# Complex Parsing for In-Network Acceleration of High-Energy Physics Experiments

Bjoern Sagstad
Illinois Institute of Technology
Chicago, USA
bsagstad@hawk.illinoistech.edu

Nishanth Shyamkumar
Illinois Institute of Technology
Chicago, USA
nshyamkumar@illinoistech.edu

Sophia Chen
Hinsdale South High School
Darien, USA
sophianaian@icloud.com

Wesley Robert Ketchum
Fermilab
Batavia, USA
wketchum@fnal.gov

Roland Sipos
CERN
Geneva, Switzerland
Roland.Sipos@cern.ch

James Kowalkowski
Fermilab
Batavia, USA
jbk@fnal.gov

Michael Wang
Fermi National Laboratory
Batavia, USA
mwang@fnal.gov

Nik Sultana
Illinois Institute of Technology
Chicago, USA
SCinet
Chicago, USA
nsultana1@iit.edu

## Abstract

This paper describes a novel application and evaluation of programmable networking in High-Energy Physics (HEP): a complete parser for the custom packet format used by Fermilab's DUNE experiment. Notably, this parser is implemented on a Tofino programmable network switch and evaluated on the FABRIC testbed by using network traffic generated by the ICEBERG DUNE prototype. The parsed network traffic consists of Jumbo Ethernet frames that contain digitizations of sensor readings from ICEBERG's detector.

This work is an early investigation into providing in-network processing support for HEP experiments. The paper describes DUNE's custom packet format, the challenges encountered when implementing a parser for that format, and an exploration of the techniques that are needed to overcome those challenges. We identify performance bottlenecks and discuss directions for future research.

## CCS Concepts

• **Networks** → **In-network processing**; **Programmable networks**.

## Keywords

Programmable Networking, P4, Tofino, High-Energy Physics, Data Acquisition (DAQ)

**ACM Reference Format:**
Bjoern Sagstad, Nishanth Shyamkumar, Sophia Chen, Wesley Robert Ketchum, Roland Sipos, James Kowalkowski, Michael Wang, and Nik Sultana. 2025. Complex Parsing for In-Network Acceleration of High-Energy Physics Experiments. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3731599.3767451

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
*SC Workshops '25, St Louis, MO, USA*
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767451

## 1 Introduction

Before any network traffic can be processed, it must first be parsed. In most cases, parsing of network traffic focuses on network headers, and network headers are designed to be easy (and fast) to parse. In some cases—such as in Deep Packet Inspection (DPI)—the payload is parsed too. Parsing is therefore a foundational feature that network functions extend in order to process network traffic.

This paper describes and evaluates a novel parser that is implemented on a hardware programmable network switch. This parser can completely handle the custom packet format used by Fermilab's DUNE experiment. This research is used to scope out ideas for future research in our community.

**Motivation.** This paper studies the in-network parsing of raw network traffic from a neutrino detector, and encounters some interesting technical challenges. This traffic is "raw" in the sense that it contains unprocessed samples from the detector's sensors, and it has not yet been reduced or filtered. This work is being done to build a foundation for in-network computing for High-Energy Physics (HEP) experiments. In turn, that effort is motivated to provide tools that can be used across different experiments. Building that foundation starts with providing in-network parsing support for actual frame formats used in HEP.

**Challenges.** While developing a parser for DUNE's packet format we encountered the following challenges:

- Frames contain data that is encoded in big endian (network) byte order, and other data that uses little-endian ordering. Little endian ordering is sensitive to word size—in DUNE's case, it uses 64-bit words. The Tofino's hardware programmable

parsers could not directly be used to carry out this conversion but we were able to craft a work-around at the cost of resources elsewhere in the chip.

- Raw traffic contains sensor readings in its payload, and our parser needed to parse that payload to provide a proof-of-concept implementation that can extract all the information from each packet. This is challenging to do on today's off-the-shelf switches, requiring low-level control of payload parsing; moreover, scaling this to high transmission rates is challenging. On top of this, DUNE uses Jumbo frames—this increases the processing that is required for each payload.
- Implementing this parser necessitated the use of specialized hardware primitives that are provided by Tofino's switching chip. We encountered a learning curve while using those primitives for a non-standard switching application, such as the type of parsing that is the topic of this paper.

**Methodology.** The development, testing, and evaluation methodology for this paper involved using network traffic captured from the ICEBERG detector [3]. ICEBERG emits network traffic that complies with DUNE's packet format, and provides us with high-fidelity test data. This traffic was streamed over the FABRIC [7] to reach a Tofino programmable network switch on which our DUNE parser was installed. This switch is one of the many resources that the FABRIC testbed makes available. Additionally, DPDK [1] is used for line-rate streaming across FABRIC resources, to carry out a performance evaluation of the parser.

**Findings.** This paper describes an "existence proof" of a full parser for DUNE's packet format on the Tofino programmable switch. Although it is able to fully parse frames (including their payloads) the throughput scalability of the current prototype is limited because of the recirculation technique we rely on. However, this paper describes an important milestone in this research, and it hopes to generate discussion in the wider community on developing specialized in-network accelerators for HEP. Two ideas for future research include the design of dedicated hardware accelerators that augment programmable switches, and the pre-processing of HEP traffic to increase its amenability for in-network processing.

**Contributions.** This paper makes the following contributions:

- It describes and evaluates the first full parser for the DUNE packet format that is implemented on a Tofino programmable switch.
- It describes the technical challenges that were encountered, and describes the problems that are yet to be solved—providing directions for future work.
- In providing extensive background on DUNE's design and the motivations for this research, this paper seeks to motivate a community-wide effort into developing high-performance in-network support for HEP experiments.

This paper is organized as follows: Section 2 describes related work and contrasts that work with this paper's contributions. Section 3 provides background details of the DUNE experiment, its packet format, and the programmable switch architecture that we target in this paper. Section 4 describes the design and implementation of our on-switch DUNE parser, referencing specific elements of the switch architecture. Section 5 evaluates the correctness, performance, and resource utilization of our DUNE parser by using workloads from a neutrino detector, and using resources on the FABRIC testbed. Section 6 discusses key technical highlights from the experience of developing this research, and provides directions for future work.

