HyperParser: A High-Performance Parser Architecture for Next Generation Programmable Switch and SmartNIC

Huan Liu, Zhiliang Qiu, Weitao Pan, Jiajun Li, Jinjian Huang
State Key Laboratory of Integrated Service Networks, Xidian University
Xi'an, Shaanxi, China

ABSTRACT

Programmable switches and SmartNICs motivate the programmable network. ASIC is adopted in programmable switches to achieve high throughput, and FPGA-based SmartNIC is becoming increasingly popular. The programmable parser is a key element in programmable switches and SmartNICs, which can identify the protocol types and extract the relevant fields. The programmable parser for the next generation programmable switches and SmartNICs requires a significant improvement in PPAL (performance, power, area, and latency), which is quite challenging. According to the Ethernet roadmap, 800 Gbps and 1.6 Tbps are expected to be the future switch interface speeds after 2022, which leads to higher throughput of the parser. Meanwhile, the end of Dennard scaling and the slowdown of Moore's Law result in limited power and area. Besides, the need for low-latency and low-jitter operations at the datacenter scale continues to grow.

Aforementioned requirements on PPAL inspire us to propose HyperParser, a high-performance parser architecture for next generation programmable switches and FPGA-based SmartNICs. The key innovation of HyperParser is the adoption of the butterfly network, which is widely used in cryptographic circuits. HyperParser supports ASIC and FPGA implementations, with low and deterministic latency. The PPAL of the ASIC implementation are 3.2-6.8 Tbps, 0.55 W, 2M gates, and 11.7 ns, and the PPAL of the FPGA implementation are 1.3-2.8 Tbps, 16.2 W, 43K LUTs, and 40 ns. The source code of HyperParser has been released on Github.

APNet 2021, June 24–25, 2021, Shenzhen, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8587-9/21/06…$15.00
https://doi.org/10.1145/3469393.3469399

CCS CONCEPTS

• **Hardware** → **Networking hardware**; • **Networks** → **Programmable networks**.

KEYWORDS

Programmable parser, Parsing, Programmable data plane, SmartNIC, FPGA

ACM Reference Format:
Huan Liu, Zhiliang Qiu, Weitao Pan, Jiajun Li, Jinjian Huang. 2021. HyperParser: A High-Performance Parser Architecture for Next Generation Programmable Switch and SmartNIC. In *5th Asia-Pacific Workshop on Networking (APNet 2021), June 24–25, 2021, Shenzhen, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3469393.3469399

1 INTRODUCTION

Programmable switches and SmartNICs motivate the programmable network [25]. The benefits provided by these new devices could extend beyond software-defined networking use cases and they prompt a shift towards a fully programmable cloud [5]. Programmable switch can be used for KV-store[16], load balancing [21], and congestion control [18]. SmartNIC can offload the workload of the host CPU, and we focus on the FPGA-based SmartNIC [8]. The key to achieving programmability is the ability to identify and process various packet headers, which is based on programmable packet parsers [3, 19].

The programmable parser for the next generation programmable switches and SmartNICs requires a significant improvement in PPAL (performance, power, area, and latency), which is quite challenging. **Performance.** According to the Ethernet roadmap, 800 Gbps and 1.6 Tbps are expected to be the future switch interface speeds after 2022 [11], which leads to higher throughput of the parser. **Power.** The end of Dennard scaling leads to the dark silicon problem [6], which results in a limited power budget. **Area.** Moore's Law is slowing down [12], and advanced process technology has become increasingly expensive. A lower chip area can effectively reduce cost. **Lantency.** There is no doubt that the need for low-latency and low-jitter operations at the datacenter scale continues to grow [17].

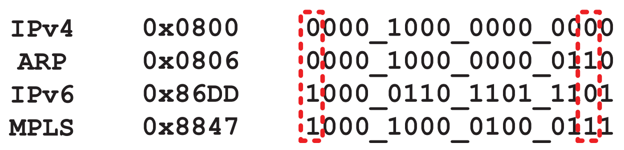

IPv4 0x0800 0000_1000_0000_0000
ARP 0x0806 0000_1000_0000_0110
IPv6 0x86DD 1000_0110_1101_1101
MPLS 0x8847 1000_1000_0100_0111

Figure 1: Bit Selection Strategy

For ASIC-based parsers, existing schemes work well for the link speed under 800 Gbps [28]. But 1 Tbps and beyond are still challenging, especially for the guaranteed power, area, and latency. For FPGA-based parsers, PPAL is also significant. Besides, FPGA-based parsers suffer from a long deployment time which is caused by the re-synthesizing of RTL (Verilog or VHDL) code [1, 4, 15, 24].

