FastProxy: Hardware and Software Acceleration of Stratum Mining Proxy

Guanwen Zhong*, Haris Javaid*, Hassaan Saadat‡, Lingchao Xu§, Chengchen Hu* and Gordon Brebner†
*Xilinx, Singapore ‡University of New South Wales, Australia §Bitmaintech Pte. Ltd., Singapore †Xilinx, USA
{*henryz, harisj, chengchen, gjb*}*@xilinx.com*, *h.saadat@unsw.edu.au*, *lingchao.xu@bitmain.com*

***Abstract*—The Stratum protocol is the *de facto* protocol for mining proxies in proof-of-work (PoW) based cryptocurrencies such as Bitcoin. A Stratum mining proxy connects to an upstream mining pool server and to downstream miners through TCP/IP connections. The proxy receives periodic jobs from the pool and broadcasts them to the miners. The broadcast operation becomes a performance and scalability bottleneck when the number of miners increases significantly. In this paper, we propose a hardware/software co-designed architecture for the proxy to accelerate the broadcast of periodic jobs. We customize the Stratum protocol with a layer 2 broadcast mechanism instead of using TCP/IP connections. The proposed architecture is implemented on a Xilinx Zynq SoC (ARM processor and FPGA) board where the layer 2 broadcast mechanism is offloaded on the FPGA. Our experiments demonstrated a speedup of 2079× in transmission time with 225 miners connected to the proxy, compared to an implementation on an Intel i7 server.**

I. Introduction

Over the last few years, blockchain technology has gained huge momentum with applications in both financial and non-financial domains. Blockchain is essentially a distributed ledger of transactions, where a transaction represents execution of some business logic agreed upon by the participating parties. One of the most successful examples is the Bitcoin cryptocurrency (highest market cap [1]), where transactions represent creation or transfer of digital currency.

The Bitcoin network, as for many other cryptocurrencies, is a peer-to-peer network of nodes as shown in Figure 1. The nodes make up the full-node network and are responsible for keeping the entire ledger. A node receives new blocks from its neighboring nodes and updates its own copy of the ledger. Furthermore, it receives unconfirmed transactions that should be included in future blocks. A mining pool server primarily acts as the interface between the nodes and the miners. It periodically fetches unconfirmed transactions from the nodes to create a new block, which is then sent to the miners as a job. The miners execute a proof-of-work algorithm, and send back valid *nonce* values (solutions to the mining problem). The mining pool server uses one of the *nonce* values, that satisfies the network difficulty, to create the newly-mined block and sends it to one of the full nodes for propagation. Furthermore, it manages the mining reward for the miners. For scalability reasons, a mining proxy is used to manage the miners locally while acting as a super miner to the mining pool server.

Motivation: The Stratum protocol [11] is the *de facto* protocol for mining proxies in most successful proof-of-work cryptocurrencies such as Bitcoin, Ethereum, etc. [1][7]. A typical Bitmain set-up involves tens of thousands of miners (e.g. 20,000) connected through switches to a mining proxy. The mining proxy will periodically (every 1 minute) receive a mining job from the mining pool server. Since the job for all the miners is the same, the mining proxy will broadcast

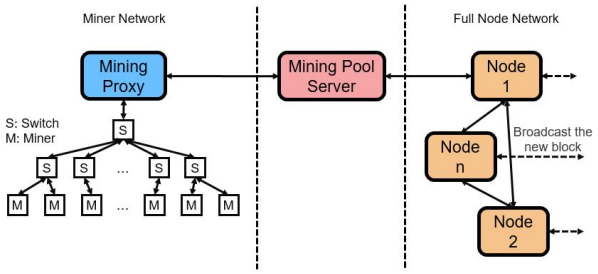


Fig. 1: A Typical Bitcoin Network

that job to all the miners. From an implementation point of view, the mining proxy will keep 20,000 TCP/IP connections with the miners, and will send the same job one by one. There are two problems: (1) This becomes a performance bottleneck, for example, it could take up to a few seconds to send the job to all the miners, and (2) This is not scalable as there is a single point of connection for all the miners, which limits the maximum number of miners that can be connected.

In proof-of-work cryptocurrencies, most of the acceleration efforts are focused on miners, and are dominated by ASIC implementations. The network-dominated parts are typically ignored, which can become a bottleneck and hamper scalability. In this paper, we focus on software customization and hardware acceleration of the mining proxy (to further improve its performance and scalability) because current approaches tend to deploy multiple mining proxies, which is not cost-effective and involves high maintenance effort.

Contributions: In this work, we make the following two observations: (1) The miners are connected to the mining proxy through a few LAN switches, that is, they are physically co-located, and (2) The job sent by the mining proxy to the miners is exactly the same. We exploit these observations to customize the Stratum protocol to use layer 2 broadcast mechanism (broadcast over LAN) for periodic communication between the mining proxy and the miners instead of using TCP/IP connections. Furthermore, we propose a hardware/software co-designed system where the layer 2 broadcast is offloaded on an FPGA. The proposed hardware directly parses each incoming job from the Ethernet interface, and sends it out as a layer 2 broadcast to all the miners. As a result, the entire kernel network stack is bypassed and, given that FPGAs can perform fast packet processing, our hardware/software co-design approach can achieve much better performance and scalability. When implemented on a Zynq SoC with an ARM processor and FPGA, our system can achieve up to 2079× improvement in transmission time compared to an implementation on an Intel i7 server with 225 connected miners. This demonstrates that a Stratum mining proxy can be run on a low-cost standalone FPGA board instead of an Intel NUC (Next Unit of Computing), which is a more costly current *de facto* implementation platform within Bitmain.

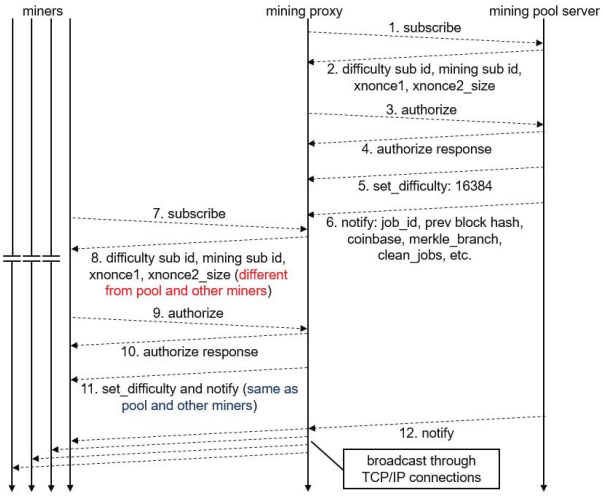


Fig. 2: Original Stratum Proxy Workflow

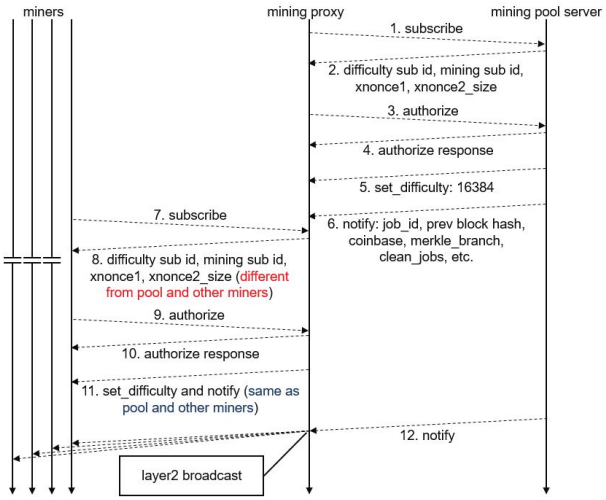


Fig. 3: Customized Stratum Proxy Workflow

II. Customized Stratum Protocol

Figure 2 shows the typical workflow of the mining proxy. At the start (steps 1 and 2), the mining proxy sends a *subscribe* request to the mining pool server. In return, it receives two ids (one for the difficulty subscription and the other for the mining subscription) and *extranonce* settings. In steps 3 and 4, the mining proxy authorizes itself with the mining pool server so it can later submit its solution to the mining problem in exchange for reward shares. The mining pool server starts sending both *set_difficulty* and *notify* jobs to the mining proxy (steps 5 and 6). When a new miner connects to the mining proxy (steps 7 and 8) through a *subscribe* request, the mining proxy returns subscription ids generated by itself instead of the ones received from the mining pool server. The mining parameters returned by the mining proxy are unique across the miners, that is, each miner has its own mining parameters. Just like the mining proxy, a miner also authorizes itself with the mining proxy.

