Design and Implementation of an FPGA-based 10 Gigabit Ethernet UDP/IP Hardware Protocol Stack Postprint

Authors: Dong Yongji, Wang Yu, Yuan Zheng

Date: 2022-04-07T15:01:56Z

Abstract

To address the inability of traditional software-based protocol stacks to meet the demands of high-speed data transmission and processing, a hardware-accelerated UDP protocol stack design scheme is proposed. Based on the highly efficient parallel characteristics of hardware, this scheme implements a UDP/IP protocol stack that satisfies the high-bandwidth requirements of 10 Gigabit Ethernet data transmission. Actual testing demonstrates that the design can achieve a maximum transmission rate of 9.32 Gbps, meeting the requirements for line-rate processing under 10 Gbps bandwidth. Compared with traditional software implementations, the processing capability is closer to the theoretical limit.

Full Text

Preamble

Vol. 39 No. 8
Application Research of Computers

Design and Implementation of a 10G Ethernet UDP/IP Hardware Protocol Stack Based on FPGA

Dong Yongji, Wang Yu, Yuan Zheng
(PLA Strategic Support Force Information Engineering University, Zhengzhou 450002, China)

Abstract: To address the limitations of traditional software-based protocol stacks in meeting high-speed data transmission requirements, this paper proposes a hardware-accelerated UDP protocol stack design. Leveraging the efficient parallel processing capabilities of hardware, this implementation realizes

a UDP/IP protocol stack that satisfies the high-bandwidth demands of 10G Ethernet. Experimental tests demonstrate that the design achieves a maximum transmission rate of 9.32 Gbps, meeting the requirements for line-rate processing at 10Gbps bandwidth. Compared with conventional software implementations, the proposed approach delivers processing capacity much closer to the theoretical limit.

Keywords: FPGA; 10G Ethernet; hardware protocol stack; UDP protocol

0 Introduction

With the rapid development of network technology and the exponential growth of bandwidth, massive volumes of video, image, and other business data have become the primary traffic over high-speed links. Faced with the transmission pressure of such enormous data volumes, traditional operating system built-in protocol stacks or software-centric acceleration techniques can no longer satisfy the demands for high-speed, low-latency transmission.

Based on network processing experience, 1Gbps Ethernet transmission requires approximately 1GHz of processor frequency. In the face of 10G high-speed network transmission requirements, substantial CPU computational resources are consumed by network protocol processing. Hardware-based UDP/IP protocol stacks, which harden protocol processing logic into hardware, can provide high throughput, low latency, and high-bandwidth transmission performance.

Given that FPGAs combine flexible configurability with high-speed parallel processing capabilities, they are particularly suitable for hardening protocol stacks in high-speed transmission domains to enhance system processing capacity. Xiong Guangyang et al. proposed an FPGA-based implementation of MAC and UDP/IP protocol stacks to address high-speed transmission in gigabit Ethernet environments. Cui He et al. utilized hardware to implement UDP/IP protocol data encapsulation and decapsulation. Guixé Orriols G developed a general-purpose gigabit UDP/IP protocol core for DAQ applications based on Intel FPGA. Feng Chen designed an efficient gigabit protocol stack for high-performance protocol processing requirements. Nianyun Liu et al. proposed a UDP/IP protocol stack for large-scale sensor networks, achieving an 8x performance improvement over software processing. For 10G application scenarios, Wang Yuheng designed a 10G Ethernet UDP/IP processor for video transmission interfaces; Xia Yang designed a 10G Ethernet data distribution platform to meet UDP transmission requirements; Peng Peng implemented a high-speed protocol processing module for low-latency requirements in nuclear physics applications. Commercial vendors such as PLDA, Fraunhofer HHI, Intilop, and Dini Group have also proposed expensive commercial IP cores.

In summary, current hardware protocol stack research primarily focuses on gigabit Ethernet environments, while 10G Ethernet hardware protocol stacks rep-

resent a current application hotspot. The UDP/IP protocol stack implemented in this paper based on FPGA is designed for 10G network environments and offers the following advantages: (1) It integrates protocol frame parsing, processing, transmission, and reception into a single system, significantly improving UDP application transmission efficiency. (2) Based on the reconfigurable and configurable characteristics of FPGA, it can be deployed in various network scenarios. (3) Designed for server applications, it supports concurrent connections for multi-port, multi-point services.