## 2 Related Work

This work involves encoding iterative behavior in a programmable switch in order to traverse a frame so that it can be fully parsed. The feature on a programmable switch that most closely maps to this iterative behavior involves recirculating a frame [23]. Recirculation has been used in research on a wide range of applications of programmable switches—including accelerating Machine Learning, cryptographic primitives, floating point computations, caching, reliable transmission, search, traffic generation, protocol boosting, load balancing, network coding, and network measurement [5, 8, 10–14, 18–21, 28–30, 32–34, 36–41]. Recirculation can be carried out using internal recirculation ports on the switching chip, or by setting switch ports into loop-back mode [24, 28, 36].

Compared to related work, and with earlier work on using a programmable hardware switch for parsing [35] and Deep Packet Inspection (DPI) [15], this paper appears to be the first to explore each of the following: (1) provide a full parser for the DUNE packet format, (2) describe and evaluate the dataplane parsing of Jumbo frames, (3) encounter and describe the conversion of little-endian ordered byte sequences in headers. This conversion is needed to enable in-switch computation using those parsed values—since the switch datatypes work on big-endian byte ordering.

Building on earlier work on in-network analysis of scientific streams [25–27], this paper appears to be the first to fully implement and evaluate such a prototype on a hardware programmable switch.

## 3 Background

The Deep Underground Neutrino Experiment (DUNE) is a particle physics experiment on neutrinos, particularly their oscillation between electron, muon, and tau flavors. Using Fermilab's particle accelerator, DUNE will generate the world's most intense neutrino beam and send it through two neutrino detectors. The "near" detector, located at Fermilab in Batavia, Illinois, is closer to the source, while the second, "far", detector will be installed 1,300 kilometers away at the Sanford Underground Research Facility in Lead, South Dakota. Both detectors are underground, with the near detector at a depth of 60 meters and the far detector at a depth of more than a kilometer. This diminishes the interference from cosmic rays and other background radiation with the detection of neutrino interactions. The distance between the detectors allows researchers to measure neutrino oscillations that occur between the two points, furthering our understanding of subatomic phenomena and the composition of the universe.

Both the near and far detectors in DUNE utilize Liquid Argon Time Projection Chambers (LArTPC), shown in Figure 1. A LArTPC is a particle detector consisting of a large volume of liquid argon with an electric field applied to it. When neutrinos collide with argon nuclei, they may form charged particles that ionize argon atoms as they travel through the chamber, producing free electrons and argon ions. These free electrons then drift towards a series of anode wire planes. As they pass through, the electrons induce a

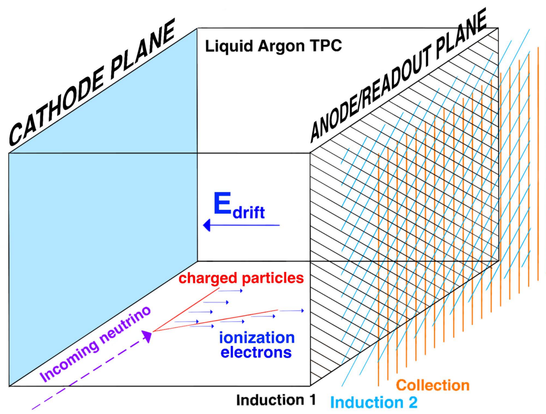


**Figure 1: Liquid Argon Time Projection Chamber, described in Section 3. Current is induced in a mesh of wires surrounding the cryostat. These wires are sampled and their values are digitized to provide the payload of the network traffic that is emitted by the detector, structured as shown in Figure 2.**

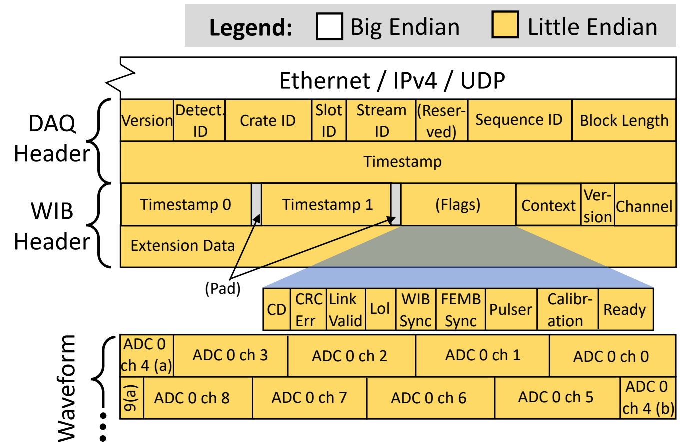


**Figure 2: Format for network traffic emitted by the DUNE detector and its prototypes, containing raw waveform data. This is described in Section 3.1.**

current in the wires of induction planes, producing bipolar signals before being collected by the collection plane, which generates a unipolar signal. Simultaneously, scintillation light produced when a neutrino interacts with liquid argon is detected by photomultiplier tubes (PMTs), providing timing information about the interactions. The measurement of the drift time and position of the electrons through the signals allows for the reconstruction of the charged particle's trajectory.

## 3.1 DUNE's Packet Format

Figure 2 shows the composition of data emitted by the detector described above. The detector emits Jumbo Ethernet frames through network interface ports. These ports are mounted on Warm Interface Boards (WIB)—which are "warm" in contrast to the cryogenic conditions inside the LArTPC [22]. Within these frames, the Ethernet header is followed by IPv4 and UDP headers [4, 6, 17, 31].

The Ethernet, IPv4, and UDP headers are encoded in network (big endian) byte order. Subsequently, the frame encoding switches to little-endian byte ordering with a 64-bit word size.

