BATMAN
THE BETTER APPROACH TO MOBILE AD-HOC NETWORKS

BY AYMERIC WIBO

In the expansive realm of network protocols, one stands out as a versatile and resilient contender: BATMAN, the Better Approach to Mobile Ad-hoc Networks. Through the airwaves of large cities, BATMAN allows devices to seamlessly communicate over a mesh network, without any one device requiring knowledge of the wider network topology.

Over the summer, I participated in Google Summer of Code (GSoC) where I ported the `batman-adv` kernel module—which provides support for the BATMAN protocol on Linux—to FreeBSD. This GSoC thing is a program where students are awarded a stipend by Google for working on an open source project over the summer, overseen by a mentor, who, in my case, was the one and only Mahdi (or Mehdi, depending on which day of the week you ask him) Mokhtari, aka mmokhi@. He's a classy guy, and I'm very grateful to have had him as a guiding hand throughout my GSoC journey!

Currently with `batman_adv` (FreeBSD's equivalent to `batman-adv`), you can create and participate in meshes and send/receive packets over Ethernet. This all also works from within the Linuxulator. The porting effort mostly consisted of work on the LinuxKPI (notably backing `struct net_device` by `struct ifnet` and `mbuf` by `sk_buff`), which should hopefully ease porting other networking-related drivers from Linux in the future.

> Through the airwaves of large cities, BATMAN allows devices to seamlessly communicate over a mesh network.

What Did the Porting Process Look Like?

Though I'm far from an expert in the ways of porting kernel components (this is the first time I've ported anything remotely as large as BATMAN), here's a high-level overview of what the process looked like for me—on the off-chance some insight may be gleaned from my little adventure.

The first step was—naturally—to pull in the code for `batman-adv` from Linux into `sys/contrib/dev/batman_adv` and to create the Makefile to build it, which contains a list of all the source files to use and which compile-time options to set, such as telling it to include the LinuxKPI stuff. As expected, things didn't compile on the first try; `batman-adv` calls a lot of functions and uses a lot of structures which only exist or make sense in the context of the Linux kernel. This is the raison d'être of the LinuxKPI; it provides a compatibility layer that implements a subset of the Linux kernel by calling to equivalent (or sometimes not-so-equivalent) functions in the FreeBSD kernel.

So, the next natural thing was to get things to compile by writing stubs for all the missing functions and structures in the LinuxKPI, filling in the missing fields as they're used (we can't copy the structures from Linux as-is, because Linux is GPL-licensed). These stubs just contain debug print statements which let us know when they're being used.

With everything compiled, I could load the kernel module, which immediately panicked the kernel. Then it was just the process of going through all the stubs which are being called, looking up and understanding their implementation in Linux, and implementing an equivalent for FreeBSD in the LinuxKPI, until the kernel not-panicked. This was maybe 70% of the work.

Once every operation (loading the module, creating interfaces, sending stuff over it, and so forth) works and doesn't blow up anything, it is time to close the curtains, open up the ol' Wireshark on a second monitor, and lock myself in my kot (== Belgian dorm room) for a week straight getting a) all the devices on the mesh to recognize each other, and b) data to actually go from device A to device C passing through device B without getting mangled or lost in the process. A lot of this time was spent revising the (sometimes insufficient) implementations made during the previous step. This was maybe 30% of the work, but it felt like 90%, with most of my time spent staring directly at Wireshark and it staring back at me.

With a stable ABI, a single device driver can be used with multiple different network stacks.

Finally, BATMAN worked on FreeBSD, and all that was left to do was make userland tools support manipulating BATMAN interfaces, write a few lines of documentation, and open the curtains.

Upstreaming batman_adv

At EuroBSDCon last year, I spoke to some members of core@ about the possibility of upstreaming `batman_adv`, which they'd like to avoid, as BATMAN is GPL-licensed. So it'll probably remain as a port forever because there isn't really a use case where BATMAN is necessary to get a working FreeBSD system; if you need to use BATMAN for something, you'll likely be in a position where it's easy to fetch and build the port yourself anyway (as opposed to NIC drivers e.g.).

What's Left to Do?

The big limitation of BATMAN on FreeBSD right now is that it can't participate in wireless networks. I fully intend on getting at least wireless working in the following year, as that's where BATMAN's major use case lies. Hopefully I'll get that done in time for BSDCan 2024 :)

I wholeheartedly recommend participation in GSoC to anyone who meets the requirements to apply. I went from not knowing much about FreeBSD's network stack and kernel to, well, still not knowing all that much about it in the grand scheme of things, but certainly knowing a lot more than when I started. It especially helped me become a lot more comfortable navigating through the source tree and debugging kernel panics, even not-net-work-code-related.

Further Reading

Here's a link to my GSoC project wiki page where all the specifics, code, and a small demo video are:
https://wiki.freebsd.org/SummerOfCode2023Projects/CallingTheBatmanFreeNetworksOnFreeBSD

And, you might find this link to the batman-adv overview interesting, as it goes more in detail on the BATMAN implementation on Linux (and thus also on FreeBSD):
https://www.open-mesh.org/projects/batman-adv/wiki/Wiki

AYMERIC WIBO is a CS student at UCLouvain in Belgium and has been using and developing projects based on FreeBSD since high school. His primary interests lie in graphics and networking.