electronics

Article

100 Gbps Dynamic Extensible Protocol Parser Based on an FPGA

Ke Wang [1,2], Zhichuan Guo [1,2,*], Mangu Song [2] and Meng Sha [1,2]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, No. 21, North Fourth Ring Road, Haidian District, Beijing 100190, China; wangk@dsp.ac.cn (K.W.); sham@dsp.ac.cn (M.S.)
[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, China; songmg@dsp.ac.cn
* Correspondence: guozc@dsp.ac.cn

Abstract: In order to facilitate the transition between networks and the integration of heterogeneous networks, the underlying link design of the current mainstream Information-Centric Networking (ICN) still considers the characteristics of the general network and extends the customized ICN protocol on this basis. This requires that the network transmission equipment can not only distinguish general network packets but also support the identification of ICN-specific protocols. However, traditional network protocol parsers are designed for specific network application scenarios, and it is difficult to flexibly expand new protocol parsing rules for different ICN network architectures. For this reason, we propose a general dynamic extensible protocol parser deployed on FPGA, which supports the real-time update of network protocol parsing rules by configuring extended protocol descriptors. At the same time, the multi-queue protocol management mechanism is adopted to realize the grouping management and rapid parsing of the extended protocol. The results demonstrate that the method can effectively support the protocol parsing of 100 Gbps high-speed network data packets and can dynamically update the protocol parsing rules under ultra-low latency. Compared with the current commercial programmable network equipment, this solution improves the protocol update efficiency by several orders of magnitude and better supports the online updating of network equipment.

Keywords: high-speed network; extensible protocol parser; multiple queue management; ultra-low latency update

check for updates

Citation: Wang, K.; Guo, Z.; Song, M.; Sha, M. 100 Gbps Dynamic Extensible Protocol Parser Based on an FPGA. *Electronics* **2022**, *11*, 1501. https://doi.org/10.3390/electronics11091501

Academic Editor: Akash Kumar

Received: 31 March 2022
Accepted: 5 May 2022
Published: 7 May 2022

1. Introduction

In recent years, with the explosive growth of bandwidth-intensive industries, such as video streaming and the industrial Internet of Things, efficient data distribution and acquisition have gradually become the main requirements of internet applications. The traditional TCP/IP network architecture is based on an end-to-end communication mechanism between hosts. Therefore, the network entity does not support multi-address and variable address operations, which makes it difficult to meet the current content-based internet application mode.

In contrast, Information-Centric Networking (ICN) [1] adopts the idea of information identification and address separation, weakens the concept of the host, and allows naming information at the network layer to improve the security and flexibility of information transmission. As a new type of network, in order to facilitate the transition between networks and the integration of heterogeneous networks, the current mainstream ICN network architecture at home and abroad also considers compatibility with TCP/IP network characteristics. For example, DONA [2] uses flat names to replace hierarchical URLs, uses top-level resolution services to decouple content and host addresses, and uses IP routing for data transmission.

NetInf [3] published and analyzed information through the assembly interconnection network and directly used the analysis node to request content, and the information return was still based on the underlying network transmission. SEANet [4] is a network architecture with on-site, flexible, and autonomous features. It adopts SeaDP, a transport layer protocol that expands and supports ID-to-ID on the basis of IPV6, for efficient data block transmission.

Therefore, these ICN network architectures are built on the basis of common network protocols. By extending the customized ICN protocol on the basis of the general Ethernet protocol or the IP-layer protocol, a transmission channel based on information identification is established.

As the core module for protocol parsing in most network transmission devices, the parser's goal is to identify the protocol types in packet header fields and to allocate appropriate processing logic according to the protocol types, such as protocol-based packet filtering, more accurate routing and forwarding [5], etc. This article summarizes three key features of a high-performance parser in ICN network architecture: first, it can support efficient parsing of the general network protocols, which is the basis for integration with existing IP networks; second, it can identify customized ICN protocols to ensure that network equipment has better scalability; and third, packets can pass through the parser with deterministic low latency, which is the fundamental guarantee for the best performance of the system.

At present, the latency introduced by the pure software-designed parser is relatively large, and it is difficult to achieve zero packet loss in a high-speed network. However, the traditional ASIC-based hardware parser makes it difficult to flexibly expand the ICN protocol parsing rules due to the fixed chip performance. Even if a hardware platform that supports reconfiguration is used, it is often necessary to recompile the parsing logic to update the protocol parsing rules, which makes it difficult to update the protocol parsing rules in real-time.

The ICN is an important architecture of the future network that is mainly used in large-scale and high-concurrency network environments; therefore, it has high requirements in terms of the delay and throughput. In addition, with the continuous expansion of network services, ICN networks designed for different application scenarios need to support more network protocols. However, the current commercial network transmission equipment supports a limited number of protocols and cannot flexibly expand new protocol parsing rules according to the requirements of different ICN networks.

Therefore, in this paper, we propose a 100 Gbps dynamic extensible protocol parser (DEPP) based on FPGA. DEPP supports the real-time expansion of new protocol parsing rules based on common high-priority network protocols, thereby, facilitating flexible deployment in various ICN networks. In addition, the real-time nature of the protocol extension is beneficial to the online update of network equipment and ensures the normal operation of the network. The main contributions of our work are as follows:

(1) We use an FPGA to implement packet parsing and protocol management. The data transmission channel of the parser adopts a width of 512 bits and a clock frequency of 200 MHz, which enables it to identify various packet protocols at a line rate of 100 Gbps.

(2) We propose a dynamic extension mechanism for the protocols, which allows extending new protocol parsing rules in real-time at arbitrary locations in the existing parser tree by passing descriptors containing protocol information.

(3) We also provide a multi-queue management mechanism for extended protocols, which supports group management of extended protocol parsing rules. Different from existing parsers designed with a hierarchical pipeline structure, this method can manage the update and execution of various types of extended protocol parsing rules under the same framework and support the storage of more protocol parsing rules.

(4) The bus protocol conversion module is used for stream mode data conversion from an AXI4 bus to an Avalon bus. This module allows the parser to receive two bus protocol

signals, making it more flexible to deploy on the mainstream FPGA platforms, such as Intel and Xilinx.

The remaining part of the article is organized as follows: Section 2 briefly describes the main work and innovation of this paper. Section 3 shows the related work of the current software and hardware parsers. Section 4 describes the system architecture and main features of DEPP. Section 5 elaborates the process of the protocol update and packet parsing of DEPP in detail. Section 6 shows the deployment of DEPP on the Intel FPGA board and related performance analysis. Finally, the research work is summarized.