UDP datagrams encapsulate two custom headers and a payload. First, the 128-bit DAQ (Data Acquisition) header contains metadata on the detector-level source of the data—such as the detector's identifier and the part of the detector that produced the data. Second, the 128-bit WIB header describes the state of the WIB that emitted the Ethernet frame—such as whether it is in a diagnostic mode. These headers are followed by a 7,168-byte payload. The payload describes the waveform that was induced by particle collisions as described in the previous section. This waveform is reduced to a series of 14-bit ADC (Analog-to-Digital Converter) digitizations—one from each wire from the LArTPC's mesh that appears on the right side of Figure 1. These digitizations are concatenated as shown in Figure 2.

Downstream *read-out servers* receive these frames and carry out *event reconstruction*—a sequence of steps that analyze the data to determine particle tracks. These tracks show particles moving within the LArTPC's volume in three dimensions and across time. From these tracks we can infer which particles were involved in producing these tracks. Before reconstruction can take place, however, we must first parse the frame and its payload. This paper describes how to do that on a Tofino switch, whose architecture is described next.

## 3.2 Tofino Native Architecture

This paper describes an implementation of a DUNE frame parser in the P4 language [9] that targets a Tofino programmable switch. This switch implements the Tofino Native Architecture (TNA) with two frame-processing pipelines—an Ingress pipeline and an Egress pipeline, with a Traffic Manager between them [16]. Based on metadata from upstream logic, the Traffic Manager can process the frame in various ways, such as replicating, recirculating, or sending the frame to the Egress pipeline, after which the frame reaches a MAC block and is eventually transmitted to the network through a physical port on the switch.

Frames reach the Ingress pipeline through the MAC block, and are initially parsed by the Ingress Parser.[1] This parser extracts headers from the frame, under the control of the P4 program. The parsed headers and *intrinsic* metadata are carried by special switch resources called the **Packet Header Vector (PHV)**. Intrinsic metadata refers to information added by the switch and is independent of the P4 logic running on it. For example, the frame arrival timestamp and the physical port of arrival are some of the intrinsic fields. The parsed header fields are used in the **Match Action Unit (MAU)** stages of the pipelines. The MAU is used to match a parsed header field with a table key, and based on the match with that key, the

[1]This provides hardware parsing support in the Tofino for frame and packet headers—but it is unable to fully parse the DUNE packet format, as described later in the paper. Our DUNE parser therefore uses control logic in the ASIC to carry out additional parsing of headers, and to parse the payload.

MAU process the frame using a specific action. An action consists of P4 code that can involve reading and updating the contents of parsed fields and of switch metadata, such as the egress port. PHVs act as input and output for each stage of the pipeline. The Tofino1 architecture can support 4 parallel pipelines and each pipeline can have a maximum of 12 such MAU stages.

Once the MAU has completed its processing, the frames move to the Ingress Deparser where the parsed headers are combined back with the frame's payload. The metadata (such as whether to drop a frame, or on which port to transmit the frame) from the match-action stages is also propagated to the Deparser block.

Our implementation makes use of the recirculation capability of the TNA architecture. Specific ports in the Tofino are assigned as recirculation ports. If the destination port of the frame is set to one of these ports, that frame will eventually reappear at the Ingress Parser. Every frame arriving at the Parser is prepended with 16 bytes of information. The first 8 bytes are always intrinsic metadata (defined earlier) and the next 8 bytes contain the port metadata information [2]. In the Tofino, the port metadata is a struct defined by the P4 developer that is at most maximum 64-bits. Our program does not use this, therefore it skips over this field.

## 4 Design and Implementation

The high-level design idea of the DUNE parser described in this paper consists of sliding a window across each frame to process its contents. Sliding the window involves recirculating a frame to progressively process the bytes more deeply within that frame. Recirculation is a primitive provided by the switching chip.

The window's width is constrained by the size of the memory on the switch that can be used to store a prefix of the frame for processing. At each recirculation, this prefix focuses on a deeper part of the frame. While part of the prefix can be processed by the hardware parsing logic in the switch—for the Ethernet, IPv4, and UDP headers in Figure 2—the rest of this prefix must be processed by more general control logic in the switch because of the operations that are needed for byte re-ordering and payload processing.

Figure 3 shows the design of our system in terms of hardware blocks in the Tofino's programmable pipeline. The P4 program implementing this logic consists of 3500 lines of code. Of these, around 2500 lines implement testing functionality. This functionality is used to validate the correctness of the implementation, as described in Section 5. In our current implementation, all logic is implemented in the Ingress pipeline. This was done to simplify the implementation for this proof-of-concept. A future improvement could distributed the processing to the Egress pipeline too.

Working through the design, we start on the left of Figure 3 when a frame reaches the Ingress Parser from a MAC block, which receives a frame from a physical port on the switch. The Ingress Parser also receives recirculated frames. In our implementation, recirculated frames are given a custom recirculation header containing the current recirculation count. This is extracted after the intrinsic metadata and the port metadata fields that were explained earlier in Section 3.2.

The Ingress Parser is used to parse Ethernet, IPv4 and UDP headers, and to treat the subsequent bytes (containing the DAQ and WIB headers) as an opaque sequence of bytes. Parsing the DAQ and WIB headers requires additional hardware operations to interpret little-endian byte strings (Section 3.1). These operations involve reversing and recomposing the bytes into network byte order, in order to use them within the P4 program. These operations exceed what the Tofino's programmable parser can support, therefore we use the programmable logic outside of the parser to handle the DAQ and WIB headers. The DAQ and WIB headers are converted to big endian format inside the Ingress control block.

As explained in Section 3.1, the waveform consists of a sequence of 14-bit ADC values. Each of these values is sampled from a channel in the detector. A *channel* refers to the digitization of a single wire in the mesh sketched in Figure 1.

We parse waveform payloads in *segments* of 21 words of 64-bits each during every recirculation. The size of a segment is the "window size" mentioned earlier, and it is constrained by the PHV and parser limits on the Tofino. PHV limits are explained later in Section 5.3.

