Efficient Implementation of Dynamic Protocol Stacks in Linux

Ariane Keller
ETH Zurich, Switzerland
ariane.keller@tik.ee.ethz.ch

Daniel Borkmann
ETH Zurich, Switzerland
HTWK Leipzig, Germany
borkmann@iogearbox.net

Wolfgang Mühlbauer
ETH Zurich, Switzerland
muehlbauer@tik.ee.ethz.ch

ABSTRACT

TODO: rewrite abstract - beginning is copied from an old paper... 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. In this paper we address the performance aspect of such architectures and we show that their performance is comparable with the performance of a standard Linux protocol stack.

1. INTRODUCTION

Some references that might be useful: [3] (ANA) and [7] (Click) and [4] (From protocol stack to protcol heap: role-based architecture) and [5] (PLUTARCH:an arbument for network pluralism) and [1] (netgraph) and [8] (survey of next generation internet) and [6] (xKernel) and [9] (model for flexible high-performance communication subsystem). - should we explicitly say something on active networking or should we try to avoid it completely?

Placeholder: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque eu arcu ut est volutpat consequat sit amet dignissim enim. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc magna purus, vehicula sit amet fringilla ac, interdum ut dui. Ut in magna tortor, vitae dignissim lorem. Praesent condimentum eros aliquam mi pharetra egestas. Sed sem tortor, iaculis non ornare consequat, auctor eget velit. Nam nibh nibh, ullamcorper vitae gravida non, rhoncus vitae mi. Nunc vestibulum suscipit justo in laoreet. Praesent ac porta ante. Integer sem urna, pretium sed dignissim id, laoreet sit amet sem. Cras ac risus nec nibh tempor gravida. Integer ac ligula sed orci luctus condimentum at quis dui. Etiam dignissim dignissim tellus, et dapibus elit venenatis nec. Sed hendrerit imperdiet lacinia. Sed enim purus, mattis vel ullamcorper vel, malesuada vitae turpis. Pellentesque nec lacus tortor, eget scelerisque lectus. Suspendisse lectus mauris, tempor eget porttitor id, consectetur vel sem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque eu arcu ut est volutpat consequat sit amet dignissim enim. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc magna purus, vehicula sit amet fringilla ac, interdum ut dui. Ut in magna tortor, vitae dignissim lorem. Praesent condimentum eros aliquam mi pharetra egestas. Sed sem tortor, iaculis non ornare consequat, auctor eget velit. Nam nibh nibh, ullamcorper vitae gravida non, rhoncus vitae mi. Nunc vestibulum suscipit justo in laoreet. Praesent ac porta ante. Integer sem urna, pretium sed dignissim id, laoreet sit amet sem. Cras ac risus nec nibh tempor gravida. Integer ac ligula sed orci luctus condimentum at quis dui. Etiam dignissim dignissim tellus, et dapibus elit venenatis nec. Sed hendrerit imperdiet lacinia. Sed enim purus, mattis vel ullamcorper vel, malesuada vitae turpis. Pellentesque nec lacus tortor, eget scelerisque lectus. Suspendisse lectus mauris, tempor eget porttitor id, consectetur vel sem. Praesent ac porta ante. Integer sem urna, pretium sed dignissim id, laoreet sit amet sem. Cras ac risus nec nibh tempor gravida. Integer ac ligula sed orci luctus condimentum at quis dui. Etiam dignissim dignissim tellus, et dapibus elit venenatis nec. Sed hendrerit imperdiet lacinia. Sed enim purus, mattis vel ullamcorper vel, malesuada vitae turpis. Pellentesque nec lacus tortor, eget scelerisque lectus. Suspendisse lectus mauris, tempor eget porttitor id, consectetur vel sem.

2. THE BODY OF THE PAPER

Description of the base architecture see Figure 1.

- Environment: kernel space - kernel XXX
- Application Interface: socket API
- Hardware Interface: Vlink subsystem
- Configuration: what can be configured?, Command line interface
- Internals: state Management, Functional block notifier

2.1 Configuration Interface

The protocol stack can be configured from user space with the helb of a command line tool. The most important commands are summarized below.

- `add`, `rm`: adds (removes) a functional block from the list of available functional blocks in the kernel.
- `set`: sets properties of a functional block with a `key=value` semantic
- `bind`, `unbind`: binds (unbinds) a functional block to another in order to be able to send messages to it.
- `replace`: replaces one functional block with another