In this paper, we propose HyperParser, a high-performance parser architecture for next generation programmable switch and SmartNIC. HyperParser can support ASIC and FPGA implementations with a single architecture. A single instance of ASIC-based HyperParser can achieve 3.2-6.8 Tbps, which is far more enough for the coming 1.6 Tbps Co-Packaged Optics [7]. A single instance of FPGA-based HyperParser can achieve 1.3-2.8 Tbps, which can easily hold 800G Ethernet and is promising for 1.6T Ethernet. Besides high throughput, HyperParser achieves better power, area, and latency than state-of-the-art. Especially for the FPGA implementation, HyperParser can reduce the deployment time from 10s of minutes to 10s of seconds by adopting a LUT-oriented design strategy. We sincerely look forward to sharing our idea with the community, and the source code of HyperParser has been released on Github[1].

2 MOTIVATION AND BASIC IDEA

2.1 Need for Better PPAL

The parser is used to identify the protocol type of the incoming packets and extract the concerned fields to the Packet Header Vector (PHV). The process of identifying is sequential and typically time-consuming. The crossbar is widely used in the process of extracting, which is area-consuming. Previous works leverage finite state machine (FSM) [2, 3, 10] or pipeline [4, 28] to realize parsers. **FSM-based schemes** are flexible, and they can parse headers of any length efficiently. On the other hand, FSM-based schemes suffer from limited throughput, which is due to the limited bus width. Multiple instances of FSM-based parser can sustain higher aggregate throughput, but they cannot parse packets arriving through a single port [27]. Contrary to the FSM-based schemes, **pipeline-based schemes** can achieve high performance, but the number of pipeline stages is a problem. State-of-the-art Ethernet controller can parse up to 504 bytes[2], and state-of-the-art switch ASIC can parse up to 190 bytes[3]. A deeper pipeline can support a deeper parsing, at the cost of worse power, area, and latency. The latency problem is even more significant for short packets(e.g., 64 bytes). A new architecture is desired to achieve better PPAL.

2.2 Basic Idea

As mentioned in Section 2.1, protocol identifying and field extracting are the two basic operations of parsers. Our basic idea is to speedup the identifying and reduce the cost of extracting. To achieve a throughput of multi-terabits per second by one instance, either higher clock frequency or a wider bus should be adopted. A meaningful observation is the performance wall [26], which indicates that the clock frequency of switch silicon is in the order of 1 GHz. Therefore, the adoption of a wider bus is the only choice. HyperParser adopts 4096 bits (i.e., 512 bytes) as the bus width, and even the longest header can be accommodated in a single bus word.

As for protocol identifying, the bit selection strategy is adopted to achieve high throughput. As shown in Figure 1, only two bits are needed to identify four upper protocols of Ethernet frames. We found that 32 bits (we call them protocol bits) are enough for the four classic parse graphs in Figure 5. The further question is how to extract protocol bits. A naive choice is a crossbar, which is area-consuming. We choose an inverse butterfly network instead of a crossbar, and the $O(n^2)$ area complexity can be reduced to $O(nlog(n))$, where n is the bus width.

Fields extracting is also faced with the crossbar problem. Instead of an inverse butterfly network, a butterfly network is adopted here. The $O(n^2)$ area complexity of the crossbar can also be reduced to $O(nlog(n))$. With the adoption of butterfly and inverse butterfly networks, HyperParser can achieve better performance, power, area, and latency at the same time.

2.3 Why Selecting Butterfly Network

Butterfly and inverse butterfly networks have been widely used in cryptology [14], they can perform bit permutation efficiently. Butterfly network and inverse butterfly network are well studied, and more details about them can be founded in [14][13]. As shown in Figure 2(a) and Figure 2(c), the inverse butterfly network can extract protocol bits and gather

[1]https://github.com/FPGA-Networking/HyperParser

[2]https://www.intel.com/content/www/us/en/products/details/ethernet/800-controllers/e810-controllers.html

[3]https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series

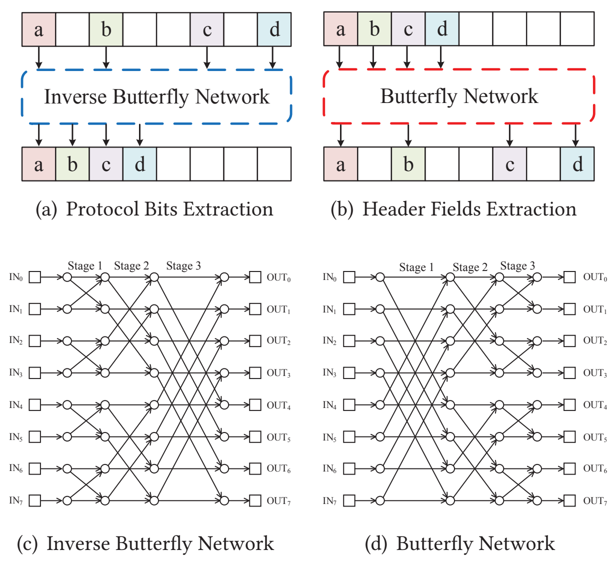

a
b
c
d
Inverse Butterfly Network
Butterfly Network
(a) Protocol Bits Extraction
(b) Header Fields Extraction
Stage 1
Stage 2
Stage 3
IN0 … IN7
OUT0 … OUT7
(c) Inverse Butterfly Network
(d) Butterfly Network

Figure 2: Butterfly Network.

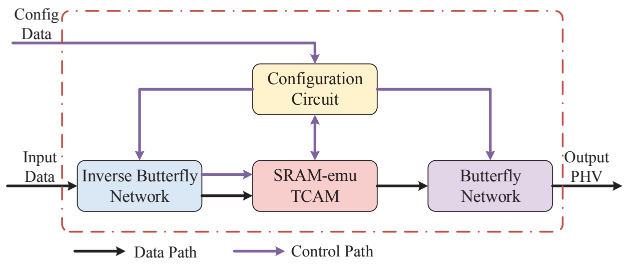


Figure 3: HyperParser System Architecture

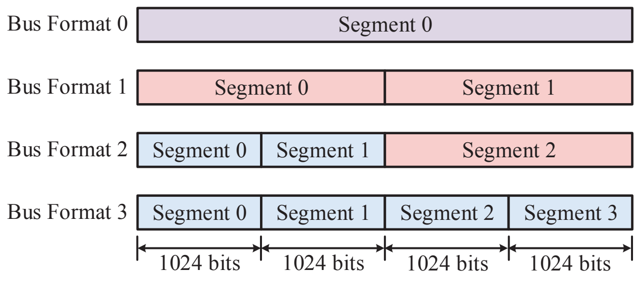


Figure 4: Bus Format

them together efficiently. Meanwhile, in Figure 2(b) and Figure 2(d), the butterfly network can extract the concerned header fields and scatter them to the proper positions in the PHV.

Both butterfly network and inverse butterfly network can achieve $O(nlog(n))$ area complexity, where $log(n)$ is the number of the stages. The inverse butterfly network performs bit-level operations, while the butterfly network performs byte-level operations. Given $n = 4096$, the inverse butterfly network has 12 stages, and the butterfly network has 9 stages. Insertion of registers after every stage can achieve the highest frequency and best performance. However, it can leads to higher latency at the same time. We have made trade-offs. The two networks of the ASIC implementation can achieve 1.45 GHz and 10 clock cycles latency, and the two networks of the FPGA implementation can achieve 600 Mhz and 17 clock cycles latency.