In our design, the `Reverse_stage` converts the endianness of the payload. At first, each word of the segment undergoes endianness switching from little to big endian. Second, we pack the extracted 14-bit ADC values into 16-bit fields since they are easier to work with on a Tofino. In this implementation, we also emit each parsed 14-bit value as a big-endian 16-bit value inside a custom header in order to validate and test correctness of the parsing.

We split the logic of `Reverse_stage` over multiple actions, with each action handling a single 14-bit field extraction. For example, a segment of 21 words (of 64-bits each) can fit 96 channels, thus requiring 96 separate actions. However, this led to compilation problems due to the complexity of the control block. To work around this, in our design, the `Reverse_stage` was split into three control blocks handling seven words with 32 actions each. These blocks are `Reverse_stage0`, `Reverse_stage1` and `Reverse_stage2`.

The outputs of `Reverse_stage` are then passed to the `Chunk Processor` to split each 64-bit word into the 14-bit adc$X$ch$N$ fields that are shown in Figure 2. This table is used for debugging purposes—it allows the developer to instruct the P4 program at runtime to create ADC values that can be tested in the Checkpoint tables that are described below.

The depth at which a frame's payload is parsed can be controlled by the control plane by setting a value in the `recirc_control_table`. This value is used when checking the recirculation number.

Correctness Checkpoint 1 and Correctness Checkpoint 2 serve as debugging checkpoints. The control plane deposits expected values for adc$X$ch$N$ into those tables, and those values are confirmed at runtime. This is used to ensure that the recirculation logic is implemented correctly. More details about the debugging process is described in Section 6.3.

At the final recirculation, the system emits the packet with original headers (Ethernet, IPv4, UDP, DAQ, and WIB) and the most recently-processed waveform segment. If the payload was not fully parsed (by using a lower recirculation number), then the remaining unparsed payload is forwarded too. This logic is used to test the parser, as described in the next section.

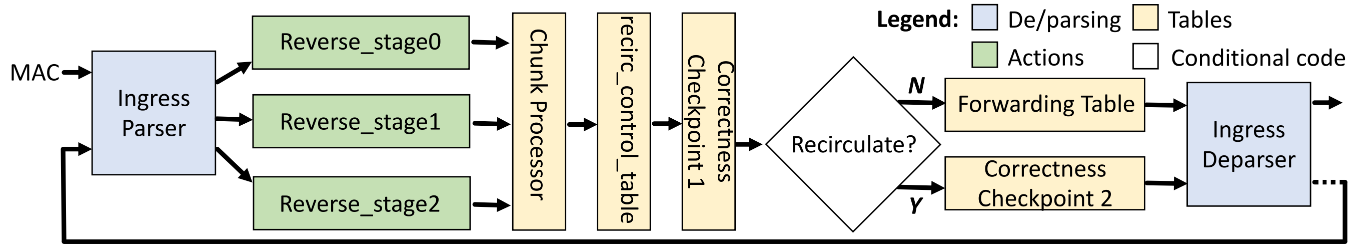


**Figure 3: DUNE network format parsing logic in P4, described in Section 4. Recirculation happens at the end of the Egress pipeline.**

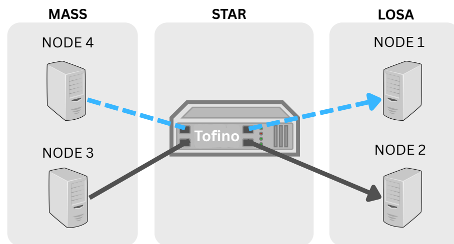


**Figure 4: Experiment topology on FABRIC, described in Section 5. This topology carries two streams in parallel: (1) (blue, dashed) Node 4 streams to Node 1, and (2) (black, solid) Node 3 streams to Node 2.**

## 5 Evaluation

The evaluation of the parser focused on three properties: correctness, performance, and resource utilization. For evaluation we used traffic that was produced by the ICEBERG detector [3], which is a small-scale DUNE prototype that produces DUNE-compliant network traffic.

**Setup.** The parser was deployed in the FABRIC topology that is shown in Figure 4. This topology uses three resources across geographically-distributed FABRIC testbed sites: a P4-programmable Tofino switch at STAR (in Chicago) and sender-receiver pairs at the LOSA (Los Angeles) and MASS (Massachusetts) sites.

### 5.1 Correctness

Given the complexity of the parsing task, it was important to carefully assess the implementation's correctness. The evaluation methodology for this task involved first writing a reference parser in Python, with which the P4 program's behavior was compared. In this Python reference parser, the Scapy library was used to parse and emit packets; we will see below how emitted packets are compared against those emitted by the Tofino switch. The Python reference implementation carried out endianness conversion and field extraction. The reference implementation was developed from scratch by closely following the DUNE design documents to build an independent implementation for testing our P4 code.

This Python reference was extended to simulate the sliding-window approach described in Section 4. Essentially, this approach projects a segment of bits that are currently within the scope of being processed by the Tofino, and emulates recirculation. This scope advances with each recirculation, as the processing is pushed deeper into the frame. This is used to check the recirculation logic in the P4 program. For example, if the P4 program was instructed to carry out 10 recirculations, then we expect the segment of interest to be at location $10 \times 168$ bytes inside the payload. In total, this reference implementation consists of 1500 lines of Python code.

To test correctness—of endianness conversion, field extraction and recirculation—we sent DUNE workloads to both the P4 program (running on a Tofino switch) and the Python program (running on the CPU), and captured their output into a pcap file. A short (30 line) Python program was used to compare the contents of these pcap files offline. This comparison checks that the fields related to the $21 \times 64$ bits extracted by the P4 program (for a given recirculation level) matches the data extracted by the reference implementation.

The outcome from this assessment was that the P4 implementation correctly matched the behavior of the Python reference implementation.

### 5.2 Performance

To evaluate performance, DPDK pktgen was used to generate continuous UDP traffic streams at 100 Gbps from each sender at LOSA in Figure 4 towards its corresponding receiver, with all traffic going through the P4 switch at STAR.

