Wifibox: An Embedded Virtualized Wireless Router

BY GÁBOR PÁLI

Due to changes in life priorities, I drifted away from FreeBSD for a few years around 2017. Later I returned and started building a new FreeBSD-based workstation for myself, a Lenovo ThinkPad X220. I noticed that although it was working, the wireless support was still far from optimal, the **`iwm`** driver was neither stable nor peformant enough for daily use.

I realized that in the wireless networking area, FreeBSD is still struggling to match the performance of Linux-based systems due to lack of up-to-date hardware support. This is happening for a reason: FreeBSD is often not considered a first-class citizen, hence it is not a target of such developments, and the respective portions of its networking subsystem need to be elevated to meet the latest requirements. This is not a trivial issue to fix, and the FreeBSD Foundation has been sponsoring a long-term project that aims to bring updates to the stack and establish a framework to facilitate re-using the wireless network card drivers from Linux.

In the wireless networking area, FreeBSD is still struggling to match the performance of Linux-based systems.

This began to bug me and I could not just wait patiently for the problem to get resolved. I wanted to be part of the development of FreeBSD again because I not only enjoyed using it but also learning about it. However, I did not have the luxury of investing time into mastering both the networking and driver code development in my free time, so I had to look for other opportunities.

Prototype

I was excited when another approach was brought to my attention (thank you to Gábor Zahemszky for that!). It was the idea of leveraging the PCI pass-through capabilities of **`bhyve`** to run a Linux guest, for which it becomes possible to talk to the real hardware, set up the wireless connection, and share the network with the FreeBSD host. I discovered David Schlachter's excellent blog post where the whole process is described in detail, and was able to build my own prototype with the help of that.

While experimenting with the whole process, I studied it from at least two perspectives. First, whether this is something that would be sustainable in the long run with regard to system upgrades, and second, whether this is something that users without a deep understanding of the solution could easily install and remove. As a former ports developer, I knew that I would have to be able to maintain the components and keep them isolated from the base system and somehow integrate with other third-parties. So why not exploit the existing ports framework for that purpose? And then the concept of the **`net/wifibox`** port was conceived in April 2021.

Initially, I used the **`sysutils/vm-bhyve`** port to build and manage a virtual machine that was based on Alpine Linux. Alpine is a lightweight Linux distribution that adopts OpenRC as the init system, uses the **`musl`** standard C library to make it possible to create small applications, and integrates BusyBox for the most commonly utilized command-line tools. Originally, it was created as an embedded-first distribution. I learned about it when I was working with Docker container images and remembered it for the small footprint and ease of use. It is actively maintained and provides a large number of packages which are managed through its "aports" system. In retrospect, the whole system bears resemblance to FreeBSD in many aspects, and I grew fond of it.

Although **`vm-bhyve`** is an excellent tool, I felt that it was too much for this specific use case. Instead, I used it as a model for the basic user interface, such as providing a console for the user to interact with the virtual machine hosted inside, and the elementary orchestration routines. Since these routines required interaction with the command-line **`bhyve`** tools, I decided to stick with the shell-based approach. I would probably not have had a better experience if I tried to implement all the plumbing in some other, higher-level language such as Python, as it would unnecessarily increase the build times and introduce a dependency on other third-party packages.

In the end, the user interface of Wifibox was composed of the **`start`**, **`stop`**, **`restart`**, **`status`**, and **`resume`** commands. The resume operation had to be handled specially because it was known that on suspending the notebook, the virtual machine loses its connection to the virtualized PCI device which has to be recovered somehow. After some experimentation, I noticed that this could be mitigated by stopping the virtual machine, reloading the **`vmm`** kernel module, and starting the VM again. There was recently a solution proposed by Joshua Rogers to fix the issue on the kernel level, but this has not yet been added to the base system.

Alpine is a lightweight Linux distribution that adopts OpenRC as the init system.