2. Related Work

Currently, commercial fixed hardware parsers can parse complex protocols; however, either they cannot effectively support the parsing of customized ICN protocols, or the flexibility of supporting new protocols is low [6], and the update of parsing strategies often requires high hardware design costs. In order to support the flexible expansion of the protocol parsing rules, many researchers prefer to choose to design software switches [7–9] to flexibly configure the parsing strategy, thereby, improving the scalability of the parser. However, it is conceivable that the software data processing system completely mounted on the CPU is not friendly in terms of the throughput and latency; therefore, this method is not conducive to deployment in high-speed networks.

Compared with the hardware packet parsing module mounted on the commercial chip and the software packet parsing program running on the server, the Field Programmable Gate Array FPGA [10] has greater advantages in terms of flexibility and high throughput, which allows designing circuits by writing (Verilog or VHDL) and other hardware description languages to generate binary executable files through simple synthesis and wiring and then quickly burn them to FPGA for testing, thereby, realizing the redeployment of the processing logic.

Furthermore, FPGA has the characteristics of low-latency and parallel data transmission and supports the design of high-performance processing logic for data packets in high-speed network communications. In particular, the introduction of the separation idea of data plane and control plane of the SDN (Software Defined Network) [11] into FPGA enables designers to realize the dynamic configuration of each processing unit deployed on FPGA in userspace through southward interface protocols, such as Openflow [12], which further improves the flexibility of FPGA.

Naous [13] and Liu [14], as well as the Blueswitch strategy proposed by Jong [15], have deployed Openflow switches in NetFPGA and provided 10 G data flow analysis examples. In these designs, users can update the protocol parsing policy through the southbound interface; however, these protocols must be within the range supported by OpenFlow. For example, OpenFlow V1.5 [16] can support up to 44 matching fields, while OpenFlow V1.0 can only support the parsing of 10 protocols. Therefore, its flexibility is limited. A more flexible method is expressed through a programmable data plane, such as protocol fuzzy forwarding [17] and protocol-insensitive P4 language [18], allowing designers to ignore the binding relationship between the protocol and device and reconfigure the data plane processing system on the software side.

RMT [19], developed in P4 language, uses an offline algorithm to store the protocol identifiers of each node in the protocol parsing tree in ternary addressable memory (TCAM) and then matches the protocol identifiers by cyclic look-up. When a packet arrives, the header is separated and sent to the parser. According to a predetermined protocol parsing process, the header is matched with the protocol information stored in TCAM, and the corresponding action is triggered after the matching is successful. Then, we locate the next packet header and repeat the operation, and the final output packet contains protocol information.

In RMT, although the real-time expansion of the protocol can be achieved by building a TCAM parser, the resource consumption and the delay brought by the layered look-up table parsing will increase with the complexity of the parsing rules, which will reduce the data processing performance of the device. Furthermore, TCAM can only be used to store protocol fields and their masks and thus can only be used for matching against fields extracted at the same location in the packet, which also limits the flexibility of the parser.

Ref. [20] showed a high-speed FPGA-based packet parser through the introduction of the PP (Packet Parsing) language and the corresponding compiler to achieve the programmability of the parsing module. In order to reduce network congestion, the solution introduces a deep pipeline processing mechanism with a longer processing delay and supports extending new protocol parsing rules but also requires a rewiring of the FPGA.

Refs. [21,22] also used deep pipelines to implement parsers. Through pipeline iterations, header fields of any length can be effectively analyzed. However, the layer-by-layer parsing method is more complex in high-bandwidth data transmission, and it is difficult to adapt to data-intensive scenes and high throughput requirements. Ref. [23] provides an automatic P4 to VHDL conversion method at a 100 Gbps line rate. In this scheme, in order to reduce the delay uncertainty caused by branching, the data packet needs to traverse all the protocol analysis structures, which makes the parser less flexible.

HyperParser [24] proposed a high-performance parser architecture for next-generation programmable switches and FPGA-based SmartNIC. Its butterfly network is optimized in terms of the packet parsing performance, logic resource occupancy, and device power. It is widely used in the design of encryption circuits. This solution supports both ASIC and FPGA deployment implementations, with low and deterministic latency, and adopts LUT-oriented design strategies to reduce the FPGA deployment time.

Compared with the previous scheme design, this method has a great improvement in performance, especially in the aspect of flexibility, supporting faster protocol update but also takes at least tens of seconds of loading time. If it is deployed in a 100 Gbps high-speed network, this time is enough to fill up the memory space of the network device, thereby, resulting in packet loss.

After the above analysis, we found that FPGA is widely used in various high-speed network devices due to its reconfigurable characteristics. However, the current design of reconfigurable parsers often adopts the method of rewiring the parsing logic, which usually requires several hours of compilation. Although some devices use TCAM to support online update of protocol parsing rules, due to its structural characteristics, it can only store protocol fields and mask information; thus, it does not support flexibly adding new protocol parsing rules anywhere in the parser tree.

DEPP, as a general-purpose parser, allows the real-time addition of new protocol parsing rules anywhere in the existing protocol parse tree through descriptors, thus, preventing temporary network downtime or network congestion caused by parser updates. Furthermore, it has the characteristics of low latency and high throughput, which facilitates flexible deployment on various types of ICN network transmission equipment.

Table 1 summarizes the brief methodology of existing parser techniques and our technique (described in § 3).Furthermore, we evaluate the performance of the parser in terms of the flexibility, latency, and throughput using three levels of 'high', 'middle', and 'low'.

Table 1. Protocol parser performance overview.

Type	Method	Flexibility	Delay	Throughput
software parser	Parser designed in software language on the host operating system	high	high	low
ASIC parser	Fixed-function commodity hardware parser	low	low	high
NetFPGA	A parser that supports the openflow protocol deployed in NetFPGA	low	low	high
RMT	A reconfigurable parser designed using P4 language	middle	low	high
HyperParser	A parser using butterfly network	middle	low	high
DEPP	dynamic extensible protocol parser based on FPGA	high	low	high

3. The DEPP System Structure

3.1. Parsing Engine Architecture

The parsing system implemented in this article is used to extract the protocol information of the network packets and to allocate different transmission channels for the packets of different protocol types. Its implementation architecture is shown in Figure 1. The system can be divided into control function development in userspace and the parsing logic design in FPGA hardware devices.