In a steady-state scenario, the mining proxy broadcasts the incoming jobs from the mining pool server to the miners. There are two types of jobs: (1) The *set_difficulty* job which directs a miner to use the supplied difficulty when solving the mining problem, and (2) The *notify* job which supplies the job id, list of Merkle branches, and other information to the miner to perform the mining operation. Although the miners get the same job, every miner performs a different mining operation based on its mining parameters. In this figure, the mining proxy sends the job to all the miners one by one through the TCP/IP connections (step 12).

We customize the mining proxy protocol (as shown in Figure 3) such that after it receives a new job from the mining pool, it creates a layer 2 broadcast packet with the payload containing the details of the job. This layer 2 packet is then sent out, which means all the miners connected through the LAN switches will receive this packet, and hence the new job from the proxy. This has two advantages: (1) The entire TCP/IP kernel stack is bypassed which means transmissions are much faster; and (2) The sending latency is independent of the number of connected miners, which means the mining proxy is more scalable and can handle many tens of thousands of miners. The layer 2 broadcast operation can be implemented in either software or hardware. For software implementation, the kernel TCP/IP stack is used to receive the job. Then, a layer 2 broadcast packet with the job details is created and a raw socket over the Ethernet interface is used to send out that packet from the mining proxy software. The hardware implementation of our broadcast operation is described in the next section.

III. HW/SW Co-Designed Mining Proxy

The mining proxy consists of non-communication-intensive functions and communication-intensive tasks. The non-communication-intensive functions include connection establishment, *subscribe*, *authorize*, etc., as mentioned in Section II, while the communication-intensive tasks contain *notify* and *set_difficulty* broadcasts. Thus, the mining proxy can exploit a heterogeneous architecture by mapping non-communication-intensive functions on to CPUs and offloading communication-intensive tasks on to FPGAs.

In this work, we leverage Xilinx Zynq [13], a heterogeneous platform, to design an efficient and scalable mining proxy. The Zynq architecture provides the software programmability of ARM processors and the hardware programmability of an FPGA on a single chip. The ARM processors and FPGA are connected by Advanced eXtensible Interface (AXI) interconnection and share access to memory and other common peripherals such as UART, USB and Ethernet. The hardware accelerators for layer 2 broadcasting are designed on the FPGA logic and attached to AXI interconnect, while the software part runs on the ARM processor.

Overview: The system architecture of the proposed mining proxy acceleration is shown in Fig. 4. At a high level, the packets sent from the ARM go through the network without any interruption from our broadcast mechanism. Likewise, all the packets received from the network go to the ARM uninterrupted, except for the *notify* and *set_difficulty* packets. The following paragraph briefly explains how packets flow through the proposed system.

The *notify* or *set_difficulty* packets only come from upstream mining pools connected via *Ethernet*. Thus, for any packet sent from the ARM (Path ①), it will be redirected to the *Dispatcher* (Path ⑦) via the *Arbiter*. Then, the *Dispatcher* submits the packet to the TX path (Path ⑤) of the *Ethernet* subsystem. In this case, the FPGA logic just bypasses the packet without modification. Any packet coming from the network via the RX path of the *Ethernet* subsystem (Path ⑥) will also be sent to the *Dispatcher* (Path ⑦) and later submitted to the ARM (Path ⑧). At the same time, the Sniffer sniffs

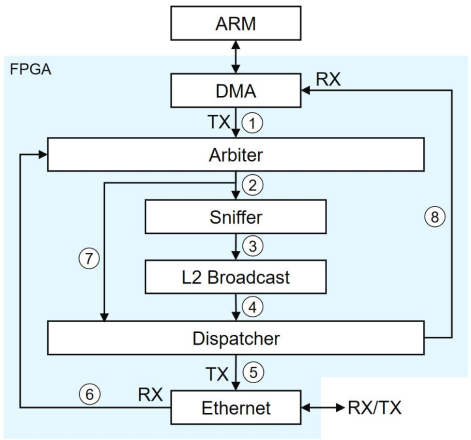


Fig. 4: System Architecture for Mining Proxy.