ANCS 2011 Brooklyn, New York, USA

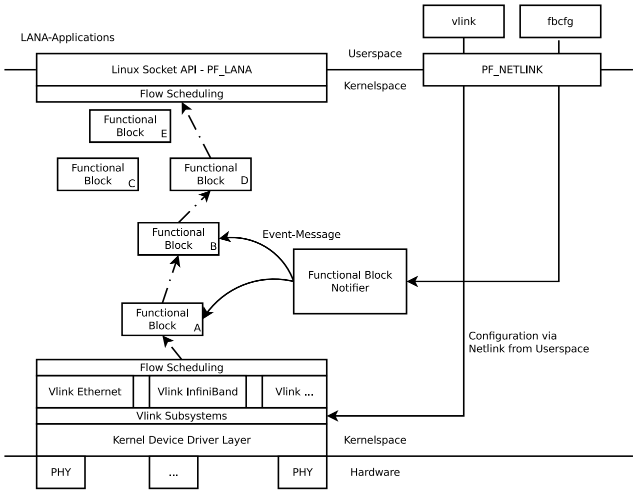

Figure 1: Lana architecture

functional block. The connections between the blocks are maintained. Private data can either be transferred to the new block or dropped.
- `subscribe`, `unsubscribe`: Subscribe (unsubscribe) one functional block to receive control messages from another functional block.

2.2 Improving the Performance

During the implementation of our framework we have evaluated different possibilities for the integration of our packet processing engine with the Linux kernel. We think the insights gained are interesting for other researcher that have to do fundamental changes on the Linux protocol stack and hence, we summarize them here. Our goal was to be able to process as many *minimum sized Ethernet frames* as the Linux kernel is able to process. In order to compare the performance of the Linux Kernel and the performance of our engine we have dropped all packets in the Linux Kernel protocol stack as soon as they were arrived (TODO: where exactly?). In our system the packets were processed by the fb_eth functional block followed by two fb_dummy functional blocks that were simply forwarding the packets. We can distinguish the following three approaches:

- On each CPU there exists one high priority thread that is responsible for processing LANA packets. This approach leads to a starvation of the interrupt handler (ksoftirqd) and hence the maximal achieved packet rate is only about half as what is achieved by the protocol stack of the Linux kernel. Also changing the priority of the LANA thread to normal only slightly increases the throughput.
- Instead of relying completely on the Scheduler of the Linux Kernel we control preemption and scheduling explicitly. This approach still exhibits scheduling overhead, but it increases the performance to about two thirds of the performance of the Linux Kernel.
- Instead of executing the LANA functions in a dedicated thread they are executed directly in the ksoftirqd function. With this approach approximately 95% of the performance of the Linux kernel is achieved.

The corresponding numbers are listed in Table 1.

mechanism	performance
dedicated kernel thread (high priority)	700000
dedicated kernel thread (normal priority)	750000
dedicated kernel thread (controlled scheduling)	900000
execution in ksoftirqd	130000
Linux kernel stack	138000

Table 1: Performance evaluation (pps) of different approaches to receiving packets in the Linux kernel. The packets are 64 Bytes long. The evaluation was done on a TODO: CPU/RAM/NETWORKCARD/KERNEL

2.3 Software Available

The current sofware is available under the GNU General Public License from [2]. In addition to the framework it also includes four functional blocks: Ethernet, Berkeley Packet Filter, Tee (duplicate a packet), and Forward (an empty Block that just forwards the packets to another block). The framework does not need any patching of the Linux kernel but needs a new, 2.6.X kernel.

3. CONCLUSIONS AND FUTURE WORK

We have shown that it is possible to implement a flexible protocol stack that has a similar performance than the default protocol stack in the Linux kernel. This allows for the easy inclusion of new, still to be developed protocols and for the change of the protocol stack at runtime to include for example compression or encryption as the networking conditions change.

In the short term we will compare the performance of real scenarios implemented in our system with the performance of an implementation in other systems (for example default Linux protocol stack or the Click router). In the midterm we will develop a mechanism that automatically sets up a protocol stack for an Application whereby the Application can specify some characteristics the communication channel should have, but not exactly how this has to be achieved. For example the application could require a "reliable communication channel" and a controller would choose between different protocols that provide reliability (e.g., one for wired communication, one for wireless communication, one for wireless, multi-hop communication). The setup of the protocol stack will have to be negotiated between the source and destination node. The end goal will be to have a networked system that requires less configuration as compared to todays networks and that is able to adapt itself to changing network conditions.

4. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n^o257906.

5. REFERENCES

[1] All About Netgraph. http://people.freebsd.org/âĹijjulian/netgraph.html (Aug 10).

[2] Lightweight Autonomic Network Architecture for the Linux kernel. http://repo.or.cz/w/ana-net.git (Jul 11).

[3] 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.

[4] R. Braden, T. Faber, and M. Handley. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22, 2003.

[5] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: an argument for network pluralism. In *Proceedings of ACM SIGCOMM FDNA Workshop*, August 2003. Karlsruhe, Germany.

[6] N. C. Hutchinson and L. L. Peterson. The X-Kernel: An architecture for implementing network protocols. *IEEE Trans. Softw. Eng.*, 17(1):64–76, 1991.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[8] 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).

[9] M. Zitterbart, B. Stiller, and A. N. Tantawy. A model for flexible high-performance communication subsystems. *IEEE Journal on Selected Areas in Communications*, 11(4):507–518, May 1993.