When the data arrives at the parser, its protocol parsing process is divided into parsing of the general network protocol and parsing of the extended ICN protocol, and then the data packets are sent to different transmission channels according to the protocol type. The extended ICN protocol parsing rule is updated in real time by the control plane through the descriptor. The specific functions of each module in the figure are shown as follows:

Data input: Receive high-speed data streams come from the MAC layer and bus control signal interface. This design uses an Avalon stream mode bus to transmit data at a 200 MHz clock frequency with a bandwidth of 512 bits, thereby, effectively supporting 100 Gbps of high-speed network data transmission.

Buffer module: The parser will generate a delay. The purpose of the buffer is to ensure synchronous output packets and their corresponding parsing results.

Pre_parser: The function of the protocol pre-parsing module is to identify the high-priority layer 2 to layer 4 network protocols in a packet, such as VLAN, QinQ, ARP, LLDP, IPV4, IPV6, TCP, and UDP. Furthermore, we synchronously extract the 1024 bits header information field within two clock cycles, which is convenient for further parsing the ICN extension protocol in the packet.

Pro_queue_mgr: The extended protocol multi-queue management module, which stores extended protocol parsing rules in the form of grouped multi-queues, supports the rapid update of protocol parsing rules and efficient parsing of packets.

Pro_adapter: Extended protocol adaptation module. This module is used to handle extended protocol descriptors from userspace. We adjust the transmission frequency of the descriptor to synchronize with the parser, and extract the extended protocol parsing rules contained in the descriptor. After extraction, the protocol information is transferred to pro_queue_mgr for storage. This module supports prefetching operations for multi-level extended protocols with dependencies as well as multi-protocol updates based on burst patterns.

Arbitration module: Receiving the protocol information extracted by the parser, the arbitration module allocates different transmission channels for the data packet according to the protocol type.

Pro_configure: The userspace protocol configuration module. Responsible for encapsulating extended protocol information in descriptors and transferring them to the FPGA data plane through the register interface.
Queue_monitor: The userspace queue monitoring module is used to monitor the protocol information in the queue entries and visualize the index of the queue where the protocol resides. As the protocol stored in the queue corresponds to the address of the queue, users can use the address to uniquely identify the extended protocol information and use the address index to delete invalid protocol information or allocate different packet transmission channels.

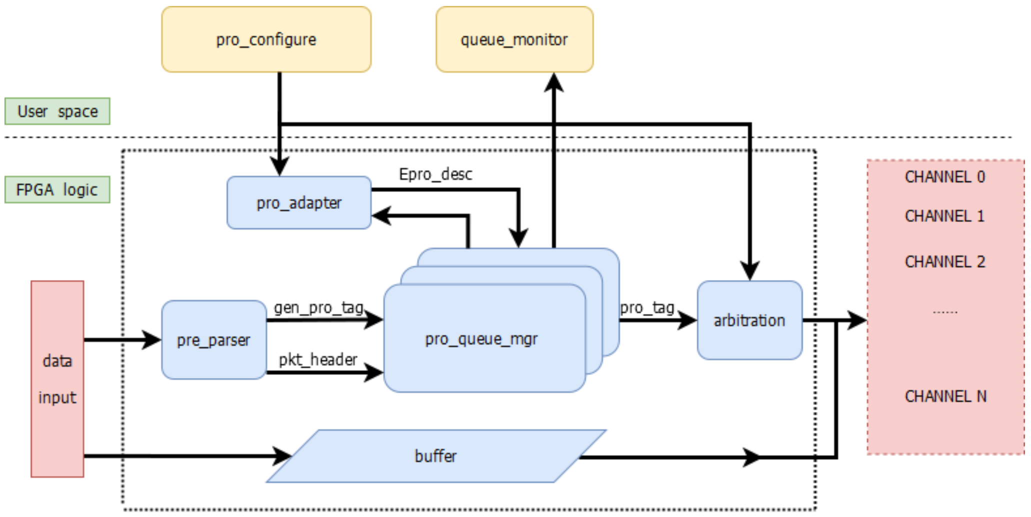


Figure 1. The abstract module of DEPP.

3.2. Dynamic Extension Mechanism

Due to the limited I/O performance of the host and the high latency introduced by CPU-based instruction set processing, it is difficult for software parsers to support high-speed network data transfers. On the other hand, the scalability of the protocol parsers deployed on programmable hardware facilities is poor. In order to update the protocol parsing rules, tedious processes, such as logic development, synthesis, and wiring, are often required. In order to enable the FPGA hardware parser to flexibly support new protocols, this paper proposes a dynamic extension mechanism using descriptor update protocols.

This lightweight protocol extension mechanism considers the integration with existing internet networks, allowing the dynamic adding of parsing rules of custom ICN protocols through extension protocol descriptors on the basis of existing parser trees. The method has high real-time performance and can upgrade the parser online without affecting the normal operation of the network. The extended protocol descriptor structure is shown in Table 2. It contains the reference protocol (Ref_pro), the valid field of the extended protocol (Epro_field), the extended protocol prefix (Epro_prefix), the relative position (offset), the next-layer extended protocol enable signal (next_en), and the level of the current extension protocol (level).

Table 2. Extended protocol descriptor.

Field	Width (Bits)	Purpose
Ref_pro	16	Reference protocol
Epro_field	16	Valid fields of extension protocol
Epro_prefix	5	The field's length
offset	8	The offset relative to the reference protocol
next_en	1	Next-level extended protocol enable signal
level	16	Level of the extension protocol