These traffic streams were used to examine how recirculation count affects throughput by measuring the received Gbps at different recirculation counts. We also compared running a single sender-receiver pair (single stream) against running two sender-receiver pairs (double stream) simultaneously, to investigate how the throughput changes with concurrent traffic loads.

Figure 5 shows the analysis of throughput loss as a function of the number of recirculations for both a *single* sender/receiver pair and a *dual* sender/receiver pair configuration. Both cases show exponential degradation of throughput when recirculation is increased. The dual-pair configuration demonstrates significantly steeper loss. At 1 recirculation the single stream has a throughput of 97.1Gbps, while the dual stream has 76.0Gbps. At 10 recirculations, the difference widens by an order of magnitude: 0.750Gbps versus 0.042Gbps, respectively. This collapse is caused by the congestion in the recirculation port from the incoming and recirculated traffic.

The outcome of this assessment was that a different approach is needed for in-network computing on this traffic workload and on

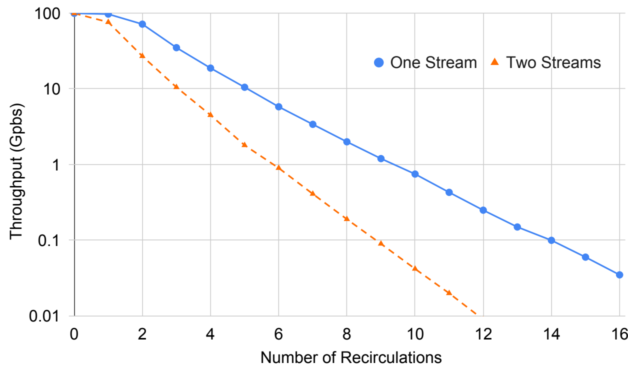


**Figure 5: Graph showing how throughput collapses as the number of recirculations increases. The graph shows that the collapse occurs faster as the number of streams increase. This is described further in Section 5.2.**

this hardware target. While recirculation is viable for this workload at low throughput, it clearly does not scale. Later in the paper we discuss future work ideas that avoid recirculation or bound it for smaller workloads to avoid congestion.

## 5.3 Resource Utilization

This section describes the utilization of Tofino resources when using the current parser prototype. Resource utilization reports were obtained from the Tofino's toolchain. This section explains the types of resources and their utilization.

As mentioned earlier, the Tofino1 pipeline has a maximum of 12 MAU stages, with each stage representing a set of switch resources. Table 1 shows that our P4 programmable logic maps to 3 stages, with the tables `recirc_control_table` and Checkpoint 1 mapped to the first stage, while `forwarding_table` and and Checkpoint 2 are mapped to the next stage.

SRAMs include exact match tables and Map RAM—which store stateful information such as registers and counters. The toolchain converts the word reversal logic into VLIW instructions. The Stats ALU field cover the ALU resources used to update the statistics on the stateful counters.

With regards to the internal state usage shown in Table 2, the Packet Occupancy Vector (POV) is used to the hold the header validity bits. For example, the 27 bits at the ingress here correspond to valid bits for the Recirculation header, Ethernet, IPv4, UDP, DAQ_RAW, WIB_RAW and 21 reversed words which form a segment.

The Packet Header Vector (PHV) holds the intermediate state between the different MAU stages. Some of the fields it stores include the IPv4 destination address, waveform segment and reversed words. The larger the segment being parsed, the more is the PHV usage. The PHV holds these intermediate states in fixed-size containers of 8, 16, or 32 bits. Looking at the breakdown of PHV usage across containers in Table 3, the effect is more pronounced with 80.7% of 16-bit container resources being utilized.

**Table 1: Resource utilization in the MAU. "Exact M. X-bar" is the Exact Match Crossbar, and "Ternary M. X-bar" is the Ternary Match Crossbar.**

| Stages | Exact M. X-bar | Ternary M. X-bar | Hash bit | Gateway |
|---|---|---|---|---|
| 3 | 2.34% | 1.01% | 5.6% | 10.4% |

| SRAM | Map RAM | TCAM | VLIW instr | Stats ALU |
|---|---|---|---|---|
| 7.5% | 6.9% | 1.4% | 22.9% | 41.7% |

**Table 2: Internal State Utilization on the Tofino.**

| Resource | Ingress (bits) | Usage (%) | Egress (bits) | Usage (%) |
|---|---|---|---|---|
| POV | 27 | 10.5 | 4 | 1.56 |
| PHV | 2795 | 68.2 | 13 | 0.317 |
| T-PHV | 560 | – | 352 | – |

**Table 3: PHV containers allocation on Ingress**

| | Containers | | |
|---|---|---|---|
| **Total usage (%)** | **8 bit (%)** | **16 bit (%)** | **32 bit (%)** |
| 68.2 | 3.7 | 80.7 | 75 |

The Exact Match Crossbar allows values to be read from the PHV and sent to tables with exact-matching lookups—i.e., exact match tables. Similarly, Ternary Match Crossbar relates to TCAMs. TCAMS are non-exact match tables—for instance, they can be used to carry out a longest-prefix match on IPv4 addresses. Hash bits are the key bits used in the key matching stage. For example, for the Forwarding table, the key is the IPv4 destination address which contributes 32 bits to the hash bits. Gateways represent the conditional branches in the P4 control blocks.

Finally, the Tag Along PHV (T-PHV) is used as a bypass mechanism. The bypass frees the PHV resources inside the pipeline for more important uses. Parsed headers that are not referenced in the Match-Action-Unit are sent directly to the Deparser. For example, header fields such as Ethernet source MAC, destination MAC addresses, IPv4 version field and UDP source port are not used after parsing in the Ingress Parser, and are thus passed through the T-PHV.

# 6 Discussion and Future Work

This section discusses key technical highlights from the experience of developing this research, and provides directions for future work.

## 6.1 Scaling

We see several ways to improve the scalability of in-network parsing to support in-network computing.

For applications of the DUNE parser that are only concerned with the DAQ and WIB headers, recirculation can be avoided. This would avoid the congestion problem that was observed earlier. Moreover, if only specific fields need to be accessed from those headers, then

the endianness conversion could be focused on those specific fields, instead of processing all the fields in the headers.

For applications of the parser that involve full payload processing—such as event reconstruction that was described in Section 3.1—then a different approach will be needed to avoid the congestion from recirculation. We envisage that a programmable switch could form part of a system design that is more suitable for full payload processing—involving hardware pipelines with more stages and memory. Based on the experience of building the parser described in this paper, we plan to build a prototype using reconfigurable hardware. We envisage the current prototype being used to handle the DAQ and WIB headers, and a complementary system that parses and computes over the payload.

## 6.2 Resource Constraints and Mitigations

Encountering and mitigating resource constraints provided us with ideas for future work. To extract the waveform—that is, adc*X*ch*N* values from the payload in Figure 2—we needed to slice 14-bit fields from 64 bit words. If the slice does not start at an 8-bit boundary, then the Tofino toolchain returns an allocation error. Since 14 does not divide 64 without leaving a remainder, we must concatenate 14-bit fields that were composed of slices from two adjacent 64-bit words. Another allocation error is returned if the two 64-bit words are located in different MAU groups—which is a collection of same-size containers. For example, a collection of 16 8-bit containers is 1 MAU group. To mitigate both kinds of allocation error, we use `@in_hash{}` to instruct the compiler to use an additional hardware resource that can access memory more flexibly, bypassing the constraints mentioned earlier. Consider the P4 statement:

```
adc14ch9 = reverse_word9[11:0] ++ reverse_word8[63:62];
```

Since the start of `[63:62]` is not byte-aligned, i.e, it is bit 7 and bit 8 of a byte, and when it is placed in bit 1 and bit 2 of the adc14ch9 field, then, the compiler will return an error. Enclosing the whole statement in `@in_hash {...}` will side-step this problem by instructing the compiler to use an additional hardware resource called a hash engine.

In our experiment, we process Jumbo frames with a 7,168-byte payload, which exceeds the 4096-bit PHV capacity of a single pipeline pass on the Tofino. This requires us to use recirculation to process the entire frame in segment-sized chunks. As mentioned earlier, the PHV consists of fixed-size containers of 8, 16, or 32 bits, into which the compiler stores header fields and metadata. Endian conversion and the use of 14-bit fields further stress PHV limits, since endian conversions require additional processing overhead, and 14-bit fields do not align well with container sizes. As we want to reduce the amount of recirculations needed to process the entire packet, we increase the number of words processed in one recirculation. Currently, as described in Section 5.3, the implementation's resource usage is below the container limit, and during development we attempted to find the maximum amount of words to optimize each pass through. This is made more complex by the structure of the waveform payload, as we need to process a group of 7 words (since the adc*N*ch*X* structure repeats itself every 7 words).

Building on the current prototype, we see an opportunity for future research into generating P4 code that reads and processes subsets of the DAQ data, to make more frugal use of hardware resources.

Moreover, in the current prototype's approach, we maintain separate headers for little-endian extraction (waveform_segment.word[0-20]) and big-endian processing (reversed_word[0-20]), which results in high PHV container usage. Future work can explore optimizations that involve in-place endian conversion, by reusing waveform_segment fields to store the converted values thus reducing this resource overhead.

Another possible resource saving can come from not passing the packet to the egress pipeline. At present, as seen from Table 2, while the egress pipeline is logically not relevant in the P4 code of this prototype, the packets still flow through these stages. This causes a small resource usage overhead, which can be avoided and instead allocated to the ingress pipeline, thus allowing for the waveform segment size to increase and thus reduce the number of recirculations.

## 6.3 Debugging techniques

This experience of developing and evaluating this parser provided opportunities to understand the difficulties around debugging the specific hardware that were were using. During debugging, most of the time was spent on inspecting P4 table entries, compiler-generated logs, and simulation output. The Tofino simulator provided step-by-step execution information that is unavailable when running on actual hardware. This was used during development, to understand Tofino behavior and limitations before deploying on the physical switch.

On the physical switch, we lack familiar debugging tools, such as configurable logic analyzers and diagnostic interfaces. To obtain visibility into the behavior of the running program, we used lookup tables to track control flow execution by matching on specific fields from headers or metadata at key points in the pipeline. These tables are designed to increment a counter when a match takes place, and we can read this counter from the control plane.

This technique was used in the two Checkpoint tables in Figure 3. These tables were used to diagnose bugs with our recirculation code. In Checkpoint 1 we monitor the recirculation number by incrementing a counter each time we match on `meta.is_recirc`, a field that tracks whether a packet is being recirculated. The table is set to match on `meta.is_recirc == X;`, where `X` is a value added in the switch using the control plane at runtime. The field `meta.is_recirc` gets set to 0 or 1 during parsing based on the ingress port of the packet. Once the look-up takes place on the checkpoint table, a counter is incremented upon a match. This counter is then polled from the control plane.

This approach helped isolate a bug that stopped all recirculation prematurely, and the combination of insights we obtained from checkpoint tables allowed us to localize bugs within the code. After making changes to the program, we noticed that the counter was no longer incrementing. We placed Checkpoint 2 after the "if" statement to provide visibility into the program flow.

We see an opportunity for research into specialized debugging support for in-network computing. Based on our experience, having runtime information about program behavior through the Checkpoint tables can greatly simplify the process of debugging. Future

work includes automatically generating inline support code for producing and querying Checkpoint tables.

## 6.4 Usability

We see an opportunity to improve the usability of programmable networking hardware for HEP scientists, to enable scientists to integrate this hardware as part of their experiment workflows. The previous two sections provided a glimpse of the specialized knowledge that is needed to program and debug this hardware target. Research into domain-specific tools and languages could improve the accessibility of programmable networking tools for scientists, and abstract low-level details of the hardware architecture—similiar to how CPU details are typically abstracted in scientific computing.

