CamIO: declaring flexible and performant application I/O

Matthew P. Grosvenor *Malte Schwarzkopf* *Steven Hand*
University of Cambridge Computer Laboratory

1 Introduction

Discussion of input/output (I/O) performance in the systems research community has centred around the interfaces between user and kernel space, and those between kernel-space and hardware. Work in this area often focuses on managing buffers to achieve zero-copy performance while maintaining isolation [1, 3–5]. However, the question of how to use existing I/O primitives *inside* applications while maintaining flexibility to alternate between mechanisms has received little attention. The mantra seems to be that "the developer will figure out the right mechanism". We are not so sure.

2 The Babylonian I/O primitive confusion

A large number of different I/O transport primitives are available for developers to chose from today, and in the face of increasingly fast communication hardware, new ones are frequently appearing. While classic examples are UNIX file I/O and BSD sockets have well-defined and understood semantics, their design dates back to an era when I/O was slow relative to computation. As a result, demand for high performance has led to a huge variety of direct or near-direct hardware access mechanisms, especially for network I/O [2, 4, 5]. In addition to these, various shared memory IPC mechanisms provide high-performance communication inside a chassis.

Each of these transports make various (often implicit) trade-offs with regards to performance, security, resource usage and reliability. For example, the mandated data copy in BSD sockets implies an assessment of the relative cost of a memory copy operation to the network operation, and the netmap [5] framework amortizes syscall cost to improve throughput at the cost of latency.

For an application programmer, it is difficult to know at design-time which of these is the right fit for the application. For example, a given application may choose to use either shared memory transport IPC on a single host, UDP inside a LAN or TCP/SSL over a WAN. Indeed, in practice developers often choose a lowest common denominator that will always work—for example, TCP in data center applications. Unhelpfully, each transport mechanism also comes packaged with its own non-portable interface. Even in the simplest case, it is notoriously difficult to port an existing application from using a UDP transport to a TCP transport. Would it not be desirable to have a single programming interface abstraction that permits late-binding to the underlying transport, but imposes near-zero cost at run time? This is what we are providing with CamIO.

3 CamIO

CamIO is a C-based I/O library that takes a declarative approach to constructing high performance stream transports. A generic stream interface is bound to an underlying transport at run-time using a URL-style specifier. These URIs can be dynamically generated or passed around, making applications flexible and easy to work with. Streams can be read from, written to and selected over using primitives similar to the familiar POSIX interface. CamIO internally ensures that buffer copying is minimized to the least number of copies required by the underlying transport.

CamIO is a work in progress, but already supports a wide variety of transports, including shared memory rings, UDP and TCP sockets, netmap direct network access and various file-based streams. Performance of a CamIO-based `cat` application is on par with the original, but offers vastly improved flexibility. At APSys 2012, we plan to demonstrate CamIO's flexibility and performance using a CamIO-based key-value store implementation that can dynamically use either shared-memory or networked transports to communicated with clients, with no code changes or recompilation required. The CamIO source code will shortly be available under the permissive BSD 3-Clause license.

References

[1] DONNELLY, A. Resource control in network elements. Tech. rep., Univ. of Cambridge, 2002.

[2] HAN, S., ET AL. MegaPipe: a new programming interface for scalable network I/O. In *Proceedings of OSDI* (2012).

[3] KHALIDI, Y. A., ET AL. An Efficient Zero-Copy I/O Framework for UNIX. Tech. rep., Sun Microsystems, Inc., Mountain View, CA, USA, 1995.

[4] PAI, V. S., ET AL. IO-Lite: a unified I/O buffering and caching system. *ACM Trans. Comput. Syst.* (2000).

[5] RIZZO, L. Netmap: a novel framework for fast packet I/O. In *Proceedings of USENIX ATC* (2012).