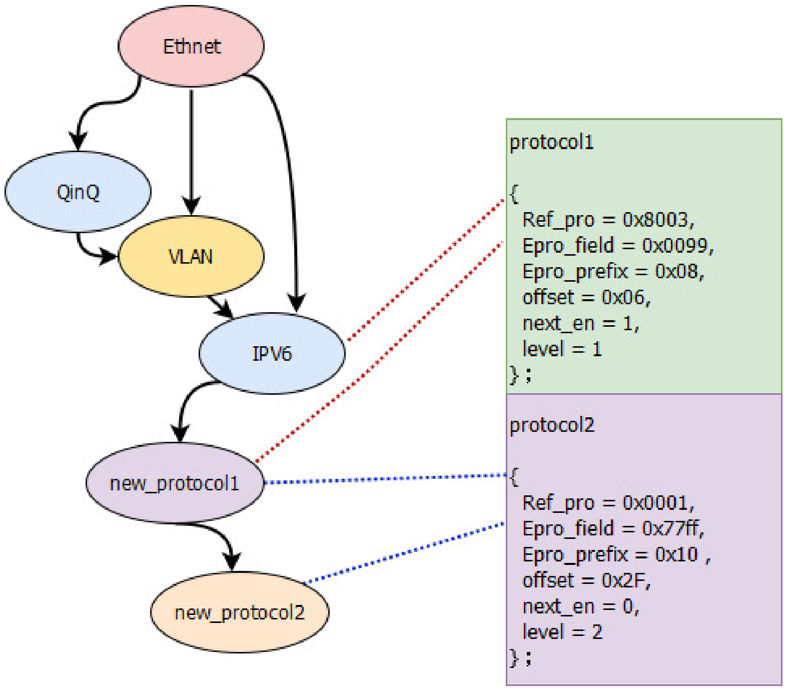
Figure 2 shows an example of a new protocol extension based on the IPV6 protocol. We write the extended protocol descriptor according to the format in Table 1 and transmit it to the FPGA data plane through the descriptor transfer interface. For new_protocol1, (Ref_pro = 0x8003) indicates that the reference protocol is IPV6—that is, new_protocol1 is a protocol added over IPV6. As the queue index is used to uniquely identify the protocol information in the multi-queue protocol management, in order to prevent the conflict between the protocol field and the queue index number, 0x8003 is used in the protocol descriptor to represent the IPV6 protocol.

For new_protocol2, its reference protocol (Ref_pro) is the index of the queue where new_protocol1 is located. The index is read by the queue monitor module or obtained by the feedback mechanism inside the parser. Next, we fill the extended protocol field (Epro_field) into a 32-bit register. If the field length does not meet 32 bits, we can fill in '0' and determine the actual valid field according to the mask information (Epro_prefix). (offset) represents the relative distance in bytes between the extended protocol fields and the IPv6 protocol identifier.

By specifying an offset, one can flexibly control the position of the new protocol. (level) indicates the level of the current extended protocol. The protocol added on the basis of general network protocols, such as VLAN and IPV6, is called the first-level extension protocol, and the second-level extension protocol is extended on the basis of the first-level extension protocol. Furthermore, (next_en) is used to determine whether there is an inner layer protocol. For new_protocol1, (next_en = 1) indicates that there is a second-level extension protocol over the current first-level extension protocol.

During parsing, the second-level extension protocol in the data packet can be further matched according to this flag. If next_en is still 1, we continue to judge whether there is a third-level extension protocol. The final output packet contains all levels of extended protocols. Through the investigation, we found that two to three levels of expansion protocols can meet the protocol expansion requirements of most of the existing ICN network transmission equipment types.

The introduction of the extended protocol descriptor mechanism simplifies the protocol update process and allows adding new protocol rules anywhere in the original protocol parse tree, thus, making the parser highly scalable. At the same time, the descriptor transmission interface encapsulated by the register can realize the real-time expansion of the new protocol, which is convenient for the online updating of network equipment or for testing the feasibility of the new protocol in an actual network environment.



```
protocol1
{
  Ref_pro = 0x8003,
  Epro_field = 0x0099,
  Epro_prefix = 0x08,
  offset = 0x06,
  next_en = 1,
  level = 1
};
protocol2
{
  Ref_pro = 0x0001,
  Epro_field = 0x77ff,
  Epro_prefix = 0x10 ,
  offset = 0x2F,
  next_en = 0,
  level = 2
};
```

Figure 2. Protocol extension diagram.

3.3. *Multi-Queue Protocol Management Mechanism*

This solution manages extended protocol parsing rules in the form of grouped multi-queues and supports dynamic storage and rapid parsing of extended protocols. As shown in Figure 3, the queue space is divided into N groups according to the reference general network protocol. The reference protocols supported by the current version are high-priority network protocols, such as VLAN, IPV4, and IPV6, and the initial location of Ethernet. We re-encode them from 0x8001. At the same time, contiguous queue storage space is also allocated for all second-level extension protocols. Furthermore, we record the status of the extended protocol information stored in each queue space through the queue status indicator 'Indc'.

In the protocol update operation, we determine the queue group to which the new protocol belongs by extending the Ref_pro information carried in the protocol descriptor and determine the available queue entry sequentially from the lowest bit of the queue group according to the queue status indicator. The new protocol information is then pushed into the entry, and the corresponding queue status indicator is updated. In the process of protocol parsing, we can similarly select the queue group that needs to be further retrieved according to the general network protocol identifier output by the pre-parsing module.

If the invalid protocol parsing rule needs to be deleted, the status indicator 'Indc' of the queue entry where the protocol is located can be directly deactivated. This extended protocol management method of packet multi-queue is beneficial to the efficient update and storage of the ICN extended protocol. At the same time, a unified extension protocol parsing logic is designed for queue management, which reduces the proportion of parser resources and packet jump operations to ensure high-speed data transmission.

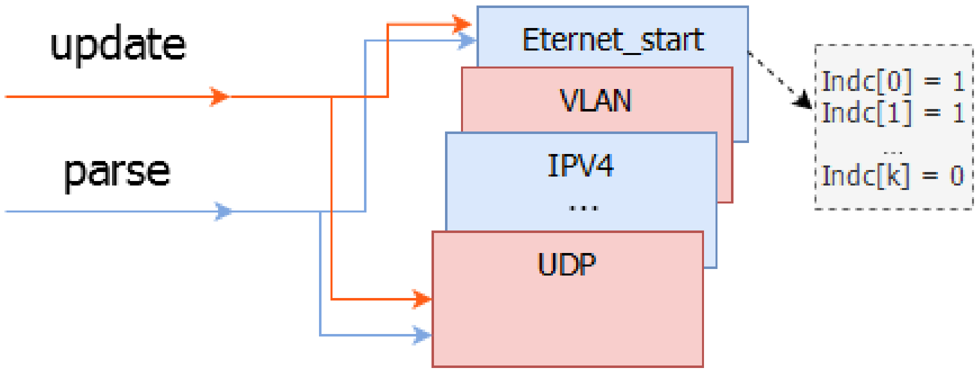


Figure 3. Diagram of the queue.

3.4. On-Chip Bus Translation Mechanism

Currently, the mainstream FPGA products and bus transmission protocols in the market are the AXI4 bus of Xilinx and Avalon bus of Intel, respectively. In order to facilitate more flexible deployment of DEPP on a variety of FPGA platforms, this paper also proposes a flow mode data conversion logic from the AXI4 bus to the Avalon bus, which is used for the conversion of bus control signals and data streams. We load this module on the data input end of DEPP so that the parser can support the parsing of AXI4 and Avalon bus data.

