WIP/CFT: Socket Buffers

BY TOM JONES AND GLEB SMIRNOFF

Historically, the BSD network stack has had a generic implementation of socket buffers that are used both for TCP, UDP, local IPC sockets (aka UNIX sockets) and others. These buffers, of course, have some similarities — they buffer data, but they also have fundamental differences.

Some are remote and some are local. Some support data streams and others support datagrams. With the introduction of non-blocking sendfile in 2015, we came up with the notion of not-ready data in the stream send buffers. Then they were further complicated with the introduction of KTLS in 2017. At the same time, these buffers were still supporting UNIX control messages specified by POSIX. So, we got generic code that needed to support all possible features at once — and it got really complicated. It became fragile to changes, as changing a socket buffer to favor one protocol may affect the behavior of another. As an example, not-ready-data changes required a wide code sweep totally unrelated to **`sendfile`**, see git commits **`cfa6009e364`** and **`0f9d0a73a49`**.

In that last commit, note the final paragraph. SCTP already does its own socket buffers in parallel with the BSD part. (This implementation gave me a lot of insight for my current work.) Meanwhile, the perception of how much copy-and-paste is bad and how much is good in FreeBSD has changed over the decades. We have multiple device drivers that began as a paste of other drivers, but it was clear that differences accumulate, and it makes sense to have a paste to edit rather than to keep supporting two alike instances in one code. An example close to the socket layer is the two TCP stacks that are also maintained as two independent source files. To sum up, we no longer think that one code for all is a good idea.

SCTP already does its own socket buffers in parallel with the BSD part.

The socket code was difficult to attack at first, second, and third glances. If you look into current **`soreceive_generic()`** and **`sosend_generic()`** you will see why. However, after all this work, I came up with a plan that allows me to pick up a stick and leave the structure standing (https://en.wikipedia.org/wiki/Pick-up_sticks).

1) We have only two kinds of SOCK_DGRAM sockets: UNIX and UDP. Redefining just **`pru_sosend`** and **`pru_soreceive`**, we have a private code implementation for **`PF_UNIX/SOCK_DGRAM`**. See **`34649582462`** and **`e3fbbf965e9`**. This leaves **`PF_INET/SOCK_DGRAM`** aka UDP as the only datagram type that generic **`sockbuf`** code in **`uipc_socket.c`** supports.

WIP/CFT: Socket Buffers

2) **sockbuf** can be split into common parts that interact with event dispatching and private parts that do actual buffering. (See commits **a4fc41423f7** and **a7444f807ec**). This makes **PF_UNIX/SOCK_DGRAM** fully independent! This leaves **PF_INET/SOCK_DGRAM** aka **UDP** as the only datagram type that the legacy part of struct sockbuf needs to support.

3) Now we can branch off into improving **PF_UNIX/SOCK_DGRAM** before pulling other sticks from the pile.

There was a longstanding problem with one-to-many **unix/dgram** sockets when one writer could flood the socket and effectively DDoSing others. Here are our historical attempts: **2e89951b6f20** and **240d5a9b1ce76**. Let's make one-to-many sockets maintain a separate sub-buffer for every peer. See **458f475df8e**.

It is also possible to make a faster **unix/dgram**, e.g., using lockless queueing/dequeueing of data but I'm not doing it this time. Packets-per-second performance of **unix/dgram** isn't that critical for me.

4) Getting back to stick structure — **PF_INET/SOCK_DGRAM** aka **UDP** is the only datagram socket left with generic implementation, so let's make it private too. It's great that Robert Watson has already prepared two functions **sosend_dgram()** and **soreceive_dgram()** for UDP. **soreceive_dgram()** is not yet ready to be a full substitute for **soreceive_generic()**. Handling of complex cases with the help of **soreceive_generic()** needs to be fixed.

5) Now we can branch into UDP performance and maybe make it use **buf_ring(9)** instead of the linked **mbuf** list? Any takers for this task? We definitely care about pps performance for UDP, don't we?

6) With no datagram support left in **sosend_generic()** and **soreceive_generic()**, we can finally simplify them! Probably for the first time in history, these two monsters will shrink rather than grow.

7) This leaves **UNIX/STREAM** as the only socket type that is supported by the generic code and has control data. If it gains a private implementation, we can drop control data support from **sosend_generic()** and **soreceive_generic()**. At this point they will shrink even more!

8) We are getting really close to having TCP and SCTP being left alone. Note that there are also exotic sockets like **netgraph**, etc. Today, it is unclear what would be a better plan: Either

–1, to isolate TCP from generic, or

–2, to isolate everything else from generic and rename generic to TCP.

Either way the end goal is to have socket buffering for TCP and SCTP isolated so that our hands are untied for performance improvements without any risk of affecting anything else.

In D36002, Alexander Chernikov is now sharing his work for a NETLINK socket type. This socket may accumulate an internet full-view of the routing table which corresponds to hun-

There was a longstanding problem with one-to-many **unix/dgram** sockets.

WIP/CFT: Socket Buffers

dreds of megabytes of data that needs to be `read(2)` out of the kernel. The generic socket buffer implementation would require allocating that many `mbuf`s to hold the data. Such full-view retrieval may lead to `mbuf` shortage, a crucial resource on a router. But why are we using `mbuf`s here in the first place? We just need to copy data from kernel to userland. The new NETLINK will definitely benefit from a protocol specific socket buffer, that would copy data to userland I/O from its own specific data structure without any use of `mbuf`s.

How can people test the work?

The new implementation of the `PF_UNIX/SOCK_DGRAM` is already part of FreeBSD main branch. Any feedback or testing is appreciated, especially by people who have heavy `syslog(3)` traffic and had been affected by logging socket overflow problems.

Further plans are still work in progress. I usually share my work at https://github.com/glebius/FreeBSD when it is in an early stage and post it to https://reviews.FreeBSD.org when it is more mature. Comments are welcome there or via email.

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

GLEB SMIRNOFF first met FreeBSD when he was 17, and forever fell in love. He has worked in companies big and small, always looking for a job that allows him to contribute to open source. Now working with the Netflix OpenConnect team, he is saturating the Internet with traffic originating from unprecedentedly powerful FreeBSD boxes.

Write For Us!

Contact Jim Maurer with your article ideas.

(jmaurer@freebsdjournal.com)