3 HYPERPARSER

3.1 System Architecture

The architecture of HyperParser is concise. As shown in Figure 3, HyperParser is composed of the inverse butterfly network, the SRAM-emulated TCAM, the butterfly network, and the configuration circuit. The inverse butterfly network extracts and gathers the protocol bits, and then transfers them to the SRAM-emulated TCAM. The SRAM-emulated TCAM identifies the protocol type and notifies the butterfly network through the configuration circuit. The butterfly network extracts the concerned fields and scatters them to the proper positions in the PHV. According to the user-defined parse graph, the configuration circuit generates the control signal of the two networks and the initial values of the SRAM-emulated TCAM.

Instead of a TCAM circuit, we adopt an SRAM-emulated TCAM to process the protocol bits. That is because modern FPGAs do not have hard-wired TCAMs. Besides, it is challenging to obtain TCAM IP cores for ASIC implementation. A parallel version of the StrideBV [9] is used here. Compared to the original pipeline version, the parallel version can significantly reduce the latency.

3.2 Bus Format

Wider bus can leads to higher throughput at the cost of lower bus efficiency [26]. For 64-byte Ethernet frames, the 4096-bit bus can only achieve 12.5% bus efficiency. Segmentation can increase the bus efficiency at the cost of a more complex control logic [4, 22]. To achieve the balance point between the throughput and area, the bus format shown in Figure 4 is adopted. The bus can be divided into up to four segments, and there are only four possible bus format types. The worst bus efficiency is 50.2%, where the frame length is 2056 bits (i.e., 257 bytes).

The relation between the throughput and inner bus can be found in Equation 1. We focus on the packet forwarding rate, which is measured in packets per second (pps). $freq$ stands for the frequency of the inner bus. pkt_num stands for the number of packets in one bus word, which equals

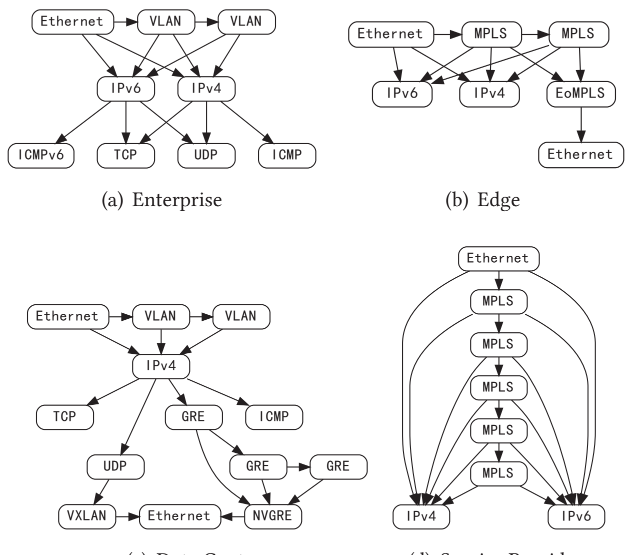


Figure 5: Parse Graph

$\left\lfloor \frac{bus\_width}{frm\_len} \right\rfloor$. bus_width stands for the inner bus width in bytes, and frm_len stands for the frame length in bytes. $freq \times pkt\_num$ stands for the packet forwarding rate of the inner bus. The number 20 stands for the 20-byte Ethernet frame gap. The number 8 stands for the 8 bits in a single byte. Equation 1 stands for the throughput of a single HyperParser instance, which is measured in terabit per second (Tbps). Given the Ethernet link speed, a shorter frame length leads to a higher packet forwarding rate, but a lower throughput for the more shares of the Ethernet frame gaps. An intuitive description of Equation 1 can be found in Figure 8(a), where the blue lines represent the throughput of HyperParser.

$$Throughput = freq \times pkt\_num \times (frm\_len + 20) \times 8 \quad (1)$$

Given the 1.45 GHz frequency, the ASIC implementation of HyperParser can achieve the worst throughput of 3.2 Tbps and the best throughput of 6.8 Tbps, with 257-byte and 128-byte frame lengths. Given the 600 Mhz frequency, the FPGA implementation of HyperParser can achieve the worst throughput of 1.3 Tbps and the best throughput of 2.8 Tbps, where the frame lengths are the same as the ASIC implementation.