1 Hardware Architecture Design

To improve UDP protocol transmission efficiency, this paper proposes an FPGA-based UDP protocol stack design implemented on Stanford University' s NetFPGA-10G board to maximize support for various applications. This solution leverages the board' s PCI Express interface, which offers strong portability and extensibility, facilitating deployment of the hardware protocol stack into server systems through physical interface expansion for accelerating UDP-based applications.

The FPGA represents the key and most challenging component of the hardware subsystem design. Based on FPGA hardware architecture characteristics, protocol processing functions are implemented, with the hardware portion divided into the following modules: 10G MAC interface module, protocol parsing module, ARP response module, ICMP response module, protocol encapsulation module, UDP protocol processing module, and PCIE-DMA module. The module partition structure is shown in Figure 1.

1) Protocol Parsing Module
This module primarily implements protocol parsing processing for incoming data packets and extracts relevant user information for subsequent response packet generation and encapsulation. Initially, the module performs layer-by-layer parsing of arriving packet protocols, ultimately achieving three functions: parsing UDP protocol request packets supported locally, extracting local protocol packets, and filtering irrelevant packets. Given the special nature of FPGA hardware architecture, which cannot completely implement parsing and processing for all protocols, the specific processing algorithm for the designed application scenario is implemented as shown in Figure 2.

The detailed processing algorithm steps are as follows:

- **Step 1:** Determine whether the Ethernet type of the incoming frame is 0x0800. If it is 0x0800 (IPv4 protocol), proceed to Step 2; otherwise, proceed to Step 7.
- **Step 2:** Determine whether the destination IP field in the IPv4 packet matches one of the locally configured service IPs. If a match is found, proceed to Step 3; otherwise, discard the packet.
- **Step 3:** Determine whether the protocol field in the IPv4 header is ICMP. If it is ICMP, proceed to Step 4; otherwise, proceed to Step 6.

- **Step 4:** Extract the source IP address and ICMP protocol type and code from the incoming ICMP packet for subsequent ICMP response generation, then terminate protocol parsing.
- **Step 5:** Determine whether the protocol field in the IPv4 header is UDP. If it is UDP, proceed to Step 6; otherwise, discard the packet.
- **Step 6:** Extract the source IP address from the incoming UDP packet for subsequent UDP response encapsulation, then terminate protocol parsing.
- **Step 7:** Determine whether the Ethernet type of the incoming frame is 0x0806. If it is 0x0806 (ARP protocol), proceed to Step 8; otherwise, discard the packet.
- **Step 8:** Determine whether the destination IP field in the ARP packet matches one of the locally configured service IPs. If a match is found, proceed to Step 9; otherwise, discard the packet.
- **Step 9:** Determine whether the destination MAC address is all Fs (broadcast address). If it is broadcast, proceed to Step 10; otherwise, proceed to Step 12.
- **Step 10:** Determine whether the destination MAC address is the local MAC address. If it is the local MAC, proceed to Step 11; otherwise, discard the packet.
- **Step 11:** Extract the source MAC and source IP addresses from the incoming ARP packet to build the ARP cache table and for subsequent ARP response packet generation, then terminate protocol parsing.
- **Step 12:** Extract the source MAC and source IP addresses from the incoming ARP packet to build the ARP cache table and for subsequent ARP response generation, then terminate protocol parsing.

2) ARP Response Module

This module implements ARP request and response packet construction and ARP cache table maintenance. When building this module based on FPGA, on-chip BLOCK RAM is used to store ARP entries. Due to limited hardware resources, the module adds a hardware resource constraint to standard ARP protocol processing: when the BLOCK RAM storing ARP entries is about to overflow, it selectively deletes the entry with the longest aging time to maintain normal ARP table operation.

The module first constructs ARP response packets based on ARP request information; second, it responds to protocol encapsulation module requests by providing IP-MAC mapping services and periodically refreshes the ARP cache table by deleting expired entries based on aging time; third, it actively organizes ARP request packet content when needed for IP-MAC mapping relationships not present in the cache table. The ARP cache table maintenance algorithm flow is shown in Figure 3.

