PC-BSD - Making FreeBSD on the Desktop a reality

Kris Moore
kris@pcbsd.org
PC-BSD Software – iXSystems

Abstract

FreeBSD has a reputation for its rock-solid reliability, and top-notch performance in the server world, but is noticeably absent when it comes to the vast market of desktop computing. Why is this? FreeBSD offers many, if not almost all of the same open-source packages and software that can be found in the more popular Linux desktop distributions, yet even with the speed and reliability FreeBSD offers, a relative few number of users are deploying it on their desktops.

In this presentation we will take a look at some of the reasons why FreeBSD has not been as widely adopted in the desktop market as it has on the server side. Several of the desktop weaknesses of FreeBSD will be shown, along with how we are trying to fix these short-comings through a desktop-centric version of FreeBSD, known as PC-BSD. We will also take a look at the package management system employed by all open-source operating systems alike, and some of the pitfalls it brings, which may hinder widespread desktop adoption.

1 Introduction

FreeBSD is well known for its performance and reliability. Many organizations recognize this and deploy it widespread across server rooms world-wide. However when it comes to the personal computer, or business workstation, FreeBSD is much more rarely found. Why is this? On the desktop side, FreeBSD offers many advantages to a user, such as the speed of the ULE scheduler, 3D acceleration with a variety of video cards, sound support, resistance to viruses, and a wide variety of software available through the ports system. With all these positives that a FreeBSD desktop can offer, we must first identify and figure out how to solve some of its shortcomings in order to make widespread FreeBSD on the desktop a reality.

2 Desktop Weaknesses

FreeBSD by itself lacks some features and abilities which make widespread desktop adoption possible. The first problem a potential desktop user will run into is the lack of an easy to use graphical installer. The FreeBSD sysinstall program, while a good installer for a server-based system, can be quite a learning challenge for a desktop user, who simply needs to get a system set up and running with a minimal amount of effort or discomfort.

After a potential desktop user has finished a basic install of FreeBSD, they will then run into a new series of challenges required to set up a functioning desktop. A user will now find it necessary to install and configure Xorg, and an associated desktop environment. This will require knowledge of the FreeBSD ports system, or pkg_add command, plus the work of some text editor such as vi, pico, or emacs to configure various settings and files, such as /etc/X11/xorg.conf, /etc/ttys, and others. Along with these challenges, the user may also need to configure support for their sound-card, network, or wireless devices. For a new user, who simply wants a desktop to surf the net, check e-mail, and run productivity applications, this can be a real challenge, and one which a user will likely not see through to completion.

If a user has made it this far, and achieved a working desktop system, then there is one last hurdle to overcome; how to maintain it? Desktop programs often have updates for security issues, or new versions are released

with features that the user wants to utilize. Now the user has to consider the task of how to apply these updates. They may need to re-compile an application from the ports tree, or if they had help setting up the system, call their support personnel to perform what should have been a simple program update.

These issues alone are enough to scare off the vast majority of desktop users from running FreeBSD, and enjoying the other positives that it may offer. If these problems were solved, we could easily expect to see FreeBSD become a serious contender in the desktop arena, as it already is in the server market. The desktop success of Mac OS X is a good example of this principle. If the system can be made easy to load, easy to manage and easy to maintain, then more and more users be willing to make the switch, often without even knowing or caring which kernel is under the hood, or what various ports such as the xorg-server is or does.

3 Addressing the weaknesses through PC-BSD

While the hurdles to mainstream adoption of FreeBSD on the desktop are large, we are attempting to solve them through PC-BSD. PC-BSD is a desktop-centric version of FreeBSD, which is not a fork, rather a bundling of various GUI's, packages and tools, designed to solve these problems, and make FreeBSD on the desktop easy enough for widespread adoption. The latest version as of this writing is based on FreeBSD 7.1-STABLE, and provides a fully-functional KDE 4.2 desktop “out of box”. First, lets take a look at the installer, which is the first and most important step to getting a desktop deployed.

3.1 Graphical System Installer

In order to solve the installation dilemma, PC-BSD includes its own, custom graphical installation solution. This GUI is written in C++, using QT 4 from Nokia[1]. In addition to the new GUI installer, the installation can now be performed from DVD, CD, USB, or network. Let us take a closer look at some of the internals of PC-BSD's installation system.

