INTERVIEW

Starting Firewall Development

AN INTERVIEW WITH IGOR OSTAPENKO BY TOM JONES

TJ: *There are numerous paths into working on FreeBSD, some through university courses and others through work experience. How did you learn about the project, and what initially drew you to Operating System development?*

IO: In the 1990s, during my school years, I had the chance to work with several programming languages and technologies, e.g., I spent hours tinkering with TR-DOS, carefully planning line numbers (which triggered flashbacks when I started using ipfw) and crafting DATAs and GOTOs for yet another game I envisioned, as well as experimenting with interrupt vectors for resident programs in MS-DOS. Because of that, a minimalistic command-line interface was not new to me. So, when I encountered FreeBSD during my Computer Science studies at university, my only question was which books to get.

It was clear that mastering FreeBSD would be challenging but incredibly rewarding, as I'd have to acquire some fundamental knowledge along the way. Why FreeBSD? The seniors promoted FreeBSD because our entire dorm network was built on it. This was the early 2000s, still carrying the inertia of the previous decade, when FreeBSD was the de facto standard for networking. In recent years, I've been focusing more on operating system development. With a broad background in software development, I've built a collection of ideas for how OS internals could be leveraged for high-level solutions.

“When I encountered FreeBSD during my Computer Science studies at university, my only question was which books to get.”

TJ: *What were your steps to making your first FreeBSD changes? How did you decide what to work on initially?*

IO: The first steps were about preparation. I wanted to refine my existing knowledge of FreeBSD, fill gaps, and develop a more structured vision. A well-known resource for this is *The Design and Implementation of the FreeBSD Operating System* book by Marshall Kirk McKusick, George V. Neville-Neil, and Robert N.M. Watson.

The FreeBSD source code was not entirely new to me; I already had a general sense of its structure formed over the years. However, I wanted a professional guide to ensure I did not miss key concepts, stylistic nuances, or structural elements. The world is lucky to have the FreeBSD Kernel Internals: An Intensive Code Walkthrough course by Marshall Kirk McKusick. It saved a lot of my time, covered all my questions, and provided valuable historical context to address the "Why?" questions in the best possible way. Additionally, the FreeBSD Networking from the Bottom Up course by George V. Neville-Neil offered further refinement of the networking stack side.

I considered the pros and cons of starting with big projects versus smaller ones and discussed it with mckusick@ and kib@. Konstantin Belousov recommended starting with a smaller task, such as bug fixing, which proved to be the most effective approach. I started working on the most recent pf bugs reported, which spawned other related improvements in the jail subsystem, test tools like Kyua's execenv=jail project, and even a new module, dummymbuf, for specific network tests. As a result, I continue to contribute to improving the project, in collaboration with Kristof Provost, Mark Johnston, and other FreeBSD developers.

TJ: *Bugs are a great way for a newcomer to start in any project. Do you have any recommended areas for new FreeBSD contributors to begin in 2025?*

IO: With official guidance and specific directions for new contributors available on the project website, e.g., https://wiki.freebsd.org/IdeasPage, I would like to consider an alternative approach. There are many possible paths to contributing, and the best ones are likely those that align with personal interests.

For example, if there is an interest in learning and using a FreeBSD network tool or kernel module such as netstat, route, pf, ipfw, netgraph, etc., then working with respective documentation and manual pages might open opportunities to improve them by providing extra examples, reworking complex concepts, or adding missing parts. If such a tool or module is not present in FreeBSD, then adding useful programs to the FreeBSD Ports Collection or keeping them up to date is another vital way to help. Such projects are usually fun and educational, as they may require a deeper understanding of FreeBSD kernel interfaces to succeed.

> "If the contribution is something more than a small patch, I recommend two initial steps: homework and communication."

If the goal is a deeper understanding of the kernel code, then a similar formula could be applied — choosing the topic that is in use or is planned to be used could provide more benefits. For instance, if it's a firewall, then a better understanding of how its rules work behind the scenes gives power users an edge. It could involve investigating the routing mechanisms if there is an interest in finding a solution for a non-generic problem. There might be a need to implement missing features or RFCs in the kernel. There is periodic interest in porting existing solutions from other platforms, such as Netlink implementation in FreeBSD, with the respective work-in-progress migration of the existing tools or Vector Packet Processing (VPP) framework porting, with open doors for further enhancements. Sooner or later, working with the existing code reveals opportunities for optimizations — spending less resources, such as time, per unit of data transferred or processed, benefits all businesses relying on FreeBSD.

If the contribution is something more than a small patch, I recommend two initial steps: homework and communication. There are many ways to reach FreeBSD developers (see https://www.freebsd.org/community/), and the basic one is the mailing lists like hackers@ FreeBSD.org. It's a good idea to discuss a potential project first to find a general agreement on the direction or to discover if someone is already working on it, as is usual for open-source projects; the more homework done, the better the communication outcome.