Comparing the structural characteristics of AXI4_st bus and Avalon_st bus in stream mode, we designed the conversion relationship as shown in Figure 4. This module mainly includes three parts: signal direct connection, control signal conversion, and high-speed data stream conversion. In the signal direct connection, bus signals with the same functions are directly transmitted: "ready", "valid", "last(eop)", and "user(channel)". According to AXI4_st bus validity flag bit "valid" and last frame identifier "last", the control signal conversion module generates the corresponding Avalon_st bus protocol packet start and stop signal "sop/eop".

Furthermore, the number of "0" in the binary AXI4_st byte valid flag "strb" signal is counted, and the Avalon_st bus invalid byte count signal "empty" is generated by the thermometer encoder. The byte reading order of data in different bus protocols is different. According to this characteristic, the high-speed data flow conversion logic is responsible for adjusting the byte reading order of AXI4_st bus data flow to the big-endian transmission structure, and zero-fill processing is performed for data packets that do not meet the bit width to ensure the aligned transmission of the data and control signals.

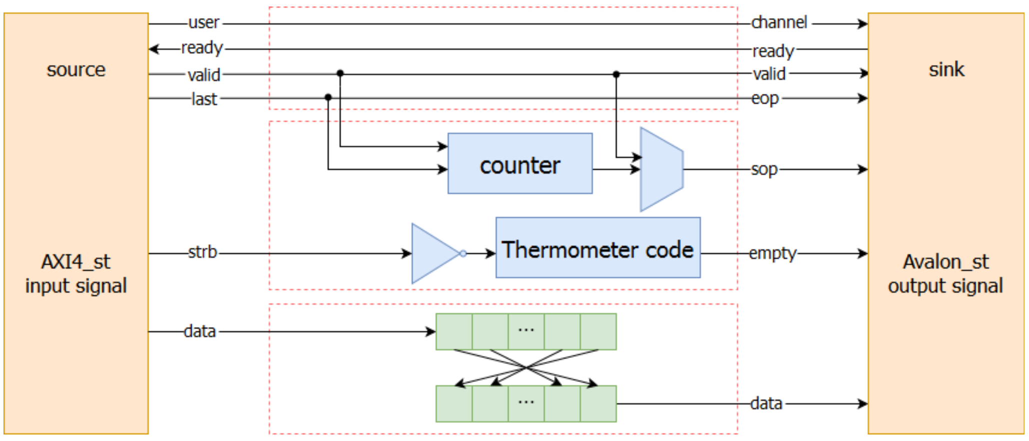


Figure 4. Bus protocol transformation diagram.

4. Work Process

This section introduces DEPP's dynamic protocol update and packet protocol parsing process in detail.

4.1. Protocol Update

Figure 5 shows the internal logic of the dynamic protocol update mechanism deployed on FPGA in detail. It mainly includes the pro_adapter module for processing descriptors and the pro_queue_mgr module used to manage protocol storage queues.

When a new descriptor arrives, it first enters the pro_adapter module, which is mainly responsible for clock synchronization and the extraction of descriptor information. In order to prevent timing instability caused by cross-clock domain operation, it is necessary to convert the clock frequency of the descriptor signal interface to the 200 MHz operating frequency used by the parser. Secondly, according to the known descriptor structure, we sequentially extract valid information, such as protocol fields and masks, in this module. Finally, along with the rising edge of the clock, the information is transmitted in parallel to the multi-queue protocol management module for storage.

For the first-level extension protocol on the general network protocol or the second-level extension protocol known to Ref_pro, we can directly extract the protocol parsing rules from the descriptor, and the mask information is obtained from Epro_prefix through the thermometer decoder. If the 16bits of Ref_pro are all 1, it means that we are updating the second-level extension protocol under the conditions of without knowing the location of the first-level extension protocol. At this time, the index of the queue where the first-level extended protocol is located needs to be passed to Ref_pro through the feedback mechanism to update the associated multilevel extension protocol.

When the protocol information reaches the pro_queue_mgr module, we first determine which queue group the protocol belongs to according to the Eth_pro signal obtained in the descriptor. Then, we determine the available queue entry in the group through the status indicator 'Indc' to add the current extended protocol information to the entry and activate the corresponding status indicator. The introduction of the queue status indicator facilitates real-time monitoring of the protocol storage status by the queue_monitor module. At the same time, invalid protocol information can be deleted by inactivating the corresponding queue entry status in the 'Indc' register. To prevent competition risks, when new protocol information arrives, the busy signal is asserted to suspend the packet parsing operation and cache the packet header information to be parsed until the update operation is complete.

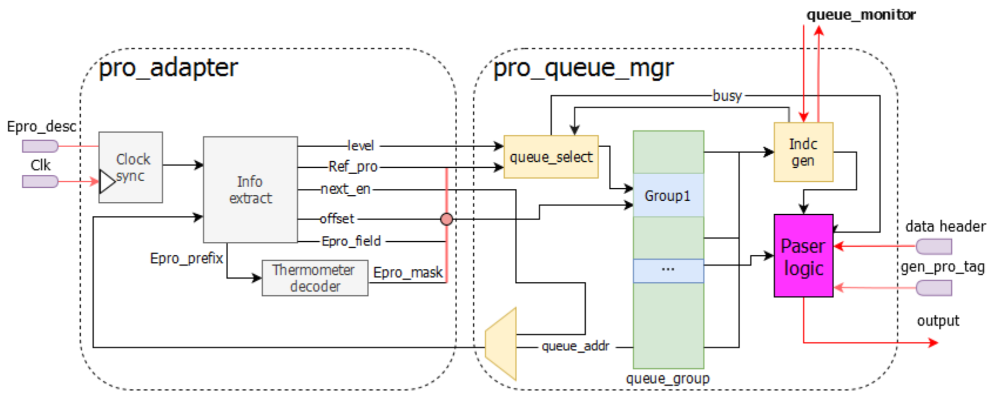
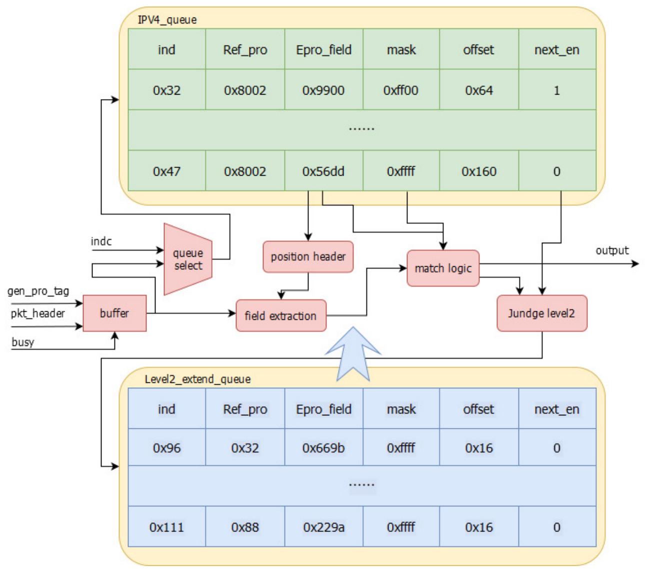


Figure 5. Protocol update microstructure.

4.2. Parsing Process

The packet protocol parsing process in this solution is divided into general protocol parsing and ICN extension protocol parsing. For the general protocol with fixed parsing rules, according to the sequence dependencies among the multi-level protocols, the combinational logic with no delay characteristic is used to parse all the commonly high-priority protocols within two clock cycles, which guarantees that data passes through the parser with minimal latency.

Figure 6 shows the parsing process of the extended ICN protocol in the multi-queue management module. The input signal is the packet header (data_header) extracted by the preparse module and the general protocol tag (gen_pro_tag). First, according to the busy signal, we judge whether the data needs to be cached until the protocol update is completed. Secondly, according to the general protocol type in the packet, the multi-queue grouping that needs to be further retrieved is selected. For example, if the packet contains the IPV4 protocol, the queue group of (Ref_pro == IPV4) should be further retrieved to determine whether the packet contains the extended protocol over IPV4.



ind	Ref_pro	Epro_field	mask	offset	next_en
0x32	0x8002	0x9900	0xff00	0x64	1
		......			
0x47	0x8002	0x56dd	0xffff	0x160	0

ind	Ref_pro	Epro_field	mask	offset	next_en
0x96	0x32	0x669b	0xffff	0x16	0
		......			
0x111	0x88	0x229a	0xffff	0x16	0

Figure 6. The extended protocol detection process.

The specific parsing process is as follows: According to the relative offset between the extended protocol and its reference protocol, we locate the position of the field to be identified in the packet header. Then, we extract the 32 bits matching field from this position and compare it with the protocol field and mask stored in the queue group in turn; after the matching is successful, we judge whether we need to parse the second-level extended protocol according to the next_en flag bit.

The parsing process of the second-level extension protocol is the same as the above method. However, its Ref_pro is the address of the queue where the first-level extension protocol is located. After the parsing is completed, the queue index where the extended protocol is located is output, and the corresponding transmission channel is allocated for the data.

5. Evaluation and Results

According to the overall framework of DEPP proposed before, we deployed it on Intel Arrias 10, which is a programmable 100 Gbps FPGA board. The parser is designed with SystemVerilog coding and uses C language to encapsulate the access control interface in user space for delivering extended protocol descriptors and queue status detection. The IXIA high-speed network traffic generation tool is used to generate test data streams to evaluate the performance of DEPP in terms of protocol extension and parsing.

The device connection is shown in Figure 7. The data packets generated by IXIA device are parsed by DEPP deployed on the FPGA and then distributed to different processing units of the R730 server for a statistic. Finally, we loop the packet back to the IXIA device.

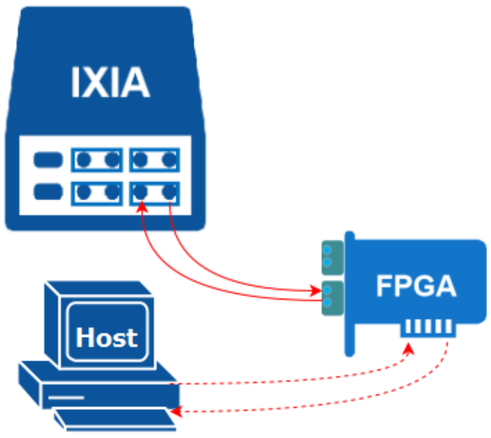


Figure 7. Equipment connection diagram.

The most important feature of the scheme is its high flexibility in the protocol extension. Using the descriptor mechanism, new protocol resolution rules can be added in any location of packets in real-time without affecting the normal running of network devices. Here, we test the average latency consumed by DEPP and TCAM parsers when adding new protocol parsing rules, respectively, and the test results are shown in Figure 8.

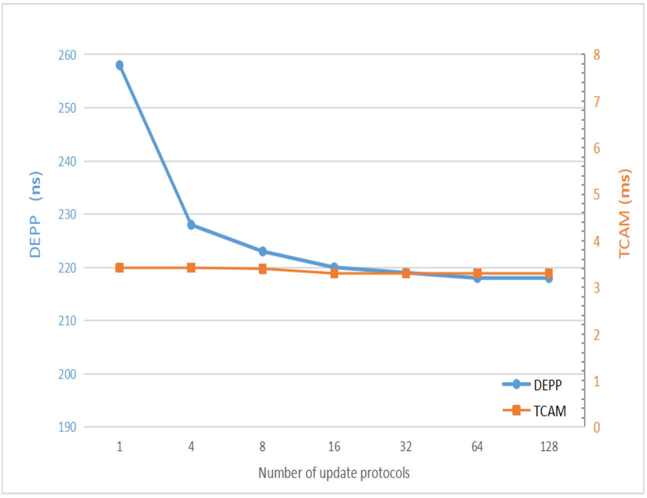


Figure 8. The average update delay.

The delay mainly includes the time to deliver the descriptor from the host and the time to process the descriptor in the hardware. In DEPP, we adopt a pipeline processing mechanism so that the average update delay gradually decreases with the increase of the number of burst transmission rules and is stable within 300 ns. In contrast, the time required for a TCAM-based [25] parser to update each rule is about 3.3 ms, which is four orders of magnitude higher than DEPP.

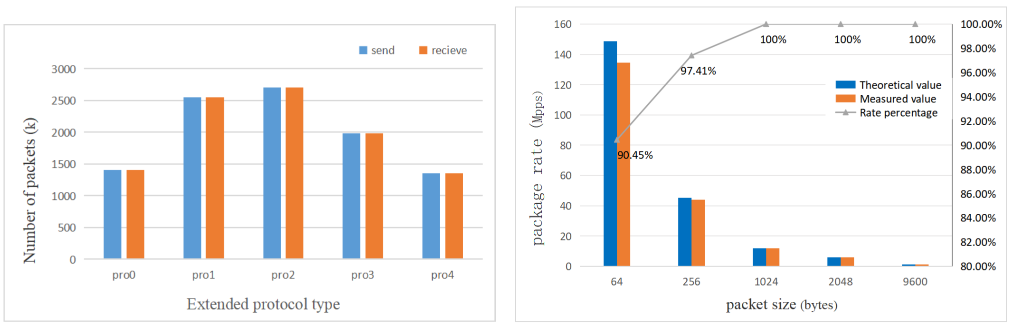
However, the current mainstream hardware reconfigurable parsers mainly rewrite the parsing logic to add new protocols and then reload them into the hardware through synthesis, wiring, and other operations. The overall time overhead is at least in minutes. Secondly, compared with the protocol management method based on TCAM, only the valid field of the protocol and its mask information can be stored in the TCAM, and the protocol positioning logic is not included; however, the descriptor can add protocol parsing rules of any length to any position of the data packet, which has higher flexibility.

In order to further verify DEPP's processing performance and packet throughput of the new protocol resolution rules, five protocol parsing rules as shown in Table 3 are added by extending the protocol descriptor. Pro0 to Pr03 are first-level extension protocols constructed based on high-priority general protocols, and Pro4 is a second-level extension protocol added on the basis of Pro3. Furthermore, we use the IXIA device for sending and receiving packet test.

Table 3. Descriptor example.

	Ref_pro	Epro_field	Epro_prefix	Offset	Next_en	Level
Pro0	IPV4	0x00fe	8	0x09	0	1
Pro1	UDP	0x0087	8	0x0a	0	1
Pro2	IPV6	0x00dd	8	0x06	0	1
Pro3	VLAN	0x8989	8	0x02	1	1
Pro4	Pro3	0x009b	8	0x1f	0	2

First, the fixed packet size is 1024 bytes, and 10 million packets with the above five protocols are randomly generated at a rate of 100 Gbps using IXIA equipment. The result is shown in Figure 9a, where the blue represents the number of different protocol packets sent by IXIA , and the orange represents the packets passing through the parser. It can be seen from the figure that the number of sending and receiving is the same, indicating that the parser can flexibly handle packets with different protocols . In Figure 9b, we fix the protocol type to Pro0, test the rate of packets of different sizes, and show the percentage of the actual received rate versus the theoretical value on the broken line. From the rate statistics results, except for a small amount of packet loss in the transmission and reception of line-rate small packets , stable packet reception of 100 Gbps can be achieved in other cases. This shows that the parser has high throughput characteristics.



(a) Multiple protocol types (b) Multiple packet sizes

Figure 9. Statistics of sending and receiving.

In Figure 10, we show the transmission delay of DEPP and the software parser designed with DPDK in a data forwarding system with fixed parsing rules. In order to show the test results better, the software parser was deployed on the Dell R740 server equipped with Mellanox ConnectX-5 NIC. The testing process is as follows: we randomly send data packets containing four-level protocol parsing rules, such as VLAN, IPV4, UDP, and Pro1, and forward them to the ixia device after passing through the parser.

Here, we use a fixed four-level protocol parse tree for evaluation, although both can support customizing more complex protocol parsing rules. From the delay test results, it can be clearly seen that, even if the kernel protocol stack is bypassed and the DPDK development kit with high-performance data packet processing is adopted, the transmission delay of data packets is at least microseconds or even milliseconds. In contrast, the transmission delay designed with the DEPP scheme is much smaller. Even if a data packet with a length of 4096 bytes is transmitted, the average delay can be stabilized at about 1200 ns, which greatly improves the processing efficiency of network data.

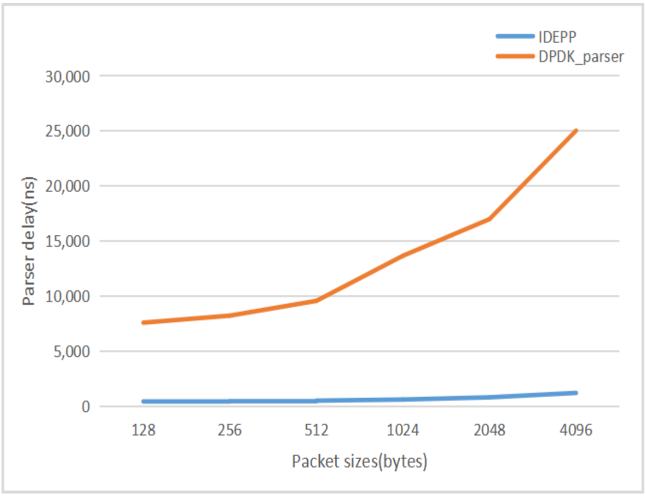


Figure 10. Parsing latency variation.

Table 4 shows the resource usage of DEPP deployed in Arria10 with different queue depths. The coverage of DEPP is small, leaving enough resource space for additional logic development. Compared with Openflow's limited support for 44 protocols, DEPP supports more protocol extensions and can meet a wider range of network protocol extension requirements.

Table 4. Resource utilization.

Queue Depth	**LUT (Total: 427,200)**		**Rigister (Total: 1,708,800)**		**RAM (Total: 55,562,240)**	
	Used	**Rate**	**Used**	**Rate**	**Used**	**Rate**
32	5945	1.39%	5334	3.12‰	22,656	4.07‱
64	10,375	2.43%	7222	4.23‰	22,656	4.07‱
128	22,226	5.20%	10,777	6.31‰	22,656	4.07‱

6. Conclusions

This paper proposes an ICN dynamically extensible protocol parser based on the FPGA platform, which supports a flexible expansion of protocol parsing rules and high-speed network packet parsing. It has a wide range of application values in data centers, computer clusters, and other traffic-intensive environments. In this solution, we introduced the extended protocol descriptor and multi-queue protocol management mechanism to realize dynamic updates and the efficient parsing of the customized ICN protocol parsing rules, which improved the flexibility and stability of the ICN network. Furthermore, the parser can be flexibly deployed on a variety of FPGA platforms through bus protocol conversion.

