eeRPC — Can Datacenter RPCs be General, Fast, and Secure?

Alex Litzenberger
alitzenb@andrew.cmu.edu

Varun Sharma
varunsha@andrew.cmu.com

Ankush Jain
ankushj@andrew.cmu.com

***Abstract*—We investigate how to provide high-performance encryption over stack bypass applications, using eRPC as our target application. We look at various design schemes possible, and describe how we integrated a low-overhead encryption scheme with eRPC, and look at its performance impact. We find that the overhead added by TLS to an HTTP call is roughly 8μs per request. We find that having encryption calls in the dispatch thread severely limits throughput, and one background thread is sufficient to generate enough AES throughput to saturate a 25 Gbps NIC.**

I. Introduction

High-performance network applications that rely on bypassing the kernel and directly accessing the NIC require security. Traditionally, adding secure communication to a networked application has been as easy as integrating TLS/IPSec, which provide a socket interface similar to a regular network socket. Not much work has been done in providing equivalent solutions for kernel-bypass networked applications, which do not want to offload encrypted communication to a regular kernel socket. Effectively, there is no performant out-of-the box solution that a stack-bypassing application could use.

We use eRPC as a target application to investigate design choices and costs of adding such a feature. eRPC is a highly-optimized RPC library that uses low-level, low-overhead interfaces provided by modern datacenter NICs.

We investigate different schemes for hardware encryption, and measure their performance. We look at eRPC's buffer management, and how to add additional calls in a manner that minimizes the buffer allocation and management overhead. We integrate an appropriate scheme into eRPC, measure its impact, and present our efforts in mitigating the costs of encryption.

II. Motivation & Background

RPCs are a fundamental building block for many networked applications. Therefore, optimizing the performance of RPCs helps a whole class of systems software take advantage of modern hardware capabilities, and keep up with increasing performance demands.

In an effort to meet these performance demands, focus has increasingly converged on specialized hardware and niche technologies, like RDMA, lossless networks, FPGAs etc. eRPC showed that a substantial amount of performance can still be extracted via convential networked applications, with a careful ground-up design. The initial versions of eRPC have left out many important features provided by standard RPC libraries, and focused solely on raw RPC throughput and minimizing network layer overheads.

eRPC is an excellent testbed for us to integrate an encryption layer to, and study its performance impact. We combine two different classes of pre-existing work:

1) 1 Secure transports and RPC libraries.
2) 2 High performance communication software with stack bypass.

A lot of pre-existing RPC schemes use SSL/TLS, like Microsoft RPC, gRPC, Consul etc.. Using TLS provides a tested, prepackaged solution that offers easy integration, secure key exchange, and message authentication, and modular support for different ciphers along with reasonably guaranteed updates that stay up-to-date as the threat environment evolves.

That's a very compelling set of features. However, using SSL/TLS requires the application developer to cede a lot of control to the library. Such a design is antithetical to the principles used in high performance, stack bypass network applications that aim to extract every ounce of throughput the NIC has to offer. *We find that TLS, on an average, adds 8μs of latency to each request, in addition to 10s of μs of kernel path overhead, which is unacceptable for eRPC-class systems.*

It is necessary for applications like eRPC to retain control of how the memory is allocated, how many times data is copied from the NIC receiving it to the multiple layers parsing it, to the application finally using it, how retransmissions and failures are managed etc.. We therefore felt it necessary to implement a custom secure transport that is deeply integrated with eRPC. We realize that such an implementation, without undergoing the rigorous security audits OpenSSL etc. do, would not be secure enough for production environments. It would, however, still help us understand the performance costs, and impact on throughput/latency.

We have analyzed the different solutions for encryption out there, including TLS and DTLS - the latter being a version of TLS optimized for datagrams. Other standards for RPC encryption like `RPCSEC` and `draft-cel-nfsv4-rpc-tls-01` (draft) exist, but they're broadly the same class of solutions and suffer from the same pitfalls as TLS.

III. Implementation

A. Components of an Encryption Scheme

End-to-end encryption schemes like TLS/IPSec usually have these four guarantees: