Reconfigurable Nodes for Future Networks

Ariane Keller
and Bernhard Plattner
ETH Zurich, Switzerland
Email: ariane.keller@tik.ee.ethz.ch

Enno Lübbers
EADS Innovation Works
Ottobrunn / Munich, Germany
Email: enno.luebbers@eads.net

Marco Platzner
and Christian Plessl
University of Paderborn, Germany
Email: christian.plessl@uni-paderborn.de

***Abstract*—Future network architectures aim at solving the shortcomings of the traditional, static Internet architecture. In order to provide optimal service they have to adapt their functionality to different networking situations. This can be achieved by dividing the networking functionality into modular blocks and combining them as required at runtime. While the feasibility and flexibility of novel network architectures have been successfully demonstrated on software-based prototypes, they are often unable to provide sufficient performance due to the lack of hardware acceleration.**

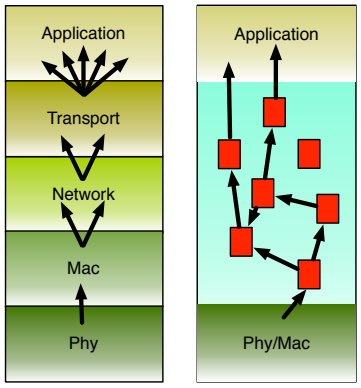
We present a networking node architecture for future Internet applications that provides a reconfigurable hardware/software platform in which the modules of the node's network stack can be flexibly distributed at runtime across hardware and software. By utilizing novel reconfiguration-aware communication mechanisms for functional blocks both within and across the hardware and software domains, our flexible node architecture enables dedicated hardware acceleration for adaptive networking.

I. Introduction

While the current Internet architecture has proven to be a successful foundation for data networks with a relatively slow-changing topology and known functionality, it frequently falls short of expectations when applied to emerging network applications. With the introduction of novel computing nodes (such as sensor nets), communication types (such as mobile communication, or communication in social networks) and security concerns (DDOS attacks, attacks against the domain name system), the Internet architecture is reaching its limits [1], [2]. Its well defined, static protocol stack is too rigid as that it could effectively adapt to those new situations.

As an alternative to the Internet architecture, several novel, clean slate network architectures have been proposed for the future Internet. An overview is provided by [3]. These architectures break up the static nature of the Internet and offer more flexibility, have built in security features, and/or support a variety of different protocol stacks. Figure 1(a) shows the Internet architecture where the protocol layering is static and well defined whereas Figure 1(b) depicts a modular architecture in which the used modules are chosen based on the current requirements. In addition to legacy networking functionality support, it is also possible to include functions in the networking architecture that were previously attributed to the application layer, such as encryption or compression.

The flexibility of modular and adaptive network architectures however comes at a cost. Where the well-defined TCP/IP architecture can benefit from high-volume high-performance



(a) Internet architecture (b) Adaptive network architecture

Figure 1. Comparison of the Internet protocol stack and an adaptive network architecture.

ASIC hardware implementation, a flexible networking environment is somewhat of a moving target for hardware acceleration. For most proposed future network architectures an ASIC implementation is infeasible, since all functions need to be known precisely at design time but the novel network architectures adapt their functionality to the current, run-time communication needs. Though a software implementation provides the required flexibility, it is intractable in many situations that demand high processing performance or performance/power ratios, such as in backbone routers, mobile communication equipment and autonomous sensor networks.

In order to combine the processing power of an ASIC implementation with the flexibility of a software implementation, research in networking increasingly turns towards FPGA platforms [4], [5]. Within this class of reconfigurable devices, hybrid CPU/FPGA platforms provide an additional measure of flexibility: they combine a microprocessor core for software processing with a programmable logic fabric capable of implementing high-performance data-parallel hardware in a *reconfigurable System on Chip* (rSoC). Of particular interest for future networking architectures are FPGA families that allow for changing the configuration of their logic cells at run-time through partial reconfiguration[1] in order to achieve highest flexibility.