The experimental results show that DEPP supports adding new protocol parsing rules in real-time on the basis of the general protocol parsing tree and can, in a 100 Gbps high-speed network, accurately identify packet protocols. The high scalability of the parser enables it to be better deployed in various ICN network architectures and supports online updates of network devices, thereby, reducing the network downtime or network congestion caused by protocol updates and meeting future network requirements for high performance and flexibility.

At present, DEPP can support the flexible expansion of new protocol parsing rules at the end of the protocol parsing tree; however, it cannot support inserting new protocols in the middle of the original parsing process or even at the root. This is also the focus of my future work.

Author Contributions: Conceptualization, K.W. and Z.G.; Investigation, K.W., M.S. (Mangu Song) and M.S. (Meng Sha); Methodology, K.W. and Z.G.; Project administration, Z.G.; Software, K.W.; Supervision, Z.G.; Validation, K.W.; Writing–original draft, K.W.; Writing–review & editing, Z.G.,

M.S. (Mangu Song) and M.S. (Meng Sha). All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100). This work was also funded by the IACAS Frontier Exploration Project (Project No. QYTS202006).

Data Availability Statement: All the necessary data are included in the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xylomenos, G.;Ververidis, C.N.; Siris, V.A.; Fotiou, N.; Tsilopoulos, C.; Vasilakos, X.; Katsaros, K.V.; Polyzos, G.C. A survey of information-centric networking research. *IEEE Commun. Surv. Tutor.* **2013**, *16*, 1024–1049. [CrossRef]
2. Koponen, T.; Chawla, M.; Chun, B.-G.; Ermolinskiy, A.; Kim, K.H.; Shenker, S.; Stoica, I. A data-oriented (and beyond) network architecture. In Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, 27–31 August 2007; pp. 181–192.
3. Dannewitz, C.; Kutscher, D.; Ohlman, B.; Farrell, S.; Ahlgren, B.; Karl, H. Network of information (netinf)—An information-centric networking architecture. *Comput. Commun.* **2013**, *36*, 721–735. [CrossRef]
4. Wang, J.; Gang, C.; Jiali, Y.; Peng, S. SEANet: Architecture and Technologies of an On-site, Elastic, Autonormous Network. *J. Netw. New Media* **2020**, *6*, 1–8.
5. Parimala, M.; Jafari, S.; Riaz, M.; Aslam, M. Applying the Dijkstra Algorithm to Solve a Linear Diophantine Fuzzy Environment. *Symmetry* **2021** *13*, 1616. [CrossRef]
6. Gibb, G.; Varghese, G.; Horowitz, M.; McKeown, N. Design principles for packet parsers. In Proceedings of the Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 13–24.
7. Fernandes, E.L.; Rojas, E.; Alvarez-Horcajo, J.; Kis, Z.L.; Sanvito, D.; Bonelli, N.; Cascone, C.; Rothenberg, C.E. The road to BOFUSS: The basic OpenFlow userspace software switch. *J. Netw. Comput. Appl.* **2020**, *165*, 102685. [CrossRef]
8. Shirmarz, A.; Ghaffari, A. Performance issues and solutions in SDN-based data center: A survey. *J. Supercomput.* **2020**, *76*, 7545–7593. [CrossRef]
9. Zhang, T.; Linguaglossa, L.; Giaccone, P.; Iannone, L.; Roberts, J. Performance benchmarking of state-of-the-art software switches for NFV. *Comput. Netw.* **2021**, *188*, 107861. [CrossRef]
10. Kuon, I.; Tessier, R.; Rose, J. *FPGA Architecture: Survey and Challenges*; Now, Publishers Inc.: Delft, The Netherlands, 2007; pp. 135–253.
11. Shin, M.-K.; Nam, K.-H.; Kim, H.-J. Software-defined networking (SDN): A reference architecture and open APIs. In Proceedings of the 2012 International Conference on ICT Convergence (ICTC), Jeju, Korea, 15–17 October 2012; pp. 360–361.
12. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
13. Naous, J.; Erickson, D.; Covington, G.A.; Appenzeller, G.; McKeown, N. Implementing an OpenFlow switch on the NetFPGA platform. In Proceedings of the fourth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, USA, 6–7 November 2008; pp. 1–9.
14. Liu, T. Implementing Open Flow Switch Using FPGA Based Platform. Master's Thesis, Institutt for Telematikk, Trondheim, Norway, 2014.
15. Han, J.H.; Mundkur, P.; Rotsos, C.; Antichi, G.; Dave, N.; Moore, A.W.; Neumann, P.G. Blueswitch: Enabling provably consistent configuration of network switches. In Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Oakland, CA, USA, 7–8 May 2015; pp. 17–27.
16. OpenFlow Switch Specification, v.1.5.0. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-re-sources/onf-specifications/openflow/openflow-switch-v1.5.0.pdf (accessed on 20 March 2022).
17. Song, H. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 12–16 August 2013; pp. 127–132.
18. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
19. Bosshart, P.; Gibb, G.; Kim, H.-S.; Varghese, G.; McKeown, N.; Izzard, M.; Mujica, F.; Horowitz, M. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 99–110. [CrossRef]
20. Attig, M.; Brebner, G. 400 Gb/s programmable packet parsing on a single FPGA. In Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems, Brooklyn, NY, USA, 3–4 October 2011; pp. 12–23.
21. Cabal, J.; Benáček, P.; Kekely, L.; Kekely, M.; Puš, V.; Kořenek, J. Configurable FPGA packet parser for terabit networks with guaranteed wire-speed throughput. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 249–258.

22. Zolfaghari, H.; Rossi, D.; Cerroni, W.; Okuhara, H.; Raffaelli, C.; Nurmi, J. Flexible software-defined packet processing using low-area hardware. *IEEE Access* **2020**, *8*, 98929–98945. [CrossRef]
23. Benácek, P.; Pu, V.; Kubátová, H. P4-to-vhdl: Automatic generation of 100 gbps packet parsers. In Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, DC, USA, 1–3 May 2016; pp. 148–155.
24. Liu, H.; Qiu, Z.; Pan, W.; Li, J.; Huang, J. HyperParser: A High-Performance Parser Architecture for Next Generation Programmable Switch and SmartNIC. 2021. Available online: https://conferences.sigcomm.org/events/apnet2021/papers/apnet2021-6.pdf (accessed on 20 March 2022).
25. Jin, X.; Liu, H.H.; Gandhi, R.; Srikanth, K. Dynamic scheduling of network updates. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 539–550. [CrossRef]