The VM's interaction with the PCI device has often proven to be a weak point, which often limits the usability of Wifibox itself. As an enhancement to this workaround, the repertoire of recovery methods have been expanded. Certain hardware configurations react differently to how the device is shut down and restarted. For example, thanks to Joshua's work, it was discovered that it matters if the guest itself shuts down the device properly during its own shutdown sequence. It was also learned that the **`ath11k_pci`** Linux kernel module does not well tolerate run in a virtualized environment, because it assumes that the location of the Message Signaled Interrupts (MSI) table matches with that of the host. This could only be handled if the FreeBSD host somehow supported injecting the host physical MSI information for the guest or disabling the MSI virtualization.

VirtFS/9P Support

One of the primary design principles of Wifibox was that users should not know about the underlying virtual machine, but be able to run it directly on the host as a local application. To create the illusion of that, the recently completed work around the VirtFS/9P file system pass-through support of **`bhyve`** was explored. By mounting the appropriate directories on the host for the **bhyve** guest, the required configuration files could be imported and the log files could be exported. This way, the user would not have to move or keep files in sync manually between the virtual machine and the FreeBSD host.

The VirtFS/9P support was made available beginning with FreeBSD 13, but I wanted to extend it to the older versions at that time, 12 and 11, to broaden its userbase. Fortunately, this feature is contained in a single module, and I was able to create another port, called **sysutils/bhyve+**, to automatically patch the **bhyve** sources in the base system to have this module included. With the help of that, there was no need to wait for the original authors to backport the feature, but this extra dependency could be pulled in for the **net/wifi-box** port when needed. Besides the addition of **virtfs-9p**, the **bhyve+** port included many other fixes that made it possible for Wifibox to run. Basically the goal was to put together a version of **bhyve** that could be the same for every major FreeBSD version and minimize the differences. This would have been based on the version in the 13.x line, but the related architectural changes made it non-trivial and the idea was later dropped. Over the years, its relevance has slowly faded away and it eventually became obsolete.

The creation of the disk image for the guest had to face many challenges. The primary concern was that in the beginning, the image itself was a pre-installed Alpine system. It was maintained locally on my workstation, it was hard to track what it contained, and it kept changing due to writes to the various temporary and work files. From the user's perspective, it raised the valid question of trusting "somebody else's VM." The initial versions were around 640 MBs, which looked gigantic compared to the ones that are typical for embedded systems. This size partially resulted from the image containing all possibly useful tools and files, so it was a logical next step make it smaller and more modular.

Version 1.0

The creation of the disk image for the guest had to face many challenges.

For version 1.0 in May 2022, a lot of effort went into reworking how the image was created. The prime directive was to make the whole process reproducible and lean. Technically, the complexity of installing the system components from scratch was translated to the port's **Makefile**. The image has gained its own sub-port, **net/wifibox-alpine**, while the orchestrator script was split into **net/wifibox-core**, and **net/wifi-box** has become a metaport. Through constant experimentation with the Alpine installation files, the root file system package, and its package manager, the **apk** tool, they were adapted to run atop FreeBSD with the help of the Linuxulator. In addition to the creation of a **Makefile** to drive the automated installation of the system to a designated directory on the host optionally extended with extra files, Alpine packages are downloaded and installed there. The packages themselves offered a way to modularize the construction of the image and make it possible for the user to select between them through the various port options. For example, the firmware files for each of the major wireless card brands could be separately installed and FreeBSD package flavors could be created for them.

The package-based approach lent itself to the creation of additional packages and the modification of the existing ones. Unfortunately, many of the upstream Alpine packages turned out to carry some extra weight, such as documentation or additional binaries, so they had to be removed. But it also allowed porting applications such as `mDNSResponder` to this platform and allowed them to run as part of the solution. The package for the Linux kernel itself had to be heavily edited to lose all the unused components, reduce its resource consumption, and shrink its attack surface. Wifibox does not need the standard `initrd`-based boot process, therefore the initial temporary root file system is completely removed and the boot happens directly with its root file system. Configuration files and patches for architectures other than AMD64 were removed, as Wifibox only supports that specific one.