4 IMPLEMENTATION

4.1 Parse Graph

HyperParser can support four classic parse graphs [10]. For any of the four graphs, 32 protocol bits are enough for identifying the specific protocol types. According to Figure 4, a single bus word can accommodate up to four packets. Therefore the total number of the protocol bits is 128. There are four TCAMs correspondings to the four bus segments, whose width and depth are 32 and 96. Besides the four classic parse graphs, HyperParser can also support parse graphs in [28][24][4][1] and new parse graphs.

The parse graph can be described by P4, and we developed a simple compiler using Python. After compiling, the initial configuration information of the inverse butterfly network, the SRAM-emulated TCAM, and the butterfly network are generated. HyperParser can support ASIC and FPGA implementations. For the ASIC implementation, the initial configuration values should be converted to the content of the configuration ram, which is based on the two-port Register File. For the FPGA implementation, the initial configuration values should be converted to the content of the LUTs.

4.2 ASIC Implementation

Industry-standard 14nm and 28nm processes are used to evaluate the PPAL of HyperParser. The two instances, we call them HP-14 and HP-28, share the same Verilog code. Both of them work well at 1.45 GHz, and the HP-14 can run up to 2 GHz. The total latency is 17 clock cycles. This work is necessary to prove feasibility in meeting goals such as timing and chip area (cost). We have not produced a complete design or actual silicon.

The two-port Register File is used to realize the configuration ram and TCAM since the Register File is more competitive than the SRAM block when the memory is shallow. The butterfly network and the inverse butterfly network are realized in pipelines. Each stage of the pipeline is composed of parallel multiplexers. Thanks to the low latency of the multiplexer standard cell, the number of registers in the pipeline can be decreased.

4.3 FPGA Implementation

Xilinx 16nm Virtex Ultrascale+ FPGA and 7nm Versal ACAP are used to evaluate the PPAL of HyperParser. They are called HP-U and HP-V respectively. Both of them can run up to 600 Mhz, the total latency is 24 clock cycles. The HP-U has been deployed to the real-world VCU118 evaluation board.

The key point of the FPGA programmability is Look-Up-Table (LUT). The LUT with six inputs and one output is called the LUT6 element. It can be treated as a tiny SRAM with 1-bit width and 64-bit depth. It can also emulate any logic functions with six inputs and one output. Inspired by PR-TCAM [23], we have realized the LUT-oriented implementation of HyperParser. All of the blocks in HyperParser (e.g., butterfly network) are realized in LUTs and DFFs. After the initial synthesis, the locations of the LUTs and the connections

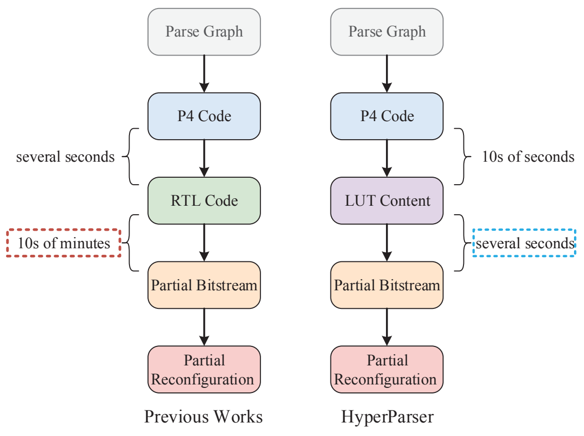


Figure 6: FPGA Configuration Time Comparation

between them do not change anymore. And the contents of the LUTs change with different parse graphs.

The key difference between the LUT-oriented HyperParser and previous FPGA-based parsers lies in the deployment time. As shown in Figure 6, previous works leverage P4 codes as inputs to generate RTL codes, which is fairly fast. But the synthesis tools (e.g., Vivado or Quartus) are needed to convert the RTL codes to the partial bitstream. This procedure may take 10s of minutes. The time is mainly composed of the re-placing and the re-routing of the LUTs.

As for the LUT-oriented implementation of HyperParser, P4 codes can be converted to the LUT contents in 10s of seconds. Thanks to the fixed positions and connections of the LUTs, LUT contents can be converted to the partial bitstream in several seconds, without synthesis tools involved. Potential timing violations can also be avoided. We made use of an open-source tool chain [20] to realize our design.

5 EVALUATION

5.1 Experiment Settings

For the ASIC-based evaluation, industry-standard 14nm and 28nm processes are used, and Synopsys Design Compiler is used for a 1.45 GHz clock cycle target. The PPAL results under 28nm are aimed to make a fair comparison with previous works.

For the FPGA-based evaluation, Xilinx Virtex Ultrascale+ VU9P and Xilinx Versal Prime VM1802 are the target devices. The Vivado Design Suite is used to synthesize the Verilog codes and generate the bitstream. The resource of VU9P is similar to Xilinx SN1000 SmartNIC[4]. We believe that HyperParser can be easily migrated to the SN1000 SmartNIC.

[4]https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html

5.2 PPAL of ASIC Implementation

The PPAL of ASIC implementation is shown in Figure 7. HP-14 stands for the HyperParser under the 14nm process, and HP-28 stands for the HyperParser under the 28nm process. HyperParser outperforms state-of-the-art works in all aspects of PPAL.

Figure 7(a) shows the single instance throughput of proposed works and the state-of-the-art works. HP-14 and HP-28 can achieve the worst-case throughput of 3.2 Tbps, which is 4× of Zolfaghari's work [28].

Figure 7(b) shows the power for 6.4 Tbps throughput. Two instances of HyperParser can achieve 6.4 Tbps, and eight architectures in Zolfaghari's work [28] can achieve the same throughput. HP-14 has a 30.8% lower power than HP-28, and HP-28 performs 6× better than Zolfaghari's work [28].

Figure 7(c) shows the area for 6.4 Tbps throughput. We normalize the area to the number of the gate to enable a fair comparison. The HP-14 and HP-28 consume similar number of gates, which is 43.2% better than Zolfaghari's work [28] and 13× better than Gibb's work [10].

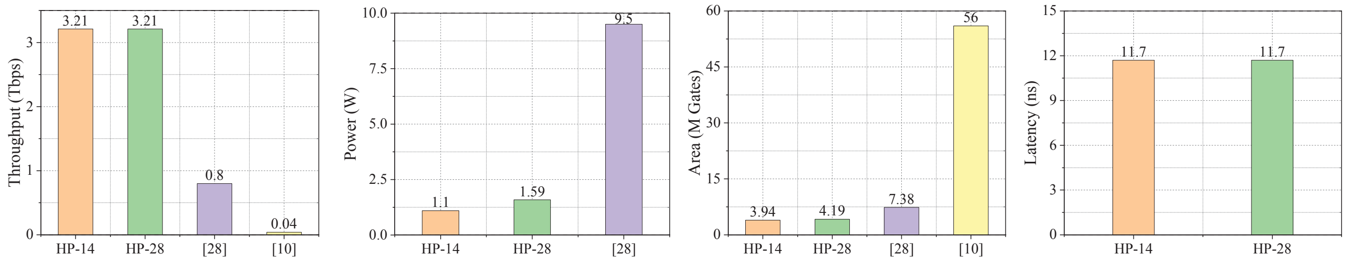
Figure 7(d) shows the latency of HP-14 and HP-28, which is always 11.7ns. HP-14 and HP-28 have the same latency for thier same frequency. Related data is unavailable for Zolfaghari's work [28] and Gibb's work [10].

5.3 PPAL of FPGA Implementation

The PPAL of FPGA implementation is shown in Figure 8. HP-U stands for the HyperParser targets for the VU9P FPGA, and HP-V stands for the HyperParser targets for the VM1802 ACAP. HyperParser outperforms Cabal's work [4] in all aspects of PPAL.