The detailed processing steps are as follows:

- **Step 1:** Match the destination IP address of the packet to be sent against entries in the cache table space. If matched, proceed to Step 2; otherwise, proceed to Step 3.

- **Step 2:** Check whether all entries in the table space have been used (whether the table space is full). If full, proceed to Step 6; otherwise, proceed to Step 7.
- **Step 3:** Compare the timestamp stored in the cache table with the current time to determine whether the entry to be matched has timed out. If timed out, proceed to Step 4; otherwise, proceed to Step 5.
- **Step 4:** Delete the timed-out cache entry information and generate corresponding ARP request packet content.
- **Step 5:** Update the timestamp information of the cache entry.
- **Step 6:** Delete the entry with the longest aging time from the cache table space.
- **Step 7:** Create a new cache entry at an unoccupied address in the table space.

3) ICMP Response Module

This module implements a subset of the ICMP protocol. Upon receiving ICMP request packets, it generates corresponding response packets according to ICMP protocol specifications based on the current system state. Since the UDP protocol stack designed in this paper is targeted for server deployment, it implements three response functions according to server terminal protocol characteristics: "Echo Reply," "Time Exceeded," and "Destination Unreachable." "Echo Reply" supports client PING operations, "Time Exceeded" handles TTL expiration in data segments, and "Destination Unreachable" indicates unopened server ports.

4) Protocol Encapsulation Module

This module primarily implements two functions: first, protocol encapsulation of UDP datagrams; second, data convergence of encapsulated UDP protocol frames, ICMP protocol frames, and ARP protocol frames for unified transmission to the 10G MAC interface. When encapsulating UDP packets with IP protocol, the module needs to match and look up entries in the ARP cache table of the ARP response module based on user IP address (destination IP address) information to obtain the corresponding MAC address, then encapsulate the IP packet into a data frame. The structure of the protocol encapsulation module is shown in Figure 4.

5) UDP Protocol Processing Module

This module primarily implements UDP protocol encapsulation and decapsulation, UDP checksum calculation, and socket interfaces and data channels with application software. The interface between the UDP protocol processing module and application software is shown in Figure 5.

Software applications run on the server side. Before providing services, software must apply for/release service ports through configuration ports to the hardware protocol stack. The hardware module decides whether to provide or deny services based on the service port status.

When the module receives data from the line, if the destination port is open, the hardware performs UDP protocol decapsulation, extracts the communication

socket and received data; if the destination port is closed, the hardware discards the UDP data segment.

When the module receives data from software, if the source port is open, the hardware encapsulates the application data into UDP data segments based on the socket provided by software; if the source port is closed, the hardware discards the application data.

To manage numerous UDP port states in server scenarios and support concurrent connections for multi-port, multi-point services, this module uses dual-port BRAM to establish a port status table with 16-bit address width. The software system sets the corresponding port number in the lookup table to open/closed through apply/release operations. The status table has 2^16 addresses, enabling one-to-one mapping of all UDP port numbers, with each entry containing a 1-bit indicator to flag whether the port is open. As shown in Figure 6, port 53 (DNS protocol) is open. Combined with time and event trigger mechanisms to poll the entire status table, the module manages the entire lifecycle of UDP ports. By building such a port status lookup table using FPGA on-chip resources, the design supports parallel lookup of multi-port states, enabling coexistence of multi-port, multi-service operations. Leveraging the operational characteristics of dual-port RAM also effectively improves state lookup rates for multi-port, multi-service scenarios.

6) PCIe-DMA Module

The PCIe-DMA module primarily includes the Xilinx PCIe IP core and DMA module. The PCIe IP core is directly instantiated in Vivado to implement the physical and data link layers of the PCIe protocol. As shown in Figure 5, the PCIe-DMA module primarily enables transparent communication between the UDP protocol processing module and host computer software.

2 Design Verification

The verification of this design is divided into functional simulation verification and physical performance verification. The hardware platform is based on Stanford University' s NetFPGA-10G board, a 4-port 10Gb/s PCI Express adapter card featuring a Virtex-5 XC5VTX240T FPGA with 18,720 configurable logic blocks, 2,400 Kbit distributed RAM, and 11,664 Kbit (324x36K) block RAMs. The entire system is developed in Verilog using Xilinx Vivado 2016.4 and simulated with Xilinx ISIM. Simulation results demonstrate that the protocol stack design correctly completes sending and receiving of ARP, ICMP, IPv4, UDP, and other protocol packets. As shown in Figure 7, the protocol stack achieves line-rate data reception and transmission at 10G rates.