The virtual machine image is compressed by SquashFS and keeps the overall size down to the ballpark of 15 MB. This approach also comes with a read-only root file system that prevents even the `root` user from tampering. It is expanded with a memory-backed temporary file system that is mounted under `/tmp` to manage the run-time file writes besides the VirtFS/9P mounts for the reading the application configuration files from the host. The boot process is run through GRUB, hence the Linux kernel (without its modules) is not part of this image, but pre-loaded with `sysutils/grub2-bhyve`.

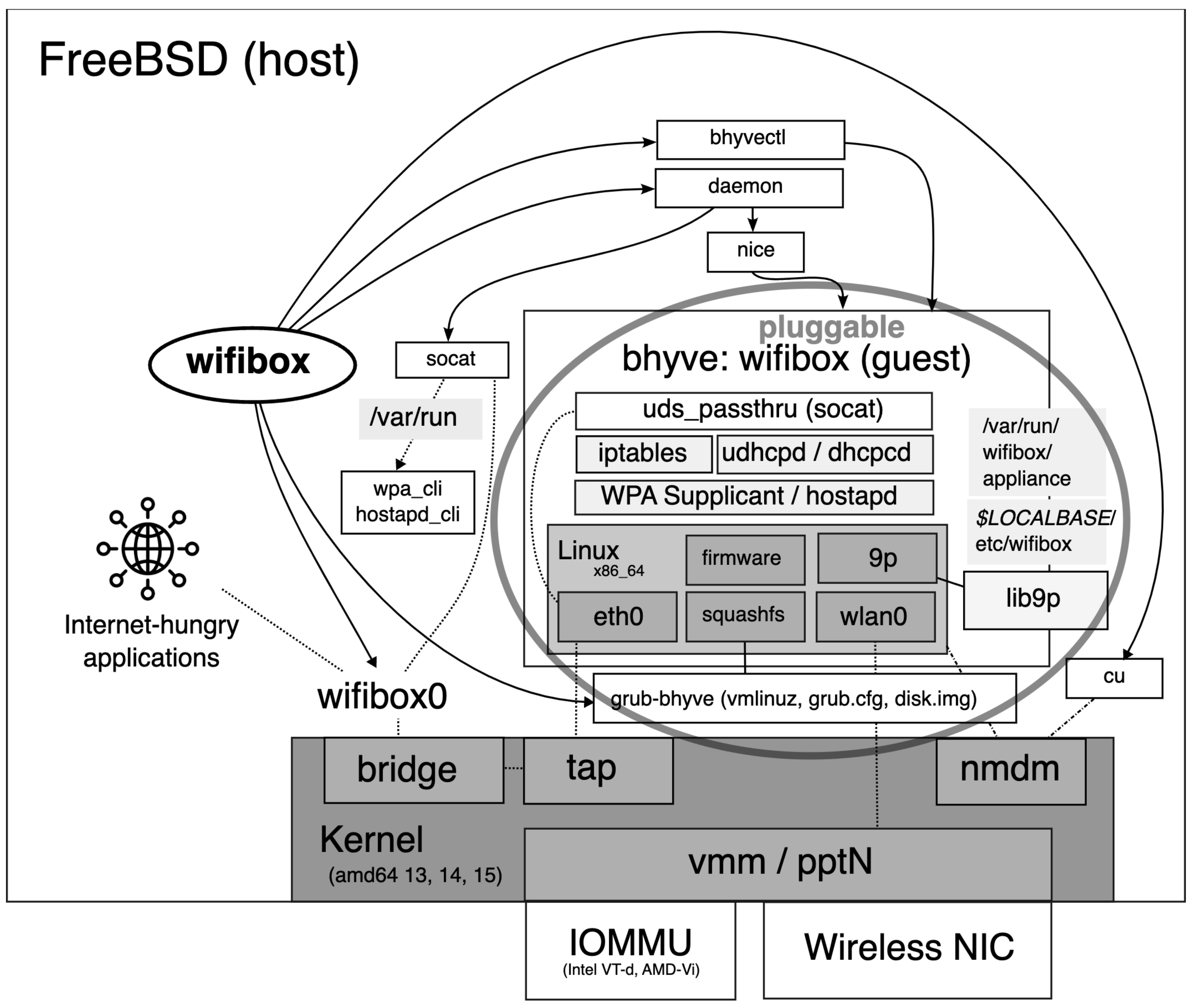
Package Framework

To roll out the required set of Wifibox packages in addition to the ones imported from upstream, the package framework of Alpine Linux has been adopted. For transparency and reproducibility, every customized package has its own `APKBUILD` file and extra files version-controlled in `git`. The packages are built on a clean, dedicated Alpine Linux `bhyve` virtual machine, often dubbed `wifibox-dev`, which is re-created for every minor Alpine release. The resulting packages are uploaded to GitHub for the user's convenience. In the past, there were experiments to build the packages in a Linux `chroot` environment, but FreeBSD's native Linux emulation support did not prove sufficient enough for this purpose. The results were similar in case of cross compilation, which is why I ended up with using `bhyve` for this as well. Per Bernhard Fröhlich's suggestion, I am currently looking into utilizing GitHub Actions to build the Wifibox packages automatically and independently in a native Linux environment.

The resulting packages are uploaded to GitHub for the user's convenience..

Linux-based Wireless Stack

A regular Linux-based wireless stack is operated inside the VM. First of all, the Linux kernel is used to detect the PCI wireless device and make it run through one of its drivers and the corresponding firmware, when necessary. The `wlan0` wireless networking interface is brought up by the standard OpenRC services. Then either WPA Supplicant or `hostapd` is hooked up on that to finalize the configuration. Next to the wireless interface, a virtual `eth0` Ethernet interface is exposed by `bhyve`. On the host, there is a bridge interface, `wifibox0` defined, which is joined with `eth0` in the guest through a `tap` software tunnel. Using `iptables`, there is Network Address Translation (NAT) and packet forwarding applied to make the traffic flow bidirectionally between `wlan0` and `eth0`. The IP addresses are obtained with the help of Busybox's built-in `udhcpd` for the host (over `eth0`) and either `dhcpcd` or `udhcpd` in the guest (over `wlan0`). The IP address range for the host can be controlled in the Wifibox configuration and adjusted according to the user's needs.



Due to the introduction of NAT, note that Wifibox uses a different range of IP address for `eth0` and `wlan0`. As a consequence, certain applications may not work properly out of the box, and the deployment of additional tools and extra configuration, e.g., port forwarding is required. This can be considered a benefit from the perspective of security, since there we have a firewall installed automatically. But this is equally a drawback, because it breaks the end-to-end connectivity, a core principle of the Internet. To overcome this, there were experiments to push down the packet forwarding to the level of Ethernet. For example, there is WLAN Kabel that implements moving packets between Ethernet and wireless interfaces. It was used with moderate success, since the traffic was flowing, but DHCP communication could not be made to pass through the barrier, and the observed performance was low. Nevertheless, that is a curious approach which is worth exploring further in the future.

An advantage of using Wifibox is the so-called "Unix domain socket pass-through", which helps tools like `wpa_cli` or `wpa_gui` communicate directly with the WPA Supplicant running inside the guest so that the user does not have to run them there and interact with the VM. That is extra functionality because Unix domain sockets on the guest file system are not exported by VirtFS/9P so the host will not see them. This is overcome by running a dedicated `uds_passthru` process in the guest, which runs a stripped-down version of `socat` in the background to convert the sockets to TCP ports. It has a pair on the side of the host, that communicates on those ports and translates the data back to a socket locally. With that in place, Wifibox can emulate the presence of that socket to implement smooth communication. In collaboration with the guest, the orchestrator script automatically manages the socket pass-through through a specific configuration file.

Wifibox as a Product

Although the Wifibox guest OS is mostly based on Alpine Linux, it often includes additional patches. For example, it imports a number of patches and packages from Arch Linux, because Arch has a better support for wireless devices. But it was also discovered that the driver for older Broadcom cards simply did not have support for MSI, which is a must for drivers that want to run in a virtualized environment. Slow initialization lead time of certain drivers can make the WPA Supplicant unable to find the `wlan0` interface on boot, so an exponential back off mechanism was implemented to enhance its resilience. As experience shows, a small operating system distribution dedicated to solving these issues is definitely warranted.

The orchestrator script makes it possible to combine `bhyve` with other tools to shape its behavior further. Thanks to Anton Saietskii's observations, `nice` was connected to assist with controlling the priority of the process that is responsible for running the virtual machine and avoid overloading the host. Similarly, a layer with `daemon` was added to monitor the status and revive the machine if it crashed or it was deliberately restarted.

As of the time of writing, Wifibox is actively maintained and it has been receiving semi-annual updates in March and September. With the help of Ashish Shukla, these releases are published to the FreeBSD Ports Collection, therefore they are available for installing with the `pkg` tool. Note that Wifibox is not featured on the installation media for the FreeBSD releases, which can make it harder to take advantage of it when one tries to install FreeBSD over a wireless network. However, it is possible to add all the required binary packages to the installation media and use them to initialize the network connection before starting the installation procedure.

Wifibox is actively maintained and it has been receiving semi-annual updates.

At the project's home page, both the source code and pointers to the respective GitHub repositories can be found, tickets can be opened, and discussions can be started. There is a separate repository for the ports themselves where the published development versions present an opportunity to take a peek into what is brewing next and test out fixes for issues.

Documentation

Wifibox is bundled with a lot of documentation, so I encourage the reader to explore it further. And I would like to emphasize an implicit but important organizing principle with regards to documentation. Wifibox follows a phased, "read as you go" model. This means that it has no extensive online documentation, the `README` file in the main GitHub repository covers an introduction, the basic installation instructions, and the list of hardware configurations that are known to be compatible. The user then has to install Wifibox to get access to the manual page, which has further details on how to use the tool and where the configuration files are. And then in the configuration files, additional instructions are provided on how to work ourselves through the related steps, together having a validation in place with helpful error messages to guide the user. The virtual machine image has its own dedicated manual page. Thanks to John Grafton, Warner Losh, and many other users for giving me feedback on how to improve on these.

Summary

This all is an interesting mix of product of curiosity, the desire to provide a quick remedy for the challenges in FreeBSD's wireless journey, and inventing another use of the technical advantages that are provided by PCI pass-through in virtualization and the design of `bhyve`.

Wifibox still has its own drawbacks, and it is nowhere near a drop-in replacement for the native solution. Hence, it is a called an embedded virtualized wireless router which removes the need to buy dedicated hardware and creatively presents the CPU's existing virtualization capabilities as an imitation of that. However, I believe this approach still has ideas to chase, such as using `bhyve` to run the Linux kernel and its drivers only in the virtual machine and expose that directly as a wireless networking device. This would bring the approach one step closer to the native solution, but it is not yet known if it is feasible and, yes, how much work it would require. In the meantime, I hope that Wifibox can alleviate the pressure around making the native solution production-ready and reassure users that FreeBSD is still a great choice nowadays and they do not have to necessarily give up on getting good speeds and reliable connections over wireless connections.

GÁBOR PÁLI has been a happy and committed FreeBSD user for many decades and he also has had the joy of being a documentation and ports developer. He lives on the edge of the beautiful Hungarian town of Esztergom with his wife. He endorses the adoption of functional programming in the industry and nowadays he contributes to Apache CouchDB where he can write Erlang and Scala code.