The contribution of this paper is an architecture enabling

[1]Partial run-time reconfiguration enables the reprogramming of a part of the FPGA's logic, while simultaneously another part carries on processing data.

transparent communication between functional blocks implementing different networking functions, regardless whether the blocks are implemented in hardware or software. As a networking architecture we build upon ANA [6], [7] and its extension towards reconfigurable hardware [8]. While in [8] we have focused on the general concepts of adaptive networking for embedded systems, the work presented in this paper describes the details of the communication mechanisms across the hardware/software boundary, as facilitated by our architecture. By integrating a communication infrastructure that respects the run-time reconfigurability of the targeted platforms, we strive to maintain the performance advantage of dedicated hardware implementations for parallelizable processing workloads while approaching the flexibility of a software-based solution.

The remainder of this paper is structured as follows: in section II we provide a brief history of configurable network architectures. In section III we give an overview of ANA, especially the node architecture. Section IV gives an overview of ReconOS, an operating system tailored for embedded reconfigurable devices. Section V shows how to combine ANA with ReconOS in order to arrive at a run-time reconfigurable adaptive networking architecture for embedded devices. Section VI concludes the paper.

II. Related Work

The composition of protocol stacks dedicated to a specific use has been an ongoing research area for the past 15 years, dating back to the x-kernel [9] which defines an explicit architecture for constructing and composing network protocols. In the following years several projects were suggested, and with the Click modular router [10] one project achieved widespread use. In the Click project the networking functionality is divided into small elements, each of which performs a simple computation such as decrementing a TTL or queuing a packet. Independent from Click, the BSD operating system offers the possibility to create a modular networking architecture with its Netgraph subsystem [11].

However, these projects are not targeted to a future network environment in which the network is more heterogeneous and should adaptively optimize its service. More recently, many projects have been launched (e.g., within the FIND initiative [12] in the US, the FIRE initiative [13] in Europe and the new generation network project [14] in Japan) that target different aspects of future networks [3]. There is a broad consensus in the community that in the future, network functionality should be organized in *flexibly composable small building blocks*. The ANA [6] project is fully in line with this approach and provides a networking architecture in which the available services can be adapted continuously. ANA even allows for running completely distinct protocol stacks side-by-side and is thus well suited for a heterogeneous and mutable computing environment. We present an overview of ANA in section III and use it in section V as a conceptual basis for our architecture.

Many networking research projects that are using FPGAs have been implemented on one of two platforms: The FPX [15] and the newer NetFPGA [5] platform. The NetFPGA platform provides a Xilinx Virtex-II Pro FPGA on a board with four 1 GBit/s interfaces. For example, [16] uses the NetFPGA platform in combination with partial, run-time reconfigurability to provide an efficient network virtualization platform.

III. ANA

The concepts developed in the Autonomic Network Architecture (ANA) project [7] form the foundation of our adaptive networking architecture. In order to provide high flexibility, ANA divides networking functionality into *functional blocks (FB)* that may be combined as required by any given situation. The functionality of a single FB can range from a full monolithic network stack down to a small entity like computing a checksum.

In order to allow the construction of arbitrary network stacks, ANA provides an API (application programmer interface) which describes all communication between different networking functions and is based on a publish/resolve architecture. This API is beyond the scope of this paper—it is described in [7]. In this paper we focus on the communication inside a node.

The dynamic combination of FB rests upon the concept of indirection and is realized with so-called *information dispatch points (IDPs)*. IDPs are somewhat similar to file descriptors and sockets in UNIX systems. Instead of a tight binding between networking functions, where one function directly calls another function, the interaction is based on a *message passing system*. Hence, when data needs to be processed with a certain operation, the data is sent to an IDP that is bound to a function implementing that operation. This layer of indirection allows us to dynamically adapt the networking stack by changing the bindings of IDPs to FBs. This indirection will also prove useful for the transparent communication between hardware and software. All the FBs, IDPs and the bindings between them are managed in a central place called *minmex* that manages them in a so-called *Information Dispatch Table (IDT)*. Sending a packet from one FB to another is therefore done with the help of the minmex. Figure 2 shows the minmex with three functional blocks. The little dots represent IDPs and the arrows indicate that messages are always routed over the minmex.

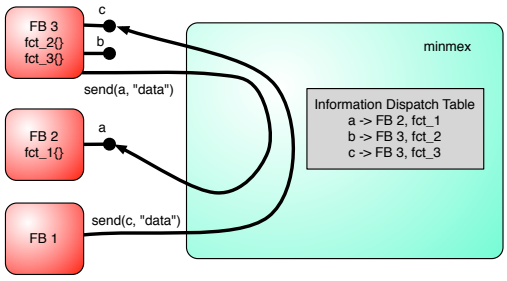


Figure 2. Data forwarding in an ANA node.

This architecture is currently implemented on different UNIX-like operating systems such as Linux and OS X. The

implementation is a pure software implementation without any hardware acceleration and therefore cannot handle high data rates. In the next chapter we introduce ReconOS, an operating system that allows the easy integration of functions implemented in hardware with functions implemented in software.

IV. ReconOS

ReconOS [17], [18] is an operating system for reconfigurable computing. It is implemented on an rSoC—a system integrating a dedicated CPU with an FPGA, which can be partially reconfigured at runtime. ReconOS extends existing operating system kernels, such as Linux, with functions and modules for integrating hardware and software into a single execution environment. More generally speaking, ReconOS extends the multithreaded programming model—already in widespread use within software-based systems—towards reconfigurable hardware. It models hardware modules as hardware threads on the same level with software threads. Hence, hardware threads interact with other threads using the same programming model primitives as their software counterparts. ReconOS applications are thus typically crafted from the following operating system objects: *threads*, *semaphores* and *mutexes*, *shared memory, message queues* and *mailboxes*.

The fact that all inter-thread activity is carried out using only these objects provides complete transparency within these interactions; a thread does not need to know whether its communication or synchronization partners are located in hardware or software. This transparency greatly facilitates the communication between functionality implemented in hardware and functionality implemented in software.

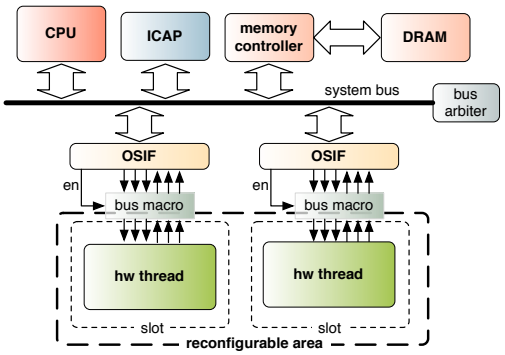


Figure 3. ReconOS system architecture.

The main part of any ReconOS system is a central microprocessor executing the host operating system kernel. Current implementations of ReconOS use a two-bus architecture for control and data communication between hardware threads and the OS kernel. Hardware threads are located within specified regions of the reconfigurable fabric, called slots, which are connected to the communication infrastructure through a dedicated hardware module, the OS interface (OSIF) (as described in [17]). Individual hardware threads can be exchanged during run-time through partial reconfiguration of the FPGA, which provides a means of dynamically adapting the system's hardware/software partitioning to changing requirements.

The architecture of an example ReconOS system with two slots is depicted in Figure 3. Besides the main CPU and the operating system interfaces, the main bus also connects a DRAM controller and the FPGAs internal configuration access port (ICAP), which is used for dynamic partial reconfiguration.

ReconOS supports transparent virtual memory management, where both software and hardware threads can use the same pointers to access the same memory region. Therefore, ReconOS provides an additional memory management unit implemented in hardware that accesses the page table of the Linux operating system and stores the translations in a translation lookaside buffer. Page faults are processed in software in order to use operating system functions for memory protection and demand paging. This virtual memory management allows hardware and software threads to easily access a shared memory region.

V. Hardware/Software Co-Design for ANA

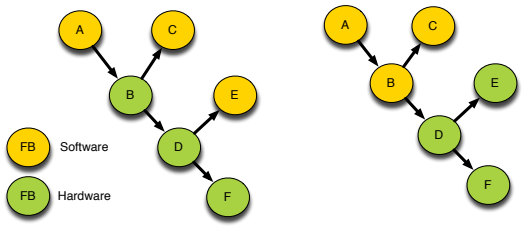
In this section we introduce our approach to hardware/software co-design of adaptive network nodes. It relies on ANA for the mechanisms to assemble functional blocks (FB) into flexible protocol graphs, and on the ReconOS/Linux operating system to realize transparent and efficient communication between FBs implemented in hardware and FBs implemented in software.

A. FB Graph

The FB forming the networking functionality of a node together with their interconnection can be displayed as a task graph. Nodes represent FBs and edges represent packets sent from one FB to another. The purpose of our architecture is to allow the implementation of FBs in hardware, software, or both. This gives us the flexibility to decide at runtime which FB should be executed in hardware and which in software, based on the packets that currently need to be processed. Figure 4 shows the graph of the FBs of one node at two different points in time. In Figure 4(a) *FB B* is implemented in hardware and in Figure 4(b) *FB B* is implemented in software. This means that *FB A* has to send the packets once to an FB implemented in hardware and once to an FB implemented in software. However, the internal operation of *FB A* should not be influenced by the current hardware/software partition. Hence, the routing of packets between functional blocks should be transparent to the sending FB.

B. System-on-Chip Architecture

Figure 5 sketches the architectural layout of a node. In addition to the elements provided by ReconOS, it includes a specialized multibus for interconnecting the hardware cores. This multibus is needed since the bandwidth of the shared bus is limited to a maximum of close to 1.6 GBit/s as shown in [8]. The shared bus is used for the communication between FBs executed in software and FBs executed in hardware. For the communication between the FBs executed in hardware the



(a) Hardware / Software partitioning at time t_1 (b) Hardware / Software partitioning at time t_2

Figure 4. Graph representing the networking functionality of a node. Depending on the time, different FBs are mapped to hardware and to software.

dedicated multibus is used. Each hardware thread receives data from one bus via a FIFO-like interface. A hardware thread can further send data to the buses assigned to other threads. For arbitration, round robin or thread priority based schemes can be used. Alternatively, more complex arbitration schemes with access control based on the priority of individual packets are conceivable.

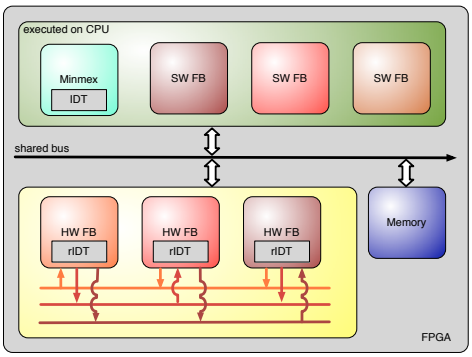


Figure 5. System architecture for an adaptive network node. FBs are executed in SW (on the CPU) or HW (on the FPGA) and communicate over a shared bus (SW↔SW, HW↔SW) or dedicated multibus links (HW↔HW).

This architecture improves the performance of a node due to two effects. First, computationally expensive FBs (e.g., for encryption) can be executed faster in hardware than in software, and, second, the processing in hardware is done in a pipelined manner, where each FB in hardware represents one pipeline stage.

C. Hardware / Software Interface

The interface between software and hardware FBs is implemented through a combination of message passing and ring buffers that are accessible by the minmex and the respective hardware FB. While data is read and written to the shared memory through burst transactions on the hardware thread's bus master interface, the communication partner is notified about new data through a OS-provided message box that specifies the number of bytes written to the ring buffer. The message format in the ring is as follows: *packet length / IDP / data*. To avoid an interrupt after every packet multiple notification messages can be aggregated in a single message, providing the total length of newly written bytes.

Packets sent from a software FB are forwarded to the minmex in an identical manner to the software-only implementation. The minmex looks up whether the destination IDP is currently mapped to a software FB or to a hardware FB. If it is mapped to a hardware FB, the IDT stores a pointer to the shared memory ring and the message queue used to notify the corresponding thread.

For *packets processed by a hardware FB* forwarding every packet to the minmex is inefficient, since this would interrupt the software for every packet. To avoid this overhead, each hardware FB maintains a partial replica of the IDT (rIDT) of the minmex that allows it to decide how to forward a given packet. The entries in the rIDT are updated by the minmex. Based on the rIDT the hardware FB decides a) where to forward a processed message (to software or to which bus interface) and b) what to do with a packet that it received on a given IDP. In order to provide an efficient lookup from IDPs to actions, a CAM (content addressable memory) is used. If the packet is sent to another hardware FB, an arbiter is asked for sending permission on the bus corresponding to the IDP.

D. Dynamic Reconfiguration

There are two aspects that demand a reconfiguration of the system at runtime. Either a node adapts the protocol stack, e.g., when a new application with new requirements is started or when new networking functionalities such as encryption or compression are added to the already existing networking task graph. Or the network traffic mix changes, e.g., the amount of data that needs to be processed by a specific FB changes. An optimal hardware/software partitioning depends on the current traffic mix, the processing time of a packet in a given FB and the system overhead.

Our architecture supports this dynamic assembly of FBs by modeling them as threads in ReconOS. Exchanging hardware threads or exchanging software threads with hardware threads and vice versa is enabled by the partial reconfiguration capability of ReconOS. If a new FB is to be executed in hardware and a hardware slot is available, ReconOS reconfigures this hardware slot with the new FB, while the FBs running in other hardware slots are not affected and can process data continuously. As soon as the FB is correctly configured, the minmex can initialize the FB and start dispatching data to it.

The physical size of the reconfigurable areas on the FPGA (the slots) is determined at design time, taking into account the actual sizes of the possibly required hardware threads.

E. Experimental Evaluation

We have implemented the individual parts of our architecture on a Xilinx Virtex-II-Pro FPGA evaluation board. Both the clock rate of the hardware logic and the clock rate of the integrated PowerPC CPU are set to 100MHz. We have measured the delay for sending a minimum sized packet between two different FB that are either implemented in hardware or in software. Table I shows this round trip time (RTT). If both FBs are implemented in hardware, the RTT is by far the fastest, since there is no overhead involved. The

communication between two FBs implemented in software is faster by a factor of 3.5 to 7 compared to the communication that crosses the hardware / software boundary. Upon accessing the ring buffer for the first time from a hardware functional block a page fault occurs, which has to be handled in software by the operating system kernel. Since the delay with a page fault is about twice the delay without a page fault, we conclude that interrupting the CPU is the largest contributor to the RTT.

Table I
ROUND TRIP DELAY BETWEEN TWO FUNCTIONAL BLOCKS

Implementation	Delay
sw→sw→sw	886μs
sw→hw→sw (with page fault)	6157μs
sw→hw→sw (without page fault)	3168μs
hw→hw→hw	0.06 μs

Note that the increased round trip delay for a hybrid hardware/software solution does not discourage hybrid solutions since it only accounts for the delay of transferring control, but not the delay of data processing. A hardware implementation of a functional block can still outperform the software implementation with respect to delay by several orders of magnitude. However, the control transfer delay limits the granularity of functional blocks that can be successfully mapped to hardware.

