Hardware Support for Dynamic Protocol Stacks

Ariane Keller Daniel Borkmann Stephan Neuhaus
ETH Zurich
Zurich, Switzerland
first.last@tik.ee.ethz.ch

1. INTRODUCTION

The tremendous success of the Internet can be attributed to the diversity of supported physical transport media, which allows users to be connected always and everywhere; and its plethora of applications that offers something for anyone. With the addition of more and more features to the original Internet architecture (such as firewalls, VPNs, NATs, P2P, etc.) the question arises whether the Internet architecture ought to be redesigned from scratch. Several research initiatives [1, 2, 4] supported work in the area of clean slate architectures. In the context of such an initiative, we suggested in earlier work [6, 7] to split network functionality into individual *functional blocks* (FBs) that can be assembled into optimized protocol stacks at runtime. However, it was never clear how such protocol stacks could benefit from hardware accelerators, for example for checksum calculation, encryption or intrusion prevention. A state-of-the-art ASIC implementation is unsuitable since the provided functionality is fixed and cannot be optimized at runtime.

In this paper we present EmbedNet, a dynamic protocol stack System-on-Chip (SoC) implementation on a state-of-the-art FPGA (Virtex-6 ML605 evaluation board [5]). This FPGA hosts a softcore MicroBlaze CPU [3] capable of running Linux, as well as several *hardware threads* that are enabled to do interprocess communication (shared memory, message passing) with the Linux kernel space. These mechanisms are provided by ReconOS, an operating system extension for reconfigurable hardware [8]. The hardware threads can be reconfigured while other parts of the FPGA are processing data.

In EmbedNet, FBs can be executed either in the Linux kernel space or in a hardware thread. Since hardware threads can be reconfigured at runtime, the mapping of functional blocks between hardware and software can be optimized with respect to the current network traffic mix. In this paper we present the architecture of EmbedNet and a performance analysis of the first prototype implementation.

2. EMBEDNET

The overall EmbedNet architecture is depicted in Figure 1. Packets are sent from an application through a BSD socket to the Linux kernel. Here, packets are forwarded between the individual FBs by a *packet processing engine*. Each FB has an associated flag that determines whether the FB is currently executed in hardware or in software. If the FB should be executed in hardware the PPE copies it to the memory region that is shared with the hardware and notifies the hardware of the new packet. The hardware then reads the packet from the shared memory and forwards it to the next FB. From there it can be either sent to another FB in hardware, software or to the Ethernet interface.

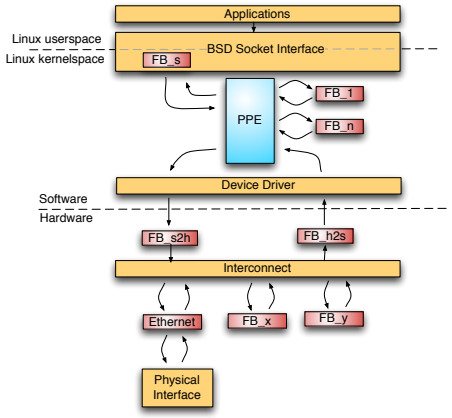

Figure 1: Architecture of EmbedNet.

In hardware packets are forwarded in a *network on chip* (NoC) consisting of switches arranged in a unidirectional ring. Each switch is also connected to a configurable number of functional blocks. A software controller configures the FBs with the addresses of the other FBs so that packets can be forwarded correctly.

3. EVALUATION

We evaluated the correct functioning of the EmbedNet node with the scenario depicted in Figure 2. In a first step, an application sends packets through the Ethernet FB to another node. In a second step, we add

a software FB while the application continues to send data. Eventually, in a third step, we move this FB to the hardware. In order to evaluate the change of the protocol stack, the FBs in software and hardware were implemented slightly different. Thus, on the packet collecting node, we are able to see a change in the payload pattern of the received packets.

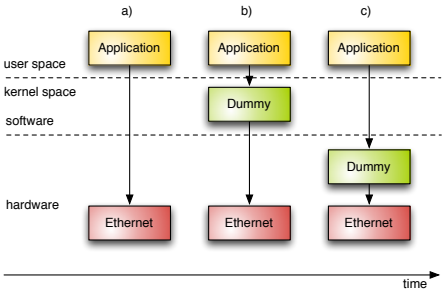

Figure 2: Functional evaluation setup.

We evaluated the maximum sending rate of EmbedNet in different scenarios (see Figure 3). The baseline evaluation reveals a maximum packet sending rate of 4'000 minimum-sized packets per second from normal Linux raw sockets on the ML605 evaluation board. This rather low rate might be due to the low MicroBlaze CPU frequency and a non DMA-capable network IP core. Exchanging the Linux raw sockets with PF_LANA sockets—a new socket class for EmbedNet—shows that PF_LANA (without hardware support) is currently about 1'000 packets per second slower. A second baseline evaluation showed that the sending rate of packets from within the Linux kernel using the EmbedNet hardware is almost independent from the payload size. Sending packets from PF_LANA trough the EmbedNet hardware is again about 1'000 packets per second slower.

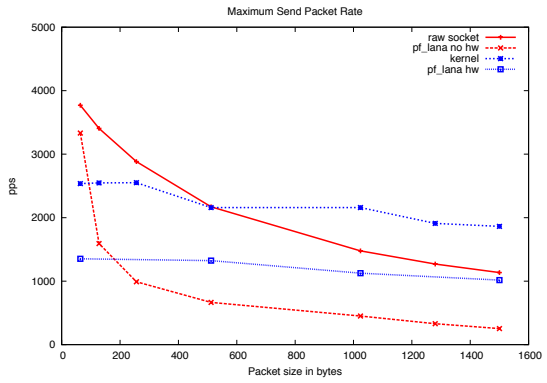

Figure 3: Maximum sending rate comparison.

This suggests that the limiting factor is the handshake required to transmit data over the software/hardware boundary. Therefore, we plan to implement a ring buffer in which several packets can be transmitted at once. This should increase the performance by a factor corresponding to the number of packets in this buffer.

We also measured the maximum throughput of the hardware by sending packets from an external node to EmbedNet where it was forwarded to a dummy FB (which does nothing) and back to the Ethernet FB. The maximum throughput is currently 0.8 Gbit/s which corresponds to the forwarding rate of the switches (8 bits at 100 MHz). In the future we plan to run the switches at 125 MHz which would allow for line rate forwarding.

4. CONCLUSIONS AND FUTURE WORK

We showed that FPGAs can be used to provide hardware support for dynamic protocol stacks, but that the softcore CPUs currently used with FPGAs are rather slow and therefore the overall throughput small. This suggests implementing as much functionality as possible in hardware and to implement a ring buffer between hardware and software to decrease the load on the CPU.

In order to provide an optimal mapping of functional blocks to either hardware and software, we are currently working on a controller that determines the best mapping at runtime based on the packets to process.

5. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n° 257906.

6. REFERENCES

[1] FIND – Future Internet Design (FIND) - US National Science Foundation. At http://www.nets-find.net/, (May 09).

[2] Future Internet Research and Experimentation (FIRE) initiative. http://cordis.europa.eu/fp7/ict/fire, (May 09).

[3] MicroBlaze Soft Processor Core. http://www.xilinx.com/tools/microblaze.htm.

[4] NWGN – New-Generation Network R&D Project - Japan. At http://nwgn.nict.go.jp/index_e.html, (June 10).

[5] Virtex-6 FPGA ML605 Evaluation Kit. http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm.

[6] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ANA). *Selected Areas in Communications, IEEE Journal on*, 28(1):4 –14, Jan. 2010.

[7] A. Keller, D. Borkmann, and W. Mühlbauer. Efficient implementation of dynamic protocol stacks. In *ANCS*, page 8384, Washington, DC, USA, oct 2011. IEEE Computer Society.

[8] E. Lübbers and M. Platzner. ReconOS: An RTOS supporting hard- and software threads. *IEEE Int. Conf. on Field Programmable Logic and Applications*, 2007.