TJ: *The development process is quite daunting, and there are many dead ends. Are there any shortcuts you would like to share with new developers to make debugging and development easier?*

IO: I believe the *FreeBSD Journal* is an excellent source of shared professional experience. I recommend scanning the TOCs of the previous issues to find topics that fill gaps or offer a new perspective on a familiar subject. For instance, new developers may be interested in articles such as "Kernel Development Recipes" and "DeBUGGING the FreeBSD Kernel" by Mark Johnston [*FreeBSD Journal*, Sep/Oct 2021][*FreeBSD Journal*, Nov/Dec 2018], "FreeBSD Kernel Development Workflow" by Navdeep Parhar [*FreeBSD Journal*, Mar/Apr 2024], and "More Modern Kernel Debugging Tools," which you wrote [*FreeBSD Journal*, Mar/Apr 2024]. These and similar articles provide a quick overview of build system capabilities and shortcuts (such as avoiding lengthy rebuilds), or how virtualization and other 3rd-party software could be leveraged to improve productivity. Each article is not a complete book that covers all details or mentions all available options, but they do provide good starting points.

To begin forming good practices encouraged within the project, I recommend checking the article "Writing Good FreeBSD Commit Messages" by Ed Maste [*FreeBSD Journal*, Sep/Oct 2020]. Sooner or later, it's a good idea to become acquainted with the article "FreeBSD Code Review with git-arc" by John Baldwin [*FreeBSD Journal*, Sep/Oct 2021]. This tool significantly improves the process of patch publishing, reviewing, updating, and landing.

In terms of work on the networking part of the kernel, I suggest taking a closer look at FreeBSD's Jails and VNET features. If the work does not involve specific hardware support or other non-generic topics, then VNET-based

Jails can be leveraged to ease the testing of a new networking feature during development. To interest a new developer, VNET-based Jails can be roughly treated to build a network of lightweight virtual machines to test specific packet paths or network stack behavior. This approach can be easier than other methods, even if the same packet buffer (mbuf) may end up playing all the roles in the scene, since nothing leaves the host. Another important outcome of this exploration is that such a playground may become a good starting point for developing a new automated test for a newly implemented feature or a fixed bug. This leads to the article "The Automated Testing Framework" by Kristof Provost [*FreeBSD Journal*, Mar/Apr 2019], which introduces the FreeBSD Test Suite. The source of existing firewall tests based on VNET Jails (src/tests/sys/netpfil) and Kristof's talks [https://www.youtube.com/watch?v=gTyt7KLz1mw] about that can be used for inspiration.

Nevertheless, I would invest time in setting up work with the code. The kernel is an unusual type of software that supports multiple architectures, compilers, and exceptional cases, where the preprocessor's ifdefs may not be enough, and some conditions are resolved outside of the code itself. In addition, the source code includes some generated code, such as the boilerplate parts of system calls or VFS operations. Hence, greping or using an editor/IDE with default settings is not sufficient. This is a subjective topic, and my "Neovim + clangd + intercept-build" setup works fine for me, so it's better to get an overview of available options:

https://docs.freebsd.org/en/articles/freebsd-src-lsp/

Using Language Server Protocol greatly simplifies code comprehension and navigation, which is essential for a new developer.

TJ: *Thank you for your time for this interview. Is there any final advice to new contributors or anything you would like to add?*

IO: Thanks for your time as well. The general advice applies to any international open-source project with a considerable volunteer base: maintain an open-minded attitude and some level of patience. Let me add a networking-based allegory here. There are many other hosts (contributors) in the community network, and they are not available at any time using any protocol we prefer. Each host is likely busy with its workload, and its network interfaces may already be overloaded. So, a metaphorical connection timeout towards us doesn't necessarily mean a strict RST; more likely, it means that more time is needed to process our SYN. Communication approaches like email can provide deep buffering, which may be handled as first-in, last-considered, so a polite retransmission is often a practical step. Latency expectations also need reconsideration due to time synchronization issues (time zones), priorities (work over volunteering), or downtimes (weekends, vacations, etc.).

This network has been running for decades, with some hosts participating since the early days. Their accumulated knowledge and experience can help us with new challenges, guide us in case of dilemmas, or keep us away from issues not yet visible to us. At the same time, newly joined hosts may bring fresh ideas or new visions to consider. Each contributor brings something unique to complete or extend the whole puzzle. Hence, staying open-minded and striving to understand the intent or idea behind a message received should provide new contributors with better connectivity.

Although not mandatory or expected, leaning towards the symmetrical bandwidth of your host improves overall network capacity and fosters its expansion. In other words, be prepared to open some ports to accept incoming connections on your side.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast.

IGOR OSTAPENKO is a contributor to FreeBSD and OpenZFS with extensive software development experience across various domains, including systems for manipulating and testing navigation devices, enterprise solutions for optimizing business processes, reverse-engineering, and B2B/B2C startups.