VI. CONCLUSION

With the combination of ANA and ReconOS we have created a compute node that provides flexible and efficient networking services designed for the future of the Internet. Our node shows that novel network architectures can be implemented with hardware support in order to provide higher packet throughput—even though novel architectures might not be amenable for an ASIC implementation.

In our architecture, a node can adapt its configuration to the current traffic mix (e. g., the fraction of packets that requires a specific processing) or to the available protocols. Protocol functions can be executed either as hardware or software threads and the mapping of functionality to hardware or software can be adapted dynamically as needed. Transparent communication and synchronization between all threads is provided by ReconOS.

Ongoing work focuses on three main aspects. First, we are working on the comprehensive integration of the different parts of our architecture in order to improve the reusability of our concepts within the research community. Second, we focus on the implementation of different functional blocks to show the benefit of a dynamic mapping of functional blocks to either hardware or software. Third, we will develop algorithms for dynamically choosing the hardware/software mapping of the functional blocks at runtime. To this end we will enhance our execution environment with several monitoring elements to obtain run-time input such as utilization of an individual functional block or throughput between two functional blocks. Finally, for a performance evaluation, the architecture can be ported to the NetFPGA platform that provides more powerful networking connections than the development board we currently use.

ACKNOWLEDGMENT

This work was supported partly by the ANA project funded by the European Union Information Society Technologies Framework Programme 6, partly by the German Research Foundation under project number PL 471/2-2, and partly by the EPiCS Project funded by the European Union Seventh Framework Programme (FP7/2007- 2013) under grant agreement n^o 257906.

REFERENCES

[1] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the internet: the end-to-end arguments vs. the brave new world," *ACM Trans. Internet Technol.*, vol. 1, no. 1, pp. 70–109, 2001.

[2] P. Molinero-Fernández, N. McKeown, and H. Zhang, "Is IP going to take over the world (of communications)?" *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 113–118, 2003.

[3] Subharthi Paul, Jianli Pan and Raj Jain, "Architectures for the Future Networks and the Next Generation Internet: A Survey," 2009, http://www.cse.wustl.edu/~jain/papers/i3survey.htm (Oct 09).

[4] "High-Speed Networking and FPGA Solutions," http://www.invea-tech.com/products-and-services/overview (Jun 10).

[5] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–an open platform for gigabit-rate network switching and routing," in *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 160–161.

[6] "Autonomic Network Architecture - EU Project (2006-2009)," http://www.ana-project.org (Oct 09).

[7] 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*, vol. 28, no. 1, pp. 4 –14, Jan. 2010.

[8] E. Lübbers, M. Platzner, C. Plessl, A. Keller, and B. Plattner, "Towards Adaptive Networking for Embedded Devices based on Reconfigurable Hardware," in *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'10)*. Las Vegas, NV, USA: CSREA Press, 2010, pp. 225–231.

[9] N. C. Hutchinson and L. L. Peterson, "The X-Kernel: An architecture for implementing network protocols," *IEEE Trans. Softw. Eng.*, vol. 17, no. 1, pp. 64–76, 1991.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[11] "All About Netgraph," http://people.freebsd.org/~julian/netgraph.html (Aug 10).

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

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

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

[15] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an fpga with partial run-time reconfiguration," in *DAC '02: Proceedings of the 39th conference on Design automation*. New York, NY, USA: ACM, 2002, pp. 343–348.

[16] D. Unnikrishnan, R. Vadlamani, Y. Liao, A. Dwaraki, J. Crenne, L. Gao, and R. Tessier, "Scalable network virtualization using FPGAs," in *FPGA '10: Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2010, pp. 219–228.

[17] E. Lübbers and M. Platzner, "ReconOS: Multithreaded programming for reconfigurable computers," *ACM Transactions on Embedded Computing Systems Special Issue (CAPA)*, 2009.

[18] ——, "ReconOS: An RTOS supporting hard- and software threads," *IEEE Int. Conf. on Field Programmable Logic and Applications*, 2007.