WIP/CFT: Packet Batching

BY TOM JONES AND JOHN BALDWIN

In the last 30 years, the computers we use have grown unimaginably faster. The 1995 Alpha AXP paper talked about designing for a machine's continuing the trend of the previous 25 years, getting 1000 times faster.

We have certainly managed to meet that goal, the 386 machines that were the original target of FreeBSD are akin to the micro controllers we use in keyboards today.

Even with these changes, the core of computer performance has remained the same, execute fewer instructions per unit of work, and things will go faster. This fundamental truth underlying networking has led to several different approaches to improving performance. We have worked on mechanisms that moved work away from our CPU and, instead, into the network card with checksum offload. If the card runs the instructions to checksum outgoing packets, then our precious CPU time can be spent doing other things.

Checksum offload saw great results, and we started to move other things away from the CPU and into the network interface. TCP Segment Offload (TSO) was the next great mechanism that improved performance for a network sender. Rather than forming the IP packets for the TCP segments we will send, we can form one template and send that with a large block of data to the card. The network interface handles the segmenting as it places the packets onto the wire. TSO gives huge benefits to a TCP sender, providing us the ability to saturate 10-Gigabit network interfaces well before we run out of even a single core.

TSO lets us be more efficient with precious resources. We reduce the number of bus (memory and PCI) transactions required to send each packet by batching them together and creating the final chunks at the point of transmission. This is straightforward for TCP to do, most of the time if we are bulk sending a stream of data and the chunking of data is clear. To mirror these improvements on the TCP receiver, we have Large Receive Offload (LRO). LRO lets us again reduce the number of transactions required to maintain high-rate data transfers.

FreeBSD has excellent support for TSO and LRO, but is lacking mechanisms similar to GSO and GRO.

For UDP, Linux has generic mechanisms that attempt to replicate TSO-like mechanisms. This support comes with Generic Segment Offload (GSO) and Generic Receive Offload (GRO). GSO enables huge improvements on the order or 20% for a UDP sender, GRO is more difficult to measure, but the mechanism is there.

FreeBSD has excellent support for TSO and LRO, but is lacking mechanisms similar to GSO and GRO. At EuroBSDCon in Vienna last year I spoke to John Baldwin about a mechanism similar to GRO that he is working on, which he calls Packet Batching.

WIP/CFT: Packet Batching

TJ: What is the background to the packet batching work?
JB: The idea of packet batching on receive has been around for a while, at least in the form of a wish list item I've heard various people mention several times. We already have some forms of packet batching specific to TCP for both sending (TSO) and receiving (LRO). This packet batching aims to be more generic than LRO so that it can apply to other protocols (primarily UDP).
TJ: Why is the work needed?
JB: The goal of packet batching approaches such as TSO and LRO is to amortize per-packet costs (various checks in the network stack on header fields, etc.) by doing them once per batch rather than once per packet. The cost of per-packet overheads becomes an increasingly worse problem as network speeds increase faster than CPU speeds. It is true that one of the fixes for this problem, in general, which does help with per-packet overhead, is horizontal scaling by using RSS to distribute packets across separate queues bound to different CPUs. However, you can't distribute a single flow across multiple cores, and batching schemes are aimed at making the performance of a single queue more efficient.
TJ: What new features/enhancements does the work make possible?
JB: The goal is higher PPS and/or reduced CPU usage for network received workloads. I don't expect it to help with TCP when LRO is enabled, mostly to help with UDP.
TJ: How can people test the work? Normally we need to emphasize testing with more diverse workloads, does this apply here?
JB: Benchmarking would be welcome. My initial set of simple benchmarks using iperf3 were mixed and not a clear enough win to justify the changes. The changes do add complexity, so it needs to be a clear win in some workloads, I think, before it should be considered a commit candidate. I have not observed any regressions in my benchmarks to date, just meager to zero gains.
TJ: How would you like feedback?
JB: E-mail directly to me is probably the best way to send feedback for now. At some point in the future, I will start a public RFC thread on net@ and/or arch@ at which point that thread will be the best place to send feedback. Folks wishing to test the patches or review them can find them at https://github.com/freebsd/freebsd-src/compare/main...bsdjhb:freebsd:cxgbe_batching.

From John's responses here, it isn't yet clear where the benefits should be seen. iperf3 measurements can't simulate the workload of a very busy server. For Packet Batching to offer a benefit in FreeBSD it is likely that more workloads need to be tested and tuned for. By pulling down John's github branch and experimenting with your network traffic, you can help establish a new receiver optimization in FreeBSD.

TOM JONES wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting.

JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for 20 years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger and LLVM. John lives in Concord, California, with his wife, Kimberly, and three children: Janelle, Evan, and Bella.