Rethinking Reliable Transport for the Datacenter

Mahesh Balakrishnan*, Joe Hoffert†, Ken Birman*, Douglas Schmidt†
{mahesh, ken}@cs.cornell.edu {jhoffert, schmidt}@dre.vandebilt.edu
* Cornell University, Ithaca, NY † Vanderbilt University, Nashville, TN

1 Introduction

Datacenter applications are inherently networked systems, distributed over multiple physical machines for scalability and availability. The predominant communication pattern within datacenters is *multicast*, where a packet is simultaneously sent to multiple nodes. However, the network communication options available today within datacenters are severely limited — typically, application developers choose between using TCP sockets or building custom application-level protocols over UDP. Research into high-performance protocols for clustered environments has failed to produce implementations that are used in the real world. We believe that the deployment failure of new transport options stems from their intrinsic design characteristics:

- Transport protocols are *generic* — each one tries to provide a one-size-fits-all solution for communication. For example, TCP provides exactly the same semantics for any application that uses it.
- Transport protocols are *monolithic* — each one has a completely different code-base, even if most of it is identical to existing protocols. For example, many TCP implementations differ only in the control loop they use.
- Transport protocols are *opaque* — they hide their internals from the applications that use them. For example, TCP exposes no information to the application regarding the status of an individual packet.
- Transport protocols are *static* — they stick to a fixed set of mechanisms, varying only the parameters to these using simple control loops. For example, TCP uses the same basic reliability and flow control devices in different settings, altering only the behavior of the window size curve dynamically.

In this white paper, we describe the design of Ricochet++, a new transport layer for datacenter applications. We argue that such a layer must have three major properties: modularity, gray-box design and adaptivity. Modularity provides the mechanism through which the layer can flexibly morph between different implementations and mechanisms on-the-fly. Gray-box design provides the mechanism through which the layer exposes information to external subsystems in an explicit and well-defined manner. These two properties combine to enable adaptivity — the behavior of the network layer can be determined by external decision-making tools which range in complexity from control loops to sophisticated machine learning algorithms.

2 Design Principles

Modularity: Two decades of research into reliable multicast has resulted in a plethora of different protocols — and the realization that no single solution works for all applications and workloads. At the heart of the many different reliable multicast protocols are a small number of core mechanisms [1] — positive and negative acknowledgments, sender or receiver retransmissions, and forward error correction. Prior work has looked at instantiating different multicast protocols using layerings of simple building blocks [3].

We extend the idea of modular protocols further by allowing arbitrary compositions of modules to interact in non-layered fashion; in a sense, we have multiple stacks existing in parallel. The data path between the sender and the receiver of a packet branches off into many different modules. For example, at the receiver an incoming packet could be sent to multiple modules — the NAK module, which generates a negative acknowledgment and despatches it back to the sender; the receiver-based FEC module, which creates an XOR from the packet to send to some other receiver; and the sender-based FEC module, which uses it to recover missing packets using XORs sent by the sender. These modules interact using a central eventing queue to obviate threading and locking inefficiencies, and use a managed memory subsystem to ensure that a single copy of the packet is safely shared across the different modules.

The key advantage of a modular network architecture is the ability to run multiple protocols in parallel — and to compare their relative performance in real-time. This opens the door to changing protocols on the fly depending on performance, whether through manual intervention or automatic/autonomic mechanisms.

Gray-Box Design: Network stacks have traditionally been closed 'black boxes'; the most notable example is TCP/IP, which provides almost no explicit feedback to the end application. Datacenter applications are usually high-performance systems written by intermediate or expert developers. Consequently, we believe that providing fine-grained feedback to applications on network conditions and protocol performance can have enormous advantages. For example, a replicated data store can use information about slow receivers to eject replicas from the group, or to shift load away from the problem receivers to healthy nodes.

A real-world example of how gray-box exposure can benefit system design is given by the SMFS mirroring solution [4] developed at Cornell. SMFS runs over a communication layer that uses sender-based Forward Error Correction for reliability — the sender generates and injects XORs of outgoing packets into the data stream, and the receiver can use these XORs to recover lost data packets. SMFS requires the communication layer to expose the number of XORs generated for any sent data packet. It uses this information to compute the expected 'reliability' of each data packet – the probability that it has been received successfully by the remote mirror. Accordingly, SMFS can then return from writes to the application once the expected reliability of the write is high enough, in contrast to waiting for the remote mirror to acknowledge the receipt of the packet. In practice, SMFS achieves orders of magnitude better performance compared to standard mirroring solutions, without sacrificing reliability.

Adaptivity: As datacenters get larger and more complex, the ability to selectively switch between protocols - and to dynamically parameterize them - becomes critical for high-performance applications. With the advent of utility computing platforms, applications are expected to run within highly virtualized containers, contending for processor and network resources. Protocol stacks for such settings will need to be highly adaptive, running different protocols in response to different conditions — for example, alternating between sender-based and receiver-based FEC based on which nodes are more heavily loaded; or using NAKs at high data rates and ACKs at low data rates.

There has been tremendous interest in recent times in the use of machine learning techniques to build adaptive systems. One approach involves measuring the performance of different 'solutions' and extrapolating the results — where a solution represents a particular configuration of the network stack. An implementation of such an approach would exist outside Ricochet++ and mandate the policy of adaptation. The mechanism of adaptation is provided by the first two properties of the framework — modularity and gray-box design. Modularity allows different protocols to be easily composed and instantiated on-the-fly, providing the mechanism through which a network subsystem can change from one protocol to the other. Gray-box design allows for the separation of this mechanism from the policy of adaptation — the decision to switch from one protocol to another can be made by an entirely separate system which uses gray-box interfaces to obtain performance information from the network subsystem.

3 Conclusion

We argue that a next-generation communication stack for datacenters must exhibit three key properties. First, it must be constructed in modular fashion, with different modules that compose together to form protocols. Second, it must expose gray-box information to applications, providing detailed information that can be used to infer protocol performance and network health. Third, it must use adaptive techniques to change its performance on the fly as system load and capacity fluctuate. Building a network layer with these properties is the first step to enabling truly adaptive datacenter systems. We are currently writing an implementation of the Ricochet++ system in C++ — the code is available at an open-source repository [2].

References

[1] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral error correction for time-critical multicast. In *NSDI*, 2007.

[2] M. Balakrishnan and J. Hoffert. Ricochet++ Open Source Repository, 2008. http://sourceforge.net/projects/ricochet-mcast/.

[3] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using ensemble. *Software–Practice and Experience*, 28(9), August 1998.

[4] H. Weatherspoon, L. Ganesh, T. Marian, M. Balakrishnan, and K. Birman. Smoke and mirrors: Shadowing files at a geographically remote location. In *Submission*.