# 7 Conclusion

This paper described the first in-network parser for the complete DUNE packet format, implemented in a programmable network switch. The paper provided a design and explained how this design is underpinned by the programmable hardware elements that the switch makes available. A functional implementation of this implementation was developed, and the correctness, performance, and resource utilization of this implementation was evaluated using high-fidelity workloads on the FABRIC testbed. This work identified different primitives that helped and challenged the implementation, which we use to assess directions for future work.

# Acknowledgments

We thank the anonymous reviewers for their feedback. We thank Ron Rechenmacher, Bonnie King, and David Christian at Fermilab for providing the ICEBERG traffic samples. We thank Vladimir Gurevich at P4ica for providing feedback and advice about accessing Tofino resources, and we thank Ganesh Chennimalai Sankaran, Komal Thareja, and Mert Cevik from RENCI for help with using the Tofino switches on FABRIC. This work was supported by a URA Visiting Scholars Program Award 2025-S-21, by the FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the United States Department of Energy, and by the National Science Foundation (NSF) under award 2346499. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

# References

[1] [n. d.]. Data Plane Development Kit. https://www.dpdk.org/

[2] 2025. Open Tofino. https://github.com/barefootnetworks/Open-Tofino

[3] B. Abi and 965 others. 2020. Volume IV. The DUNE far detector single-phase technology. *Journal of Instrumentation* 15, 08 (Aug 2020), 189–192. doi:10.1088/1748-0221/15/08/T08010

[4] A. Abed Abud, C. Batchelor, K. Biery, J. Brooke, G. Crone, D. Cussans, P. Ding, A. Earle, E. Flumerfelt, J. Freeman, A. Habig, R. Halsall, P. Hamilton, N. Ilic, A. Kaboth, W. Ketchum, B. King, J. Klein, P. Lasorak, G. Lehmann Miotto, D. Newbold, D. Oliva, M. Roda, R. Sipos, A. Tapper, A. Thea, and S. Trilov. 2023. DAQ Design document. In *Final Design Review of the DUNE Data Acquisition (DAQ) System*. https://edms.cern.ch/document/2812882/1

[5] Aristide Tanyi-Jong Akem, Beyza Bütün, Michele Gucciardo, and Marco Fiore. 2025. Practical and General-Purpose Flow-Level Inference With Random Forests in Programmable Switches. *IEEE Transactions on Networking* (2025), 1–18. doi:10.1109/TON.2025.3564465

[6] William Badgett, Sophie Baron, Nuno Barros, Ines Gil Botella, Dave Christian, Philippe Farthouat, Mike Kirby, Elisabetta Pennacchio, and Terry Shaw. 2023. DUNE FD DAQ FDR Review Committee Report. In *Final Design Review of the DUNE Data Acquisition (DAQ) System*. https://edms.cern.ch/document/2811280

[7] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47.

[8] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. 2020. Designing Heavy-Hitter Detection Algorithms for Programmable Switches. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1172–1185. doi:10.1109/TNET.2020.2982739

[9] 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. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 87–95. doi:10.1145/2656877.2656890

[10] Tommaso Caiazzi, Mariano Scazzariello, and Marco Chiesa. 2023. Millions of Low-latency State Insertions on ASIC Switches. *Proc. ACM Netw.* 1, CoNEXT3, Article 22 (Nov. 2023), 23 pages. doi:10.1145/3629144

[11] Xiang Chen, Hongyan Liu, Zhengyan Zhou, Xi Sun, Wenbin Zhang, Hongyang Du, Dong Zhang, Xuan Liu, Haifeng Zhou, Dusit Niyato, Qun Huang, Chunming Wu, and Kui Ren. 2025. Phantom: Virtualizing Switch Register Resources for Accurate Sketch-based Network Measurement. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 1383–1398. doi:10.1145/3689031.3696077

[12] Penglai Cui, Heng Pan, Zhenyu Li, Penghao Zhang, Tianhao Miao, Jianer Zhou, Hongtao Guan, and Gaogang Xie. 2022. Enabling In-Network Floating-Point Arithmetic for Efficient Computation Offloading. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4918–4934. doi:10.1109/TPDS.2022.3208425

[13] Ina Berenice Fink, Ike Kunze, Pascal Hein, Jan Pennekamp, Benjamin Standaert, Klaus Wehrle, and Jan Rüth. 2025. Advancing Network Monitoring with Packet-Level Records and Selective Flow Aggregation. In *NOMS 2025-2025 IEEE Network Operations and Management Symposium*. 1–6. doi:10.1109/NOMS57970.2025.11073723

[14] Swati Goswami, Nodir Kodirov, Craig Mustard, Ivan Beschastnikh, and Margo Seltzer. 2020. Parking packet payload with P4. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies* (Barcelona, Spain) *(CoNEXT '20)*. Association for Computing Machinery, New York, NY, USA, 274–281. doi:10.1145/3386367.3431295

[15] Sahil Gupta, Devashish Gosain, Minseok Kwon, and Hrishikesh B Acharya. 2023. DeeP4R: Deep Packet Inspection in P4 using Packet Recirculation. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*. 1–10. doi:10.1109/INFOCOM53939.2023.10228996

[16] V. Gurevich and A.Fingerhut. 2021. P4 programming for Intel Tofino using P4 Studio. https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Vladimir-Gurevich-Slides.pdf

[17] R. Huang, P. Keener, J. Klein, B. Land, A. Madorsky, and A. Nikolica. 2024. DUNE WIB firmware.