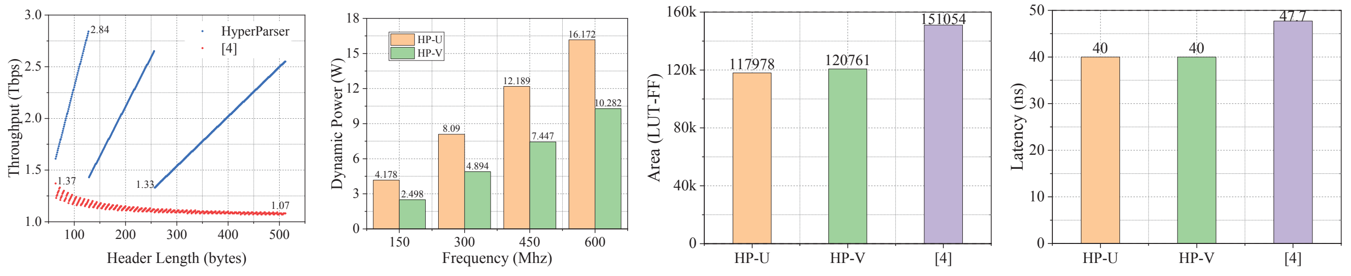
Figure 8(a) shows the relation between the frame length and the throughput. The blue lines stand for HP-U and HP-V, for their maximum frequency are the same 600 Mhz. The throughput of HyperParser meets with Equation 1. The throughput of HyperParser is between 1.33 Tbps and 2.84 Tbps, whereas Cabal's work [4] has a throughput between 1.07 Tbps and 1.37 Tbps. The bus width of Cabal's work [4] is 4096 bits. And the frequency of Cabal's work [4] can be inferred from the throughput, bus width, and bus format. The frequency of Cabal's work [4] is about 255 Mhz. Although the power of Cabal's work [4] is unavailable, we believe that Cabal's work [4] is more power-efficient.

Figure 8(b) shows the dynamic power of a single HP-U and HP-V under various frequencies. We focus on the dynamic power, for the static power is chip-scale, and the HP-U and HP-V only consume a small percentage of the FPGA resources. Dynamic power increases linearly with the frequency for HP-U or HP-V. Moreover, HP-V is 36%-40% more efficient than HP-U in power consumption. It probably due



(a) Performance (b) Power (c) Area (d) Lantency

Figure 7: PPAL for ASIC Implementation



(a) Performance (b) Power (c) Area (d) Lantency

Figure 8: PPAL for FPGA Implementation

to the advanced process (7ns vs. 16nm) and the new FPGA architecture.

Figure 8(c) shows the area of HP-U, HP-V, and Cabal's work [4]. We use LUT-FF pairs for a fair comparison. HP-U and HP-V have similar area consumption (43K LUTs vs. 45K LUTs), and they are 20% more efficient than Cabal's work [4] in area consumption. Besides, the area consumption of HP-U and HP-V do not change with various parse graphs, whereas Cabal's work [4] does.

Figure 8(d) shows the latency of HP-U, HP-V, and Cabal's work [4]. HP-U and HP-V outperform Cabal's work [4] by 19.3%.

6 RELATED WORK

ASIC-based Programmable Parser. Gibb's work [3, 10] is the first streaming programmable packet parser for ASIC implementation. It is based on the FSM model. It can achieve 40 Gbps throughput, but it is hard to achieve the throughput beyond 1 Tbps. Zolfaghari's work [28] is the state-of-the-art programmable packet parser for ASIC implementation. It is based on the pipeline model, and it can achieve 800 Gbps. It may be challenging for Zolfaghari's work [28] to perform a deep parsing, while HyperParser can parse up to 512 bytes. The parser of Tofino switch ASIC [2] can achieve 100 Gbps throughput, and the parser of the E810 Ethernet controller can parse up to 504 bytes.

FPGA-based Programmable Parser. Cabal's work [4] can achieve 1.37 Tbps with a single FPGA. It is based on the pipeline model. The key problem of [1, 4, 15, 24] is the excessive deployment time.

7 CONCLUSION

In this paper, we propose HyperParser, which is a high-performance parser architecture for next-generation programmable switch and SmartNIC. HyperParser is different from the traditional FSM-based and pipeline-based schemes. By using butterfly/inverse butterfly networks, HyperParser can achieve unprecedented throughput, with modest power, area, and latency. HyperParser supports ASIC and FPGA implementations. For the FPGA implementation, HyperParser can significantly decrease the deployment time. We sincerely look forward to sharing our idea with the community, and the source code of HyperParser has been released on Github.

REFERENCES

[1] Pavel Benácek, Viktor Pu, and Hana Kubátová. 2016. P4-to-vhdl: Automatic generation of 100 gbps packet parsers. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 148–155.

[2] Patrick Bosshart. 2018. Programmable Forwarding Planes at Terabit/s Speeds. In *2018 IEEE Hot Chips 30 Symposium (HCS)*. IEEE.