For performance verification of the proposed design, Spirent TestCenter 3.61 with two 10G interfaces was used for actual packet transmission testing. By gradually increasing the test packet MTU length, the performance of this system and the native Linux UDP protocol stack under different packet sizes was evaluated. The test results are shown in Figure 8.

The results in Figure 8 indicate that as the packet MTU value increases, the hardware protocol stack' s UDP payload transmission capacity approaches the theoretical value more closely, while the software-implemented UDP protocol stack, limited by the bottleneck of processor serial processing, shows an increasingly significant performance gap with this solution. When the MTU reaches 1,400 bytes, the protocol stack achieves a UDP data transmission rate of 9.32 Gbps, approaching the theoretical value of 9.7 Gbps, meeting design requirements and satisfying the need for line-rate data transmission on 10G links.

3 Conclusion

Traditional FPGA-based UDP/IP protocol stack research has focused on one-to-one point-to-point transmission, which cannot be applied to server scenarios requiring multi-point concurrent connections in 10G network environments. This paper studies a UDP hardware protocol stack oriented toward server application scenarios. The design fully leverages hardware advantages in parallel processing, employs pipelined design for UDP protocol processing flow, and hardens UDP protocol processing in FPGA, reducing the processing demands on the processor for transport layer protocols and improving system transmission efficiency. Experimental results demonstrate that the protocol stack can handle 10G network UDP protocol processing at line rate. Future work will focus on researching and discussing extremely high-rate services in 5G application scenarios and exploring integration with smart NIC directions.

References

[1] Ke Yang. Design and development of high-speed data transmission board based on FPGA [D]. Wuhan: Central China Normal University, 2020.
[2] Xiong Guangyang, Wang Ye, Li Zhiru, et al. Implementation of gigabit UDP/IP protocol stack based on FPGA and its application in high-speed image transmission [J]. Instrumentation Users, 2020, 27(03): 69-72.
[3] Cui He, Liu Yunqing, Sheng Jiajin. Research and implementation of UDP/IP protocol stack based on FPGA [J]. Journal of Changchun University of Science and Technology (Natural Science Edition), 2014(2): 133-137.
[4] Guixé Orriols G. Design and implementation of an UDP/IP Ethernet hardware protocol stack for FPGA based Systems [D]. Barcelona: Universitat Politècnica de Catalunya, 2019.
[5] Feng Chen. Research on high-performance protocol processing engine technology based on SAR [D]. Hangzhou: Zhejiang University, 2021.
[6] Nianyun Liu, Zhiqiang Xu. The Design of High-Speed Hardware UDP/IP Stack Based on FPGA for Large-Scale Sensing Systems. Journal of Internet Technology, 2017, 18(3): 579-587.
[7] Wang Yuheng. Design of 10G Ethernet UDP/IP processor video transmission interface based on FPGA [D]. Shenyang: Shenyang University of Technology, 2018.
[8] Xia Yang. Design of 10G Ethernet data distribution platform based on

FPGA [D]. Beijing: Beijing Institute of Technology, 2016.
[9] Peng Peng. Research on high-speed data transmission system for nuclear physics experiments based on 10G Ethernet [D]. Lanzhou: Northwest Normal University, 2021.
[10] https://www.plda.com/products/fpga-ip/xilinx/fpga-ip-tcpip/quicktcp-xilinx/.
[11] Langenbach U, Berthe A, Traskov B, et al. A 10 GbE TCP/IP hardware stack as part of a protocol acceleration platform [C]// Proc of 2013 IEEE Third International Conference on Consumer Electronics. Berlin (ICCE-Berlin). IEEE, 2013: 381-384.
[12] http://www.intilop.com/tcpipengines.php/.
[13] http://www.dinigroup.com/new/TOE.php/.
[14] Liu Yuhan. Research on network packet encryption implementation and low power consumption based on NetFPGA10G [D]. Hangzhou: Hangzhou Dianzi University, 2018.

Note: Figure translations are in progress. See original paper for figures.