[18] Fabian Ihle, Steffen Lindner, and Michael Menth. 2024. P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking. *IEEE Transactions on Network and Service Management* 21, 5 (2024), 5273–5290. doi:10.1109/TNSM.2024.3434337

[19] Wei Jiang, Hao Jiang, Jing Wu, and Pengcheng Zhou. 2023. Accelerating Network Coding with Programmable Switch ASICs. In *ICC 2023 - IEEE International Conference on Communications*. 1093–1099. doi:10.1109/ICC45041.2023.10278699

[20] Raj Joshi, Cha Hwan Song, Xin Zhe Khooi, Nishant Budhdev, Ayush Mishra, Mun Choon Chan, and Ben Leong. 2023. Masking Corruption Packet Losses in Datacenter Networks with Link-local Retransmission. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 288–304. doi:10.1145/3603269.3604853

[21] Piotr Jurkiewicz, Bartosz Kadziołka, Mirosław Kantor, Robert Wójcik, and Jerzy Domżał. 2024. Elephant Flow Detection With Random Forest Models Under Programmable Network Dataplane Constraints. *IEEE Access* 12 (2024), 158561–158578. doi:10.1109/ACCESS.2024.3485588

[22] Yichen Li. 2024. Introduction to LArTPC for Neutrino Detection and Cryogenic System. In *NuSTEAM/NuPUMAS at BNL 2024*.

[23] Devon Loehr and David Walker. 2022. Safe, modular packet pipeline programming. *Proc. ACM Program. Lang.* 6, POPL, Article 38 (Jan. 2022), 28 pages. doi:10.1145/3498699

[24] Daniel Merling, Steffen Lindner, and Michael Menth. 2021. Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4. *IEEE Access* 9 (2021), 34500–34514. doi:10.1109/ACCESS.2021.3061763

[25] Shivam Patel, Rigden Atsatsang, Kenneth M. Tichauer, Michael H. L. S. Wang, James B. Kowalkowski, and Nik Sultana. 2022. In-network fractional calculations using P4 for scientific computing workloads. In *Proceedings of the 5th International Workshop on P4 in Europe, EuroP4 2022, Rome, Italy, 9 December 2022*, Marco Chiesa and Shir Landau Feibish (Eds.). ACM, 33–38. doi:10.1145/3565475.3569083

[26] Ganesh C. Sankaran, Joaquin Chung, and Raj Kettimuthu. 2021. Leveraging In-Network Computing and Programmable Switches for Streaming Analysis of Scientific Data. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. 293–297. doi:10.1109/NetSoft51509.2021.9492726

[27] Ganesh C. Sankaran, Joaquin Chung, and Rajkumar Kettimuthu. 2024. Computing in Transit to Identify Rare Events in Streaming Scientific Data. In *2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. 1–4. doi:10.1109/ANTS63515.2024.10898859

[28] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 785–808. https://www.usenix.org/conference/nsdi21/presentation/sapio

[29] Mariano Scazzariello, Tommaso Caiazzi, and Marco Chiesa. 2024. Deliberately Congesting a Switch for Better Network Functions Performance. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. 1–6. doi:10.1109/ICNP61940.2024.10858581

[30] Siyuan Sheng, Huancheng Puyang, Qun Huang, Lu Tang, and Patrick P. C. Lee. 2023. FarReach: Write-back Caching in Programmable Switches. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 571–584. https://www.usenix.org/conference/atc23/presentation/sheng

[31] Roland Sipos. 2023. The Ethernet readout of the DUNE DAQ system.

[32] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 816–831. doi:10.1145/3603269.3604849

[33] Shie-Yuan Wang and Ting-Wei Yu. 2024. Using a P4 Hardware Switch to Enhance the Performance of Network File System. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*. 1918–1923. doi:10.1109/GLOBECOM52923.2024.10901827

[34] Sophia Yoo, Xiaoqi Chen, and Jennifer Rexford. 2024. SmartCookie: Blocking Large-Scale SYN Floods with a Split-Proxy Defense on Programmable Data Planes. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 217–234. https://www.usenix.org/conference/usenixsecurity24/presentation/yoo

[35] Yutaro Yoshinaka, Yuki Koizumi, Junji Takemasa, and Toru Hasegawa. 2025. Payload Queueing for Optimizing Complex Header Processing in Programmable Switches. *IEEE Transactions on Networking* (2025), 1–16. doi:10.1109/TON.2025.3582400

[36] Yutaro Yoshinaka, Junji Takemasa, Yuki Koizumi, and Toru Hasegawa. 2022. Feasibility of Network-layer Anonymity Protocols at Terabit Speeds using a Programmable Switch. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 292–296. doi:10.1109/NetSoft54395.2022.9844111

[37] Yutaro Yoshinaka, Junji Takemasa, Yuki Koizumi, and Toru Hasegawa. 2022. On implementing ChaCha on a programmable switch. In *Proceedings of the 5th International Workshop on P4 in Europe* (Rome, Italy) *(EuroP4 '22)*. Association for Computing Machinery, New York, NY, USA, 15–18. doi:10.1145/3565475.3569073

[38] Mai Zhang, Lin Cui, Xiaoquan Zhang, Fung Po Tso, Zhang Zhen, Yuhui Deng, and Zhetao Li. 2025. Quark: Implementing Convolutional Neural Networks Entirely on Programmable Data Plane. In *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*. 1–10. doi:10.1109/INFOCOM55648.2025.11044654

[39] Penghao Zhang, Heng Pan, Zhenyu Li, Penglai Cui, Ru Jia, Peng He, Zhibin Zhang, Gareth Tyson, and Gaogang Xie. 2022. NetSHa: In-Network Acceleration of LSH-Based Distributed Search. *IEEE Transactions on Parallel and Distributed Systems* 33, 9 (2022), 2213–2229. doi:10.1109/TPDS.2021.3135842

[40] Xiaoquan Zhang, Lin Cui, Fung Po Tso, Wenzhi Li, and Weijia Jia. 2024. IN3: A Framework for In-Network Computation of Neural Networks in the Programmable Data Plane. *IEEE Communications Magazine* 62, 4 (2024), 96–102. doi:10.1109/MCOM.001.2300587

[41] Yu Zhou, Zhaowei Xi, Dai Zhang, Yangyang Wang, Jinqiu Wang, Mingwei Xu, and Jianping Wu. 2019. HyperTester: high-performance network testing driven by programmable switches. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, Florida) *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 30–43. doi:10.1145/3359989.3365406