[3] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.

[4] Jakub Cabal, Pavel Benáček, Lukáš Kekely, Michal Kekely, Viktor Puš, and Jan Kořenek. 2018. 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*. IEEE, 249–258.

[5] Adrian Caulfield, Paolo Costa, and Monia Ghobadi. 2018. Beyond SmartNICs: Towards a fully programmable cloud. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 1–6.

[6] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *2011 38th Annual international symposium on computer architecture (ISCA)*. IEEE, 365–376.

[7] Saeed Fathololoumi, David Hui, Susheel Jadhav, Jian Chen, Kimchau Nguyen, MN Sakib, Z Li, Hari Mahalingam, Siamak Amiralizadeh, Nelson N Tang, et al. 2021. 1.6 Tbps Silicon Photonics Integrated Circuit and 800 Gbps Photonic Engine for Switch Co-Packaging Demonstration. *Journal of Lightwave Technology* 39, 4 (2021), 1155–1161.

[8] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: Smartnics in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX Association, 51–66.

[9] Thilan Ganegedara and Viktor K Prasanna. 2012. StrideBV: Single chip 400G+ packet classification. In *2012 IEEE 13th International Conference on High Performance Switching and Routing*. IEEE, 1–6.

[10] Glen Gibb, George Varghese, Mark Horowitz, and Nick McKeown. 2013. Design principles for packet parsers. In *Architectures for Networking and Communications Systems*. IEEE, 13–24.

[11] Tao Gui, Xuefeng Wang, Ming Tang, Yi Yu, Yanzhao Lu, and Liangchuan Li. 2021. Real-Time Demonstration of Homodyne Coherent Bidirectional Transmission for Next-Generation Data Center Interconnects. *Journal of Lightwave Technology* 39, 4 (2021), 1231–1238.

[12] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.

[13] Yedidya Hilewitz. 2008. *Advanced bit manipulation instructions: architecture, implementation and applications*. Princeton University.

[14] Yedidya Hilewitz and Ruby B Lee. 2008. Fast bit gather, bit scatter and bit permutation instructions for commodity microprocessors. *Journal of Signal Processing Systems* 53, 1 (2008), 145–169.

[15] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The p4-> netfpga workflow for line-rate packet processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM Press, 1–9.

[16] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 121–136.

[17] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. 2020. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 514–528.

[18] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*. 44–58.

[19] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259.

[20] Huan Liu, Zhiliang Qiu, Weitao Pan, Jun Li, Ling Zheng, and Ya Gao. 2020. Low-Cost and Programmable CRC Implementation based on FPGA. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 1 (2020), 211–215.

[21] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 15–28.

[22] Péter Orosz, Tamás Tóthfalusi, and Pál Varga. 2018. FPGA-assisted DPI systems: 100 Gbit/s and beyond. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 2015–2040.

[23] Pedro Reviriego, Anees Ullah, and Salvatore Pontarelli. 2019. PR-TCAM: Efficient TCAM emulation on xilinx FPGAs using partial reconfiguration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 8 (2019), 1952–1956.

[24] Jeferson Santiago da Silva, François-Raymond Boyer, and JM Pierre Langlois. 2018. P4-compatible high-level synthesis of low latency 100 Gb/s streaming packet parsers in FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 147–152.

[25] Anirudh Sivaraman, Thomas Mason, Aurojit Panda, Ravi Netravali, and Sai Anirudh Kondaveeti. 2020. Network architecture in the age of programmability. *ACM SIGCOMM Computer Communication Review* 50, 1 (2020), 38–44.

[26] Noa Zilberman, Gabi Bracha, and Golan Schzukin. 2019. Stardust: Divide and Conquer in the Data Center Network. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 141–160.

[27] Hesam Zolfaghari. 2020. Flexible Low-Area Hardware Architectures for Packet Processing in Software-Defined Networks. (2020).

[28] Hesam Zolfaghari, Davide Rossi, Walter Cerroni, Hayate Okuhara, Carla Raffaelli, and Jari Nurmi. 2020. Flexible software-defined packet processing using low-area hardware. *IEEE Access* 8 (2020), 98929–98945.