The PC-BSD install media was originally based on the Freesbie[2] live CD creation scripts, which have since been changed to support other installation mediums, such as USB. The boot process is standard FreeBSD, and then uses a small mfsroot file system specified in /boot/loader.conf

```
mfsroot_load="YES"
mfsroot_type="mfs_root"
mfsroot_name="/boot/mfsroot"
```

The mfsroot image contains a small FreeBSD world environment and a customized /etc/rc script which runs after the kernel load. The /etc/rc script first runs a loop, to locate the installation media, and mount it to /mnt:

```
CDDEVS="/dev/acd0 /dev/acd1
/dev/acd2 /dev/cd0 /dev/cd1
/dev/cd2"

for i in $CDDEVS
do
  # Find the CD Device
  /sbin/mount_cd9660 $i /mnt

  if [ -e "/mnt/uzip/usr.uzip" ]
  then
   FOUND=1
   CDDEV="$i"
   break
  else
   /sbin/umount /mnt
  fi
done
```

After locating the install media, the rc script then creates a ramdisk and mounts it to /uzip, in order to copy the usr.uzip file from the media into memory. The usr.uzip file contains the graphical installer, Xorg files, and other data required to bring up the graphical interface. By copying this file into memory, the script is then able to unmount the CD/DVD media and run entirely from RAM. This ramdisk usage allows the user to switch discs as the need arises. After the /etc/rc script completes, the system boots normally and the user is presented with a GUI, that guides the user through the installation.

Figure 1: The PC-BSD Installer Wizard

Once the user has selected their install options, such as locale, user names, and disk options, then the GUI begins the installation process. This process is relatively simple, with the entire PC-BSD operating system stored as a LZMA compressed tar archive. After mounting the formatted partition to /mnt, this archive is extracted to disk, user accounts are created, and then the system and KDE locales are configured. After this process is finished, the CD/DVD is ejected, and the user is prompted to reboot the system. At this point the installation is finished, and a fully-functional graphical desktop is ready to be booted for the first time. This entire process normally takes less than 20 minutes, and can be completed by users with minimal computing knowledge.

3.2 Configuration Tools

Once the user has finished the initial install of the system, there are often more things which need to be configured before an optimal desktop can be achieved. PC-BSD includes support for detecting most common sound-card hardware automatically, however for the video setup and networking, the user often will want to specify their own options. We will take a look at two of these tools, for display configuration and network setup.

Upon the first boot of a freshly installed PC-BSD system, the user will be brought up to a graphical tool, which allows the configuration of the display settings.

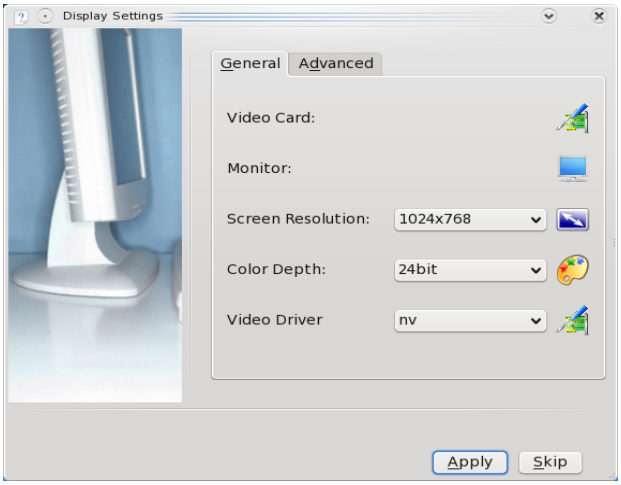

Figure 2: The X Setup Tool

This Xorg setup GUI, is possibly the most critical setup tool for a new desktop user making the switchover to BSD. Installing a system, and having it boot to a console window is one of the fastest ways to send a new user back to the comfort of Windows or Macintosh.

With this in mind, the “Display Settings” tool was created, to allow the user to easily configure their screen resolution, drivers, and color depth, with minimal difficulty. Under the “Advanced” tab, the user can also provide refresh rate information, or enable dual-head support with the click of a mouse. The tool also requires that a user “test” their chosen configuration before loading the desktop. This helps to ensure that the selected options do display properly, and the user is pleased with them, before loading the desktