Direct Telemetry Access

Authors

1. ABSTRACT

The emergence of programmable switches makes it possible to collect a large amount of fine-grained telemetry data in real time in large networks. However, transferring the data to a telemetry collector and relying on the collector CPUs to process telemetry data is highly inefficient. RDMA is commonly used as an efficient data transfer protocol but it does not work for telemetry data. This is because it is challenging to create RDMA format packets at switches, support lossy networks, and support various memory access patterns such as multiple concurrent writes. In this paper, we introduce a new direct telemetry data access system, which allows fast and efficient telemetry data transfers between switches and the remote collector. Our system introduce telemetry report primitives such as key-write, append, aggregation, and multi-increment. To support these primitives and address the limitations of RDMA, we use some programmable switches as translators that generate RDMA packets, handle packet losses, and aggregate diverse memory accesses. Our evaluation demonstrates that we can enhance existing telemetry framework by XX%.

2. INTRODUCTION

Telemetry is essential.

Telemetry is costly.

RDMA has huge potential.

RDMA limits us. (e.g., DART)

We want to expand on RDMA.

Current solutions for expanded RDMA either limits performance (two-way or CPU-based SoC), or requires significant investments (e.g., FPGA nics).

DTA significantly reduces in-ASIC costs, network BW, and improves stability compared with RDMA.

DTA *also* allows more powerful primitives, without breaking support for commodity RDMA NICs! (we are pretty cool).

3. MOTIVATION

Collectors play an important role in network telemetry systems: they receive telemetry reports and store the information in an internal data structure to be used to answer network-wide queries. One key challenge is to ensure that this process is scalable as a datacenter network can comprise hundreds of thousands of switches [4]. For example, a non-sampled INT telemetry system requires the collection of telemetry data from *every single packet*, which would result in an excessive amount of reports. Because of this, event detection is typically implemented at switches in an effort to send reports to a collector only when things change [9]. This helps in reducing the rate of switch-to-collector communication down to a few million telemetry reports per second per switch [19], at the cost of reduced network insight and increased on-switch complexity. Still, telemetry collection costs are high, and the main reason we identified is that the collectors' CPU is the main bottleneck.

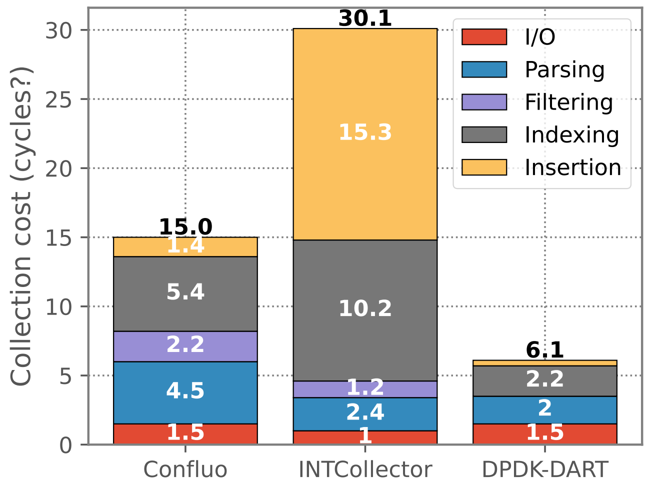

Figure 1: Breakdown of telemetry collection costs in CPU-based solutions. DPDK-DART is functionally equivalent to DTA Multi-Write (This is placeholder data!)

3.1 CPU-based collection is inefficient

MY: key bottleneck or the ignored bottleneck in many telemetry systems today

MY: Discuss existing monitoring solutions

We show examples of what the CPU spends time on in Figure 1, and discuss how they already attempt to optimize these numbers. DTA would either bypass or offload these

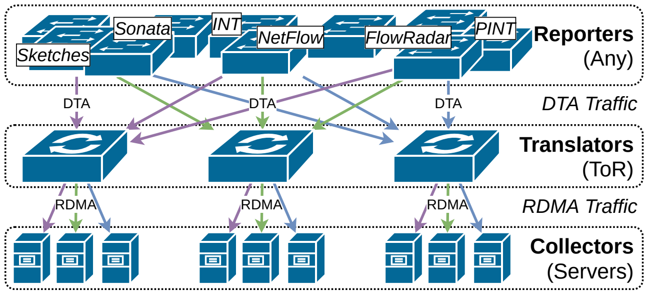

Figure 2: Overview of the DTA report data flowJL: todo:flip upside down

steps into hardware.

3.2 RDMA enables high-speed memory writes

What if we design telemetry collection around RDMA?

3.3 RDMA needs a redesign for telemetry

Generation is costly. Basically hogging entire PHV. Requires unnecessary statefulness for supporting legacy RDMA.

Packet loss kills the performance until resync. Resync support requires even more pipeline logic.

Limited primitives. Only READ/WRITE enables full-speed RDMA. Network-wide collection algorithms are severely limited.

4. BACKGROUND?

JL: Do we need this? Handled by Intro and Motivation?

5. DESIGN

DTA is leveraging programmable switches to expand on RDMA with new and powerful functionality, while significantly reducing the on-switch footprint compared to direct RDMA-generating collection alternatives. This is achieved by wrapping a DTA interface around RDMA, where RDMA calls are triggered by DTA packets, which are much cheaper to generate in terms of in-switch resources.

5.1 Overview

The DTA reporting flow consists of switches with two different roles: *reporters* and *translators*, according to Figure 2. Reporters are generating the actual telemetry data, and can be deployed on any switch in the network. These switches report their local telemetry data to central collection through DTA report-packets. These packets contain telemetry information to be reported, and are sent towards one of several collection servers.

The last hop before reaching the collector, the ToR switch, acts as a DTA *translator*. Translators are responsible for ensuring that information carried by these DTA packets is written to collector memory in an efficient and queryable way. In-translator algorithms convert between these DTA packets and RDMA, which uses CPU-bypassing techniques to perform memory operations on the adjacent collector server(s) (see Section 6, discussing DTA primitives).

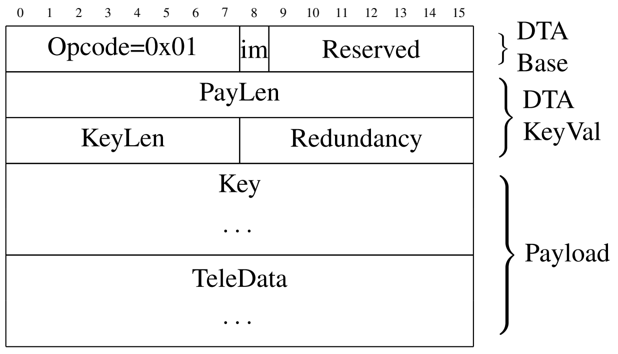

Figure 3: Header structure of DTA Key-Write reportsJL: check hpcc fig 7

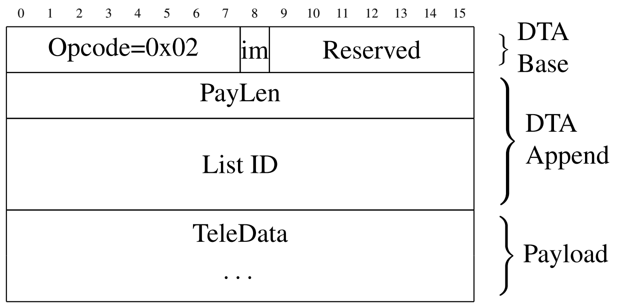

Figure 4: Header structure of DTA Append reports

5.1.1 Translator vs SmartNIC

JL: Discuss pros-and-cons of using a translator vs SmartNICs to process DTA. JL: Highlight smartnic cons? We design assuming we don't do SmartNICs

5.2 DTA Packet Structure JL: move to implementation?

Explain the headers. Base, KeyWrite, Append(when designed). IP&UDP encap

5.3 Proposed Primitives

For DTA to be truly generic, with broad and efficient support of various measuring techniques, it needs to support several different primitives.

5.3.1 Key-Write

Many telemetry scenarios can be expressed as key-value systems. One example of such a scenario is per-flow path tracing, where the key <*flow 5-tuple*> can store a list of integers to represent the switch IDs on the path. The *Key-Write* primitive is designed to support such telemetry scenarios, where pre-known keys can be used as identifiers for telemetry data of static-size.

5.3.2 Append

There are telemetry scenarios that can not easily be trans-

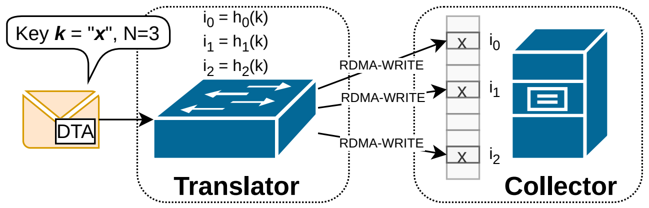

Figure 5: The DTA Key-Write primitive.JL: add querying

lated into equivalent key-value representations, either due to keys not being pre-know, or if the identifier (i.e., key) should be used to store several different values instead of just a single static entry. We introduce the *Append* primitive for cases where pre-defined global lists could be used to hold network-wide information. These lists would be statically defined to each keep homogeneous values of the same telemetry type. Examples could be *network loss events*, *high-congestion devices*, or *heavy hitters*. Switches would be pre-informed of which network-wide lists are available, and be able to append their local information into a fitting list for that value when they for example detect a period of high network loss.

5.3.3 *Sketch-Merge*

JL: Discuss design

5.3.4 *Key-Increment*

JL: Discuss design

6. DTA PRIMITIVESJL: MERGE DESIGN?

JL: Move implementation details into implementation. Keep this description high-level and from user perspective

DTA expands on RDMA with new primitives. These primitives allow for a significant reduction in collection overheads compared to equivalent RDMA solutions. These more complex primitives only require a hardware implementation, and therefore a footprint, on the DTA translator switches; reporter switches only have to specify their operation-code in the DTA packets together with the telemetry payload and potential key, fully offloading the RDMA generation to the translator(s).

The potential for triggering real-time reactivity in the collector CPU is supported by all primitives through the *immediate* flag in the DTA Base Header.

6.1 Key-Write

The DTA Key-Write primitive (Figure 5) is built to enable efficient key-value storage of telemetry data.

Dynamic keys. Shared global memory regions. Built-in aging

JL: Compare pros cons with reading server memory to avoid collision JL: footprints, suspending packet processing JL: Suspending packet processing might not be feasible, making READ solutions not work in reality. Check TEA solution (writing entire packets to server memory through DRAM. Scratchpat solution might not even work for our use-case. How make packet stored adjacent to the value?)

6.1.1 *Reporter Perspective*

We make no assumptions about the underlying measuring solution on the network switches, other than that they can map into a key-value representation of the telemetry data. Switches start by calculating a hash of the telemetry key, to find the collection server that shall hold information about this key. The switch generates a DTA packet according to Figure 3, which is IP-encapsulated to ensure routing towards the translator that is adjacent to the correct collector.

6.1.2 *Translation*

The translator will ingress the DTA packet, specifying the Key-Write primitive and relevant parameters. A number of RDMA-WRITE packet will be generated according to the specified level of redundancy (N). These N RDMA packets will write the same *TeleData* payload to different memory addresses, as calculated by a hash function $h(key, n)$, where each RDMA packet is based on its own n ranging from 0 to $N-1$; these are used to hash the telemetry data into N different memory locations, generating N-level redundancy. Introducing this N-level redundancy yields a more robust and memory-efficient data structure (see Figure 6), and significantly increases the time until data ages out of storage. The telemetry data is written to storage with a small concatenated checksum of the telemetry key, to be used to ensure correct query responses in case of data overwrites from other keys.

6.1.3 *Operator Perspective*

Per-key storage locations are entirely decided by global hash functions and lookup tables, enabling any network actor to compute the location of any potentially reported telemetry key. All N data redundancies are stored together on the same collector, enabling efficient access to redundancy information. A querying agent (e.g., an operator) can read all N redundancy slots, and use checksum-verification, and potentially plurality voting, to ensure value correctness.

6.1.4 *Key-Write Collection Rate*

Each incoming DTA Key-Write packet will generate several RDMA-WRITE packets equal to the packet-specified level of redundancy (N), with no additional use of internal busses[1]. This means that the DTA Key-Write primitive will deliver collection speeds at $\frac{1}{N}$ of the RDMA-capable NIC processing rate.

6.1.5 *Probability Theory*

Dig into level of redundancy, and return-error (checksumming and plurality). Figure showing impact of checksum lenght on return-error? Same with pluraily and level of re-

[1]We are assuming a Key-Write implementation based on multicasting, as is the case in our Tofino implementation

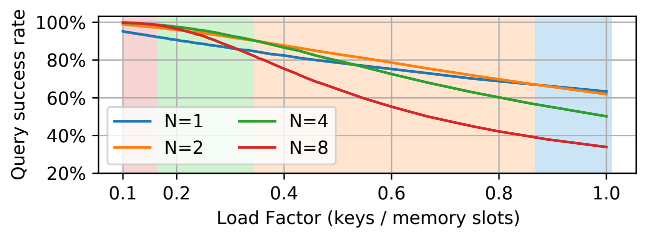

Figure 6: Average query success rates for DTA Key-Write data, depending on the collector load factor and the level of redundancy per key (N). The background color indicates optimal N in each interval.

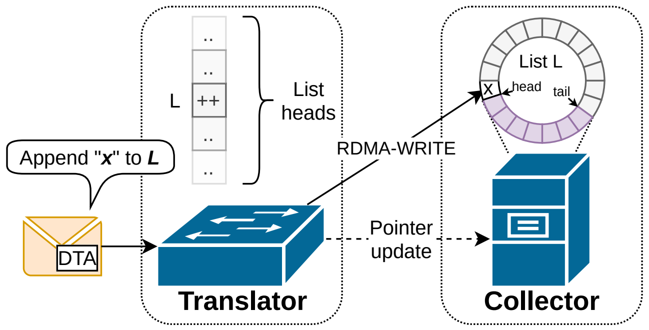

Figure 7: The DTA Append primitive.JL: add querying

dundancy? Add figure showing optimal N at various load factors (Figure 6)?

6.2 Append

DTA Append (Figure 7) is a powerful primitive that enables telemetry collection at RDMA line-rate, by requiring just a single RDMA-WRITE operation per-report. This is achieved through a ring-buffer design, managed by a collector-adjacent translator. However, as opposed to the DTA Key-Write, there is no support for dynamic key storage. Append is instead designed around statically-allocated centralized lists, where each list contains operator-specified telemetry information categories (e.g., loss events, congestions, or latency spikes).

6.2.1 *Reporter Perspective*

Network operators have pre-allocated central lists, each containing homogeneous telemetry information. When a reporting switch wants to report telemetry information to one of these lists, then it simply generates a single DTA packet according to Figure JL: make append header-structure fig. Switches are pre-loaded with brief lookup tables, containing a mapping between list IDs and IP addresses of the collectors that host their memory regions.

6.2.2 *Translation*

The translator will ingress a DTA-Append packet, specifying the list ID and the data to append. These lists are designed as ring buffers, where per-list head pointers are kept at the translator. A pre-loaded lookup table maps the list ID into stateful per-list metadata (including the buffer start address, head pointer offset, and list length), and are used to calculate the destination memory address of this piece of information. The head pointer is then incremented in preparation for inserting the next piece of telemetry data into the list. These head-pointers are periodically send down to the collector server, making operator more efficient by not having to constantly poll pointer information from the ToR before each data query.

6.2.3 *Operator Perspective*

Collectors keep two pointers per-list: one for the head, and another for the tail. The tail is pointing to the oldest unread piece of telemetry information, and is incremented each time that the operator extracts this oldest data. The head, however, is managed by the translator switch, and is incremented each time that new telemetry data is appended into the ring buffer. Operators can iterate over reported telemetry data in either chronological order (i.e., from newest-to-oldest, or oldest-to-newest), depending on if they base their data extraction on the head-pointer or tail-pointer.

This design allows for very efficient telemetry data processing. For example, a list containing locations of network-wide loss events could be processed in reverse chronological order, thereby delivering an up-to-date list of network-wide loss that might require controller reactivity.

JL: Refer to another section where we show DTA support in e.g., Sonata? Giving more examples where Append is useful.

6.2.4 *Append Collection Rate*

Each Append-call that ingress the translator will generate a single RDMA-WRITE packet, with no additional overheads in terms of inter-device communications or internal bus traffic. Thus, the DTA collection solution should be able to handle these operations at the exact same rate that the RDMA-capable NIC manages to process RDMA packets, and is therefore entirely dependent on the deployed RDMA hardware at the collector server(s).

6.2.5 *Pre-processing?*

JL: Append is a great candidate for in-translator pre-processing. E.g., appending flow sizes, and Translator can easily build list of per-epoch outliers. Add a DTABase flag for allow-preprocessing?

Calculate aggregated results, before appending raw into to ring buffer

- List of min/max values
- Median (stas101 way)
- Mean
- Threshold triggers (either static or dynamic). Outliers trigger CPU?

Duplicate detection? Pre-processing to determine IF value should even append to list? Append to one of several lists DEPENDING of pre-processing? E.g., ACTUAL network-wide outliers go into different list?

6.3 Sketch Aggregation?

JL: Shall we name and include this one? JL: Best to keep aggregated sketches in-ASIC, or RDMA down do x86 directly? Fetch&Add

6.4 Multi-Increment?

JL: Similarly to aggregated sketches. Like Key-Write, but with epoch counters from switches. JL: Improves support for per-flow counters

7. MONITORING INTEGRATION

JL: Discuss how DTA could fit with other (current) telemetry systems. JL: This sections is not connected to the evaluation or implementation, only focusing on discussing possible integration solutions

7.1 In-Band Network Telemetry

JL: Only sink-version, or also postcarding? Explain both Key-Write and Append useful, depending on scenario. E.g., paths: Key-Write. High-latency events: append.

7.2 Probabilistic In-Band Network Telemetry (PINT)

The base is INT. But can potentially use redundancy slots for encoding PINT pktID? (check meeting slides). Possible with our current primitives by: Key-Write N=1, key=key+pktID

7.3 Sonata

Two possible DTA locations... JL: Can we copy the design figure from their paper?

Sonata is a query-based telemetry system, where the control plane continuously updates on-switch traffic selection rules in an attempt to keep collected information relevant to the stated telemetry query.

There are two telemetry transfer steps in their architecture that are potential candidates for optimization through DTA. The first such is when packet tuples are sent from individual switches towards stream-processing servers. Their current solution uses the switch CPU to populate local key-value stores and then transmit the raw packet tuples towards steam-processing servers. DTA could allow for the transfer of these packet tuples either directly from the switch ASIC[2], or from the switch CPU post-pre-processingJL: re-word. Designing packet tuple transmission around the DTA Append primitive would significantly reduce server-side processing of incoming telemetry data, and increase the data read-rate. Instead of requiring costly packet I/O and parsing overheads, they can simply assume that relevant packet tuples are neatly presented in the in-memory ring buffer as they become available.

Transmit of query results from processors to controller runtime...

[2]Transmitting these packet tuples directly from the ASIC would require additional changes to the Sonata architecture, potentially requiring a hybrid approach where the switch CPU is still leveraged for building local key-value stores.

7.4 Sketches?

JL: Close overlap with the potential sketch-aggregation primitive section

8. IMPLEMENTATION

Text

8.1 Reporter

Only discuss DTA generation? Omit things like change detection and probabilistic triggers (not relevant to paper?) This is the same for all primitives. They simply set opcode and header according to DTA specification..

8.2 Translator

JL: Overall number of stages... JL: RDMA generation logic is re-used for all primitives. Significantly reducing footprint JL: Resynchronization through CNP parsing

8.2.1 Key-Write

Multicast.... Re-using logic... Minimal footprint

8.2.2 Append

Ring buffer...

8.2.3 Sketch-Update?

JL: POTENTIAL implementation solution. Not yet verified

8.2.4 Multi-Increment

JL: POTENTIAL implementation solution. Not yet verified

8.2.5 RDMA Connection

Initiator script. PSN counter. Desynchronization. PktID. Metadata-population. etc...

8.3 Collector

Explain C RDMA service, registering allocated buffer in Bluefield RDMA engine

9. EVALUATION

Text goes here

JL: Redesign evaluation and figures around RDMA-rate! E.g., y-axis ranging from 0 to 100% of RDMA line-rate. JL: Removes the issues of unoptimized RDMA setup (if Mellanox fails to respond), and we can argue this is due to the speed fully dependant on the RDMA hardware. 30% of RDMA-rate with out NIC should mean 30% of rate with another NIC as well. The Tofino will not bottleneck. (RDMA-rate already encompasses DRAM speeds etc.. no hidden factors)

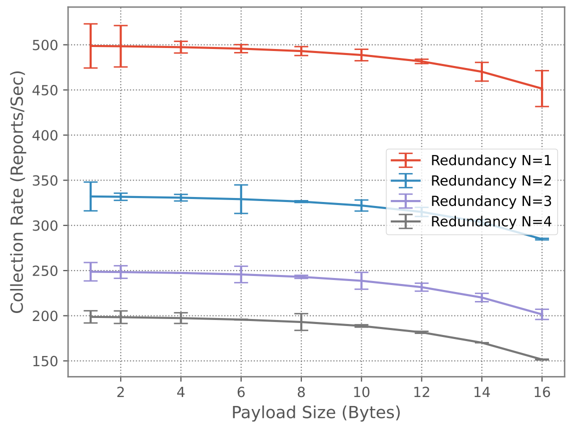

Figure 8: Collection rate of DTA Key-Write operations (PLACEHOLDER DATA)

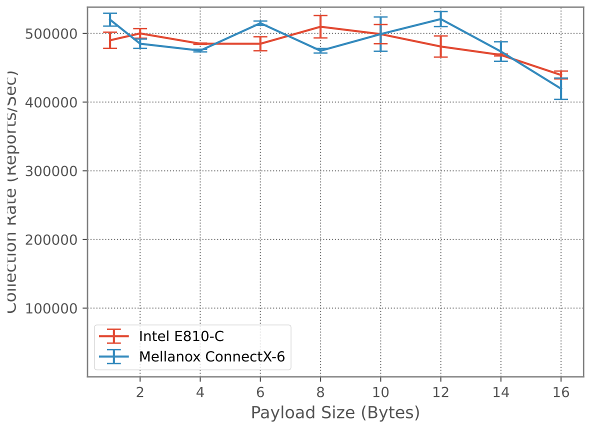

Figure 9: Collection rate of DTA Append operations (PLACEHOLDER DATA)

9.1 Collection Rate

Benchmark how fast we can collect telemetry reports. Be VERY up-front that this is directly correlated with the RDMA processing rate of the NIC. Explain how this historically increases faster than CPU speeds? Compare results to the advertised rate of our NIC? **Benchmark several RDMA NICs?**

9.1.1 Key-Write

Parameters: Size of report payloads, level of redundancy Also discuss return-error!

9.1.2 Append-to-List

Parameters: Size of report payloads

9.2 Marple

Evaluate collection of Marple data.
From paper: "we estimate that the cache eviction rate from a single top-of-rack switch can be handled by a single 8-core server running Redis [20].". Not difficult to beat this. Check Figure 9.

9.3 Querying rate

How fast can we query the reported information?

9.4 Key-Write Data Persistance

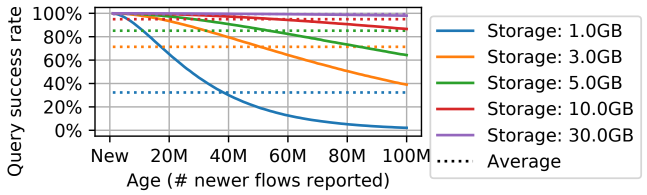

Figure 10: DTA Key-Write ages out stale data. This figure shows INT 5-hop path tracing queryability of 100 million flows at various storage sizes. The results are based on storing 160-bit values with 32-bit checksums, where each key writes into $N = 4$ different memory addresses.

JL: Show and discuss how reports age out of storage with Key-Write? (Re-use HotNets figure 10)

9.5 Collection Traffic

How much traffic is sent? Let's say different query-types. CAIDA and path tracing? Or random sampling (transmit data for every 100 packets). Comparing DTA vs RDMA vs UDP?

9.6 Resynchronization

Show the responsiveness of RDMA PSN resynchronization. Periodic ack requests to ensure periodic resync if initial resync failed?

9.7 PCI bus traffic?

JL: DMA results in reduced PCI load compared with e.g., UDP. Is this interesting?

10. DISCUSSION

Text goes here

11. RELATED WORKS

JL: this section is a clone of HotNets, but we likely now have space to expand it

11.1 Telemetry.

Traditional techniques have looked into periodically collecting telemetry data [4, 7, 6]. Even though these techniques generate coarse-grained data, they can be significant given the large scale of todays networks. The rise in programmable switches has enabled fine-grained telemetry techniques that generate a lot more data [16, 3, 2, 19, 20, 5]. Irrespective of the techniques, collection is identified to be the main bottleneck in network-wide telemetry, and previous works focus on either optimizing the collector stack performance [10, 17], or reducing the load through offloaded pre-processing [13] and in-network filtering [9, 12, 18, 19]. To our knowledge, all currently available collection solutions are CPU-based and therefore struggle with the same fundamental performance bottleneck. An alternative approach is letting end-hosts assist in network-wide telemetry [16, 8], which unfortunately requires significant investments and infrastructure changes.

11.2 RDMA in programmable networks.

Recent work has shown that programmable switches can perform RDMA calls [11], and that programmable network cards are capable of expanding upon RDMA with new and customized primitives [1]. Especially FPGA network cards show great promise for high-speed custom RDMA [15, 14].

12. REFERENCES

[1] E. Amaro, Z. Luo, A. Ousterhout, A. Krishnamurthy, A. Panda, S. Ratnasamy, and S. Shenker. Remote memory calls. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 38–44, 2020.

[2] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher. Pint: probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 662–680, 2020.

[3] T. P. A. W. Group. Telemetry report format specification. `https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_latest.pdf`. Accessed: 2021-06-23.

[4] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[5] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[6] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2014.

[7] C. Hare. Simple network management protocol (snmp)., 2011.

[8] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 404–421, 2020.

[9] Intel. In-band network telemetry detects network performance issues. `https://builders.intel.com/docs/networkbuilders/in-band-network-telemetry-detects-network-performance-issues.pdf`. Accessed: 2021-06-04.

[10] A. Khandelwal, R. Agarwal, and I. Stoica. Confluo: Distributed monitoring and diagnosis stack for high-speed networks. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 421–436, 2019.

[11] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan. Tea: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 90–106, 2020.

[12] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, and G. Antichi. Enabling event-triggered data plane monitoring. In *Proceedings of the Symposium on SDN Research*, pages 14–26, 2020.

[13] Y. Li, K. Gao, X. Jin, and W. Xu. Concerto: cooperative network-wide telemetry with controllable error rate. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 114–121, 2020.

[14] W. Mansour, N. Janvier, and P. Fajardo. Fpga implementation of rdma-based data acquisition system over 100-gb ethernet. *IEEE Transactions on Nuclear Science*, 66(7):1138–1143, 2019.

[15] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso. Strom: smart remote memory. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[16] P. Tammana, R. Agarwal, and M. Lee. Distributed network monitoring and debugging with switchpointer. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 453–456, 2018.

[17] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong. Intcollector: A high-performance collector for in-band network telemetry. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 10–18. IEEE, 2018.

[18] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, and G. Pongracz. Programmable event detection for in-band network telemetry. In *2019 IEEE 8th international conference on cloud networking (CloudNet)*, pages 1–6. IEEE, 2019.

[19] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, et al. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 76–89, 2020.

[20] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, et al. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 479–491, 2015.