the packet (Path ②) and checks whether it is a *notify* or *set_difficulty* packet. If the packet contains neither *notify* nor *set_difficulty* data, the *Sniffer* will simply drop the packet. Otherwise, it extracts the payload of the packet and sends it to the *L2 Broadcast* to construct a layer 2 broadcast packet (Path ③). The newly-generated packet will then be submitted to the *Dispatcher* (Path ④) and sent out to the network through the *Ethernet* subsystem (Path ⑤).

Arbiter: It has two input channels (FIFOs) to buffer packets coming from both the ARM (Path ①) and the network (Path ⑥). Its output channel will periodically select packets from the input channels in a round-robin fashion.

Sniffer: It sniffs any packet coming from the network (Path ⑥→②) and extracts its payload if it is a *notify* or *set_difficulty* packet from a mining pool. The *Sniffer* has a finite state machine (FSM) to first check whether the packet has valid Ethernet, IP and TCP headers. If any invalid header is detected, it simply drops the packet. For the valid packet, the *Sniffer* then parses the payload of the packet (format shown in Fig. 5) and searches the keywords, *mining.notify* and *mining.set_difficulty*, to identify whether it is a *notify* or *set_difficulty* packet. Once detected, the *Sniffer* will buffer the payload and signal the *L2 Broadcast* to construct a broadcast packet. Any packet apart from *notify* and *set_difficulty* is discarded.

```
/*Job notify format */
{"method": "mining.notify", "id": null, "params": ["job_id","merkle branches", ...]}
/*Job set_difficulty format */
{"method": "set_difficulty", "id": null, "params": [difficulty value]}
```

Fig. 5: Payload Format

L2 Broadcast: The component is used to generate an Ethernet broadcast frame. It first checks whether the payload FIFO in the *Sniffer* has valid data. If the FIFO is empty, it will stay idle. Otherwise, it will construct an Ethernet frame header by inserting the broadcast address (*FF:FF:FF:FF:FF:FF*) in the destination MAC address field. The data is then extracted from the payload FIFO and appended to the generated frame header to form a complete Ethernet broadcast frame. The frame packet will be sent later to the network via the *Dispatcher* and the *Ethernet*.

Dispatcher: It has two input and two output channels (FIFOs). It works similarly to a switch and redirects input data to a specific output channel. For data coming from the ARM (the TX path ①→⑦) or the *L2 Broadcast* (Path ④), it will be sent to the TX path of the Ethernet (Path ⑤). For data received from the network (Path ⑥→⑦), it will be redirected to the RX path of the ARM (Path ⑧).

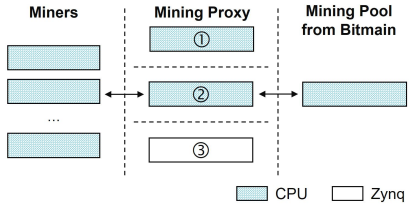


Fig. 6: Experimental Set-up

IV. Experimental Setup and Results

We have three different mining proxy set-ups for comparison, as shown in Fig. 6:

- ① *Original Proxy*: runs on a CPU with TCP/IP broadcast.
- ② *SW-L2 Proxy*: runs on a CPU with layer 2 broadcast.
- ③ *Zynq Proxy*: uses new hw/sw co-designed architecture.

Each proxy was connected to the same Bitmain mining pool, AntPool [3], via 1 Gbps Ethernet. We leverage CPU-based miners [12] and run up to 225 miners per proxy connection. For the ② and ③ setups, the miners were modified to get raw socket packets directly from the Ethernet interface.

The proxies in ① and ② ran on workstations with Intel i7 at 2.1GHz, while servers with Intel Xeon 4416 CPUs at 2.1GHz were used for the miners. The *Zynq Proxy* is built on an ONetSwitch45 board [4][10] featuring a Xilinx XC7Z045 Zynq SoC [13] as shown in Fig. 7a. The ARM core runs at 800MHz, while the FPGA accelerator works at 125MHz. The accelerator was synthesized and implemented with Vivado (v2018.2). The Linux kernel running on the ARM was generated by Petalinux (v2018.2).

A. Performance Improvement

Fig. 7b shows the real-time execution of the three proxy setups over time. The x-axis represents the i^{th} job sent every minute from the connected mining pool, while the y-axis (plotted on a logarithmic scale) denotes the total transmission time for broadcasting the i^{th} job to m miners. We start with $m = 50$ miners and increase the number up to 225. For the *Original Proxy* design, its transmission time (the red line with dots in Fig. 7b) increases linearly when changing the number of connected miners. The reason is that the *Original Proxy* sends the job to all the miners one-by-one via TCP/IP. Table I shows that the total transmission time for the *Original Proxy* to deliver a job to 225 miners is about 6030 μs.

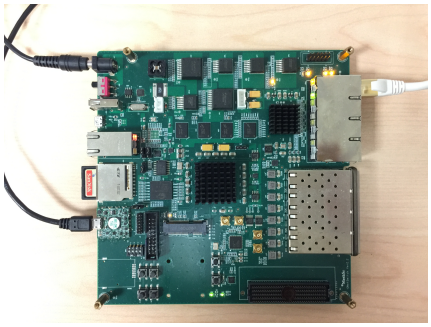
TABLE I: Mining Proxy Speedup

225 Miners	Performance	
	CPU	Zynq
TCP/IP Broadcast	① 6030 μs (26.8 μs/miner)	–
Layer 2 Broadcast	② 51 μs	③ 2.9 μs
Speed-up	∼118x	∼2079x

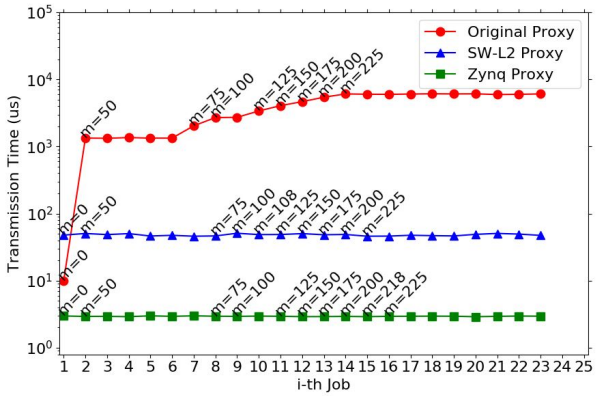
① *Original Proxy*; ② *SW-L2 Proxy*; ③ *Zynq Proxy*

When leveraging layer 2 broadcast, the *SW-L2 Proxy* only needs to send one job request to connected switches. The switches then dispatch the job to all the connected miners. This significantly reduces the transmission time (the blue line with triangles) compared with the *Original Proxy*. Moreover, the transmission time of the *SW-L2 Proxy* is independent of the number of miners connected. As reported in Table I, it takes 51 μs to inform 225 miners of a new job, which is 118× faster than the time taken by the *Original Proxy* design.

Compared to *SW-L2 Proxy*, the *Zynq Proxy* (the green line with rectangles) delivers much higher performance, as shown in Fig. 7b. To transmit a new job to all the connected



(a) Zynq Platform: ONetSwitch45 Board



(b) Real-time Execution of the Mining Proxies

Fig. 7: Hardware Platform and Performance Comparison

TABLE II: Resource Utilization on ZYNQ Z-7045

	LUT	LUTRAM	FF	BRAM-18Kb
Used	10159	652	16893	87
Utilization (%)	4.7	0.9	3.9	8.0

miners, it only takes about 2.9 μs as reported in Table I. The *Zynq Proxy* achieves nearly 2079× and 18× speedup with 225 miners compared to the *Original Proxy* and *SW-L2 Proxy* implementations, respectively.

