David Choffnes

U-Net: A User-Level Network Interface for Parallel and Distributed Computing
T. von Eicken, A. Basu, V. Buch, W. Vogels (Cornell)

One-liner: The authors present a user-level network interface that 1) removes much of the processing overhead from kernel-to-user space (and vice versa) operations, thus reducing network latency, and 2) gives user applications direct access to a virtual network interface, enabling custom network protocol implementations.

In a narrative that reads much like an exokernel diatribe, the U-Net authors make the case for taking a service out the kernel, where it has been implemented narrowly and poorly, and placing it in user space, where the user has the flexibility to implement network communication freely and without incurring the overhead of excessive buffer copying performed by the kernel. The result of this work is reduced message latencies, enabling applications to saturate the bandwidth of high-capacity links, which was not possible using the kernel NI. The target of this service is parallel and distributed computing, but the authors also implement services such as TCP, UDP and IP over U-Net to demonstrate its flexibility.

There are several drawbacks to this work when put in context. For one, although the work is visionary, it is questionable how much success it could have give the lack of hardware support for key optimizations in U-Net, such as “true zero-copy” buffering. Another limitation of the implementation was the need to pin physical memory allocated to user processes. For computers that run many processes, this could lead to terrible system errors such as out-of-memory. Finally, although the idea of virtualizing the NI to perform multiplexing is admirable, I did not read any discussion of fairness in the NI queues. Given that the target environment is parallel and distributed computing, security risks may be minimal; however, unfairness in the queuing policies could lead to simple exploits (and poorly written code) that ruin application performance. My final gripe is that the authors argue the same way that exokernel designers do, but the former did not pay as much attention to protection, and it seemed to me that there was potential for abuse.

I'm not sure what the impact of this work is today. It seems like a great idea to take something out of the kernel and replace it with user-level code providing the same functionality, but with additional flexibility. However, the same argument that drives the authors to go to user-land can be turned against them: once you move something out of the kernel, the kernel developers and not responsible for maintaining it, meaning the user-level project may someday die and the software may become incompatible with modern systems.. It is difficult to convince widespread adoption of services that move to a more volatile environment such as this. Overall, this is a very strong paper with a small number of issues related to protection and resource management that need to be dealt with.