NixOS for FRC Coprocessors

FRC Team 3636 — Generals Robotics
Max Niederman, Class of 2024
April 17, 2024

Motivation

Year after year, the state of *FIRST* Robotics software has increased in sophistication, from the adoption of AprilTag pose estimation to the increased prevalence of on-the-fly trajectory generation and pathfinding.

This increase in sophistication has been accompanied by an increase in computation resource requirements — far beyond what the aging RoboRIO 2 platform can provide. This has led to an explosion in the number of coprocessors on FRC robots. It's not unusual to see a robot with half a dozen coprocessors handling various concerns.

Managing networks of coprocessors is difficult under any circumstances, but FRC presents a few additional challenges:

1. Coprocessors have **no Internet connectivity** while on a robot network.
2. Violent actuations and collisions with robots have the potential to inflict **physical damage** on coprocessor hardware.
3. The densely-packed nature of many FRC robots often make it **physically difficult to access** coprocessors, particularly in the short times between competition matches.
4. During competitions it is necessary to perform maintenance without advance warning and under extreme **time constraints**.

Because of the difficulty of physical access, the wide class of issues that disable coprocessors' SSH servers become very time-consuming to resolve.

Even with shell access, the lack of Internet connectivity often hinders maintenance by making it extremely difficult to install, update, or repair software. Furthermore, there is often a lack of *any* Internet connection to coprocessors during competitions, due to the difficulty of creating an Internet-connected Ethernet network in a pit.[1]

All of these challenges are, of course, further exacerbated by the stressful, time-constrained nature of FRC competitions.

We propose that NixOS, an immutable Linux distribution built on the principles of pure functional programming, can, in many cases, address and mitigate these issues significantly better than traditional Linux distributions such as Debian.

Declarative Deployments

Today, coprocessors in FRC are virtually always configured *imperatively*. Teams first install an operating system on their coprocessors using a monitor and peripherals, then they set up login credentials, SSH keys, etc. Finally, they install FRC-specific software such as a camera server or pathfinding offloading service. Team-specific software is most often installed and updated from source by copying from a local machine and then building on the coprocessor. This process is then manually repeated for each coprocessor.

This is not, however, the only way to deploy software. Tools such as Puppet, Chef, and Terraform popularized *declarative* configuration and deployment. Rather than manually running commands to configure the system and deploy to it, the administrator *declares* the software they want on the system and the configuration they want it to have and the system is automatically driven to that state.

The declarative system administration paradigm has numerous advantages: among other things,

- many systems can be managed together, and common configuration shared between them;

[1] Typically, cellular connections to mobile phones are the only available Internet connections, and few FRC teams have the time and expertise to use this as an uplink for a correctly-configured copper Ethernet network. Making this easier is a possible subject of future work.

- it doesn't matter what state the system is currently in (brand-new install, partially broken, etc.), deploying to it once will bring it to the desired state; and
- declarations can serve as documentation of how systems are configured, eliminating the need to dig through `/etc`.

NixOS

NixOS is one of the most powerful declarative deployment tools in existence. It uses Nix, a purely functional build system and package manager, to achieve *reproducible* and *immutable* deployments of entire Linux systems. Although this method has drawbacks, it also has many benefits.

First, by mounting most parts of the system read-only and creating new bootloader configurations for each deployed system configuration, NixOS achieves incredible resilience. Accidental misconfigurations can be rolled back by selecting the previous configuration during boot, and nothing short of erasing the root or EFI filesystem will permanently break a NixOS system. This means **teams don't need to worry about their troubleshooting steps (or software) breaking anything**.

Second, NixOS systems are *stateless*. Although Puppet, Chef, Terraform, et. al. attempt to hide the effects of state, traditional Linux distributions are essentially stateful and these tools cannot fully reset a system to a declared configuration. However, with NixOS, **teams get what they declare, nothing more, nothing less**. It is impossible to, for example, forget that some crucial setting was set imperatively and that the coprocessor won't work if you redeploy from a scratch install.

Third, NixOS can generate pre-configured SD card images, USB installer images, virtual machine images, Docker containers, and more from system configurations. If a coprocessor breaks, its entire state can be rebuilt from a NixOS configuration and flashed straight to an SD card. With NixOS, **provisioning a new coprocessor takes minutes, not hours**. Furthermore, the ability to generate VM images which behave exactly like physical coprocessors opens up the possibility to **faithfully simulate coprocessors on a developer machine**.

Fourth, there exists a rich ecosystem of packages and deployment tools for NixOS. Nixpkgs, NixOS's package repository, is the most comprehensive package repository in existence,[2] and NixOS comes built-in with tens of thousands of configuration options. Team 3636 has packaged a large variety of FRC software for Nix and NixOS, which **makes installing FRC software as easy as adding a line to a configuration file**.

```
{
  services.photonvision = {
    # enable the PhotonVision daemon
    enable = true;
    # open the firewall for the web UI
    openFirewall = true;
  };
}
```

Listing 1: Installing and daemonizing PhotonVision on a NixOS system.

Similarly, NixOS-based deployment tools like NixOps and Colmena simplify and automate the process of deploying to large numbers of coprocessors. They also make it easy to build the system configurations on an Internet-connected machine and then deploy later, mitigating the lack of Internet connection on coprocessors and allowing teams to **update and install software on coprocessors without ever disconnecting them from the robot network**.

A NixOS Distribution for FRC

FRC teams use a lot of specialty software (WPILib, PathPlanner, Choreo, PhotonVision, etc.), which isn't available in most distributions. For NixOS, we've worked to package a large selection of FRC software and upstream it into Nixpkgs. In the meantime, we're maintaining these packages in an FRC-specific package repository.

Currently, we support the following software on Linux and Darwin platforms with 64-bit x86 or ARM architectures:

Software	Package	NixOS Module
PhotonVision	❄ (upstream)	❄ (upstream)
AdvantageScope	✓ (in frc-nix)	N/A
Choreo	✓ (in frc-nix)	N/A
PathPlanner	✓ (in frc-nix)	N/A
PPLibCoprocessor	Planned	Planned
Glass	✓ (in frc-nix)	N/A
Shuffleboard	✓ (in frc-nix)	N/A
SmartDashboard	✓ (in frc-nix)	N/A

[2] As of writing, according to Repology's statistics.

DataLogTool	✓ (in frc-nix)	N/A
RoboRIOTeam-NumberSetter	✓ (in frc-nix)	N/A
SysID	✓ (in frc-nix)	N/A
OutlineViewer	✓ (in frc-nix)	N/A
PathWeaver	✓ (in frc-nix)	N/A
RobotBuilder	✓ (in frc-nix)	N/A

The distribution also contains tools for configuring networking settings for typical FRC use, generating installer ISOs, commonly-used administration tools, and more.