B. Resource Consumption

In terms of the hardware cost, as reported in Table II, the resource utilization of the *Zynq Proxy* is very low. This illustrates that, although we used ONetSwitch45 board for implementation, the proposed proxy can be implemented on smaller and more cost-efficient platforms such as the Zynq Z-7010. This will significantly reduce the cost compared to the existing Intel NUC-based mining proxies.

C. Discussion

From a performance perspective, one might argue that a hand-optimized CPU implementation of the mining proxy could prove better than our hardware/software co-designed platform. However, for CPU implementations, the TCP/IP kernel stack becomes a bottleneck, even with a highly optimized implementation of the network stack, which means all the CPUs will be occupied for network activity. In our platform, the CPUs are free to execute other workloads while the FPGA handles the job broadcasts in an efficient manner. Furthermore, our proposed approach opens up the possibility of applying hardware/software co-design methodology to acceleration of the mining pool server and the full-node network as well by offloading certain functions on FPGAs. For example, the FPGA integrated in a mining pool server can directly generate jobs after receiving transactions and block information from the network, and send the job to the connected mining proxy or miners. Likewise, for a full-node, the FPGA can help with fast relaying of blocks by offloading the block verification and transmission.

V. Related Work

FPGAs are widely deployed in the cryptocurrency domain. Most of the prior work [5][6][8] focuses on developing various high-performance and low-power consumption algorithms for mining cryptocurrencies. However, as the current best hashing rates are dominated by ASIC miners, FPGA-based miners have become less attractive. In today's miners (for example Bitmain Antminer [2]), FPGAs are used as the control unit to communicate with mining pools and send jobs to its ASIC miners.

In [9], the authors used FPGAs from a network perspective to develop a key-value store accelerator to reduce latency when accessing Merkle hashes stored in a full node. In [7], the authors proposed various mechanisms to improve the security of the Stratum protocol. In contrast to this existing work, we address the scalability issue in mining proxies by proposing a hardware/software co-designed architecture for the Stratum mining proxy.

VI. Conclusion

In proof-of-work cryptocurrencies like Bitcoin, a mining proxy is used to manage miners locally, and this can become a performance bottleneck and hamper scalability. In this paper, we show how to use a layer 2 broadcast mechanism for sending jobs from the mining proxy to the miners. Our broadcast mechanism can be implemented in both software running on a CPU or hardware synthesized on an FPGA. From our experiments, the modifications to the mining proxy can achieve a speedup of 2079× in transmission time compared to an original Intel i7 server implementation.

Acknowledgments

The authors thank Ji Yang from Xilinx, and Feng Jin, Qun Li, Xunliao Guo and Youqing Han from Bitmain for their valuable support and useful discussions.

References

[1] CoinMarketCap. https://coinmarketcap.com/all/views/all. Accessed: 2019-03-28.

[2] Bitmain, 2019. https://www.bitmain.com/.

[3] Bitmain. AntPool, 2019. https://www.antpool.com/.

[4] Chengchen Hu et al. Design of all programable innovation platform for software defined networking. In *Presented as part of the Open Networking Summit 2014 (ONS 2014)*, Santa Clara, CA, 2014. USENIX.

[5] J. Barkatullah et al. Goldstrike 1: CoinTerra's First-Generation Cryptocurrency Mining Processor for Bitcoin. *IEEE Micro*, Mar 2015.

[6] M. V. Beirendonck et al. A Lyra2 FPGA Implementation for Lyra2REv2-Based Cryptocurrencies. *arXiv*, 2018.

[7] Ruben Recabarren et al. Hardening stratum, the bitcoin pool mining protocol. *CoRR*, abs/1703.06545, 2017.

[8] Teknohog et al. Open Source FPGA Bitcoin Miner, 2011. https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner.

[9] Y. Sakakibara et al. An FPGA NIC Based Hardware Caching for Blockchain. In *HEART*, 2017.

[10] MeshSr. ONetSwitch Development Board, 2019. http://www.meshsr.com/SmartSwitches/ONetSwitch45.

[11] Marek Palatinus. Stratum Mining Proxy, 2019. https://github.com/slush0/stratum-mining-proxy.

[12] Pooler. CPU Miner, 2019. https://github.com/pooler/cpuminer.

[13] Xilinx. Zynq-7000 SoC, 2019. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.