be, e.g., flows holding less than three packets per RTT. Thus, similar to the square bit [7], it is an open question how to configure the threshold in practice.

Future Work. Our evaluation focusses on the general feasibility of our prototype, i.e., showing that pure spin bit measurements as well as additional post-processing and reorder protection can be implemented on Tofino. However, the measurement performance depends on several set screws which might further need tuning to the expected network settings. We thus plan to perform an additional thorough evaluation of our prototype, e.g., inspecting the impact of different reordering thresholds, ring buffer sizes, or measurement read-out intervals, as well as different network conditions.

7 CONCLUSION

The IETF's decision to encrypt almost all of QUIC's signaling information, including its notion of SEQs and ACKs, challenges the established network monitoring world as conventional solutions for tracking flow RTTs are no longer applicable. While the spin bit addresses this challenge and allows for RTT measurements, it is susceptible to reorderings, which have to be filtered using additional methods of spin bit tracking software. This, however, goes against the recent trend of increasingly performing networking monitoring tasks directly on programmable networking hardware.

This paper shows that the spin bit does not stand against this trend as it can be effectively tracked on an Intel Tofino switch. Furthermore, simple mechanisms to protect against reordering prove to be quite effective and also map to the programmable data plane, as does additional post-processing of the measurements in the form of simple averaging and RTT-based classification.

However, some features of QUIC, such as not having a connection ID length indicator in short header packets, significantly complicate data plane implementations on the current generation of programmable hardware.

ACKNOWLEDGMENTS

This work has been funded by the German Research Foundation DFG under Grant No. WE 2935/20-1 (LEGATO). We thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Mark Allman, Robert Beverly, and Brian Trammell. 2017. Principles for Measurability in Protocol Design. *ACM SIGCOMM Computer Communication Review* 47, 2 (2017). https://doi.org/10.1145/3089262.3089264

[2] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. 2021. Performance Driven Internet Path Selection. In *ACM SIGCOMM Symposium on SDN Research (SOSR '21)*. https://doi.org/10.1145/3482898.3483357

[3] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic In-Band Network Telemetry. In *ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. https://doi.org/10.1145/3387514.3405894

[4] Peter Benko and Andras Veres. 2002. A Passive Method for Estimating End-to-End TCP Packet Loss. In *IEEE Global Telecommunications Conference (GLOBECOM '02)*. https://doi.org/10.1109/GLOCOM.2002.1189102

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95. https://doi.org/10.1145/2656877.2656890

[6] Xiaoqi Chen, Hyojoon Kim, Javed M Aman, Willie Chang, Mack Lee, and Jennifer Rexford. 2020. Measuring TCP Round-Trip Time in the Data Plane. In *Workshop on Secure Programmable Network Infrastructure (SPIN '20)*. 35–41. https://doi.org/10.1145/3405669.3405823

[7] Mauro Cociglio, Alexandre Ferrieux, Giuseppe Fiocolla, Igor Lubashev, Fabio Bulgarella, Isabelle Hamchaoui, Massimo Nilo, Riccardo Sisto, and Dmitri Tikhonov. 2021. *Explicit Flow Measurements Techniques*. Internet-Draft. IETF. https://datatracker.ietf.org/doc/draft-mdt-ippm-explicit-flow-measurements/ Work in Progress.

[8] Mauro Cociglio, Giuseppe Fioccola, Guido Marchetto, Amedeo Sapio, and Riccardo Sisto. 2019. Multipoint Passive Monitoring in Packet Networks. *IEEE/ACM Transactions on Networking (TON '19)* 27, 6 (2019). https://doi.org/10.1109/TNET.2019.2950157

[9] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. 2018. Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP. In *ACM Internet Measurement Conference (IMC '18)*. https://doi.org/10.1145/3278532.3278535

[10] Ericsson. 2021. Spindump on GitHub. https://github.com/EricssonResearch/spindump

[11] Giuseppe Fioccola, Alessandro Capello, Mauro Cociglio, Luca Castaldelli, Mach(Guoyi) Chen, Lianshu Zheng, Greg Mirsky, and Tal Mizrahi. 2018. *Alternate-Marking Method for Passive and Hybrid Performance Monitoring*. RFC 8321. IETF. http://tools.ietf.org/rfc/rfc8321.txt

[12] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *ACM SIGCOMM Symposium on SDN Research (SOSR '17)*. 61–74. https://doi.org/10.1145/3050220.3050228

[13] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. 2021. A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. https://arxiv.org/abs/2101.10632 arXiv preprint.

[14] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. OmniMon: Re-Architecting Network Telemetry with Resource Efficiency and Full Accuracy. In *ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. 404–421. https://doi.org/10.1145/3387514.3405877

[15] IEEE 1588-2019 2019. *IEEE 1588-2019 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. Standard. IEEE.

[16] Jana Iyengar and Martin Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. IETF. http://tools.ietf.org/rfc/rfc9000.txt

[17] Juniper Networks. 2017. *The Self-Driving Network*. Technical Report. https://www.juniper.net/assets/us/en/local/pdf/pov/3200053-en.pdf

[18] Elie F Kfoury, Jorge Crichigno, and Elias Bou-Harb. 2021. An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends. *IEEE Access* 9 (2021), 87094–87155. https://doi.org/10.1109/ACCESS.2021.3086704

[19] Ike Kunze. 2021. Tofino Spin Bit Tracker on GitHub. https://github.com/COMSYS/tofino-spin-bit-tracker

[20] Ike Kunze, Moritz Gunz, David Saam, Klaus Wehrle, and Jan Rüth. 2021. Tofino + P4: A Strong Compound for AQM on High-Speed Networks?. In *International Symposium on Integrated Network Management (IM '21)*. IFIP/IEEE.

[21] Jeremy Lainé. 2021. aioquic on GitHub. https://github.com/aiortc/aioquic

[22] Pilar Manzanares-Lopez, Juan Pedro Muñoz-Gea, and Josemaria Malgosa-Sanahuja. 2021. Passive In-Band Network Telemetry Systems: The Potential of Programmable Data Plane on Network-Wide Telemetry. *IEEE Access* 9 (2021), 20391–20409. https://doi.org/10.1109/ACCESS.2021.3055462

[23] Robin Marx, Maxime Piraux, Peter Quax, and Wim Lamotte. 2020. Debugging QUIC and HTTP/3 with qlog and qvis. In *Applied Networking Research Workshop (ANRW '20)*. 58–66. https://doi.org/10.1145/3404868.3406663

[24] The P4.org Applications Working Group. 2020. Telemetry Report Format Specification. https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_latest.pdf.

[25] Brian Trammell and Mirja Kuehlewind. 2019. *The Wire Image of a Network Protocol*. RFC 8546. IETF. http://tools.ietf.org/rfc/rfc8546.txt

[26] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. Flow Event Telemetry on Programmable Data Plane. In *ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. 76–89. https://doi.org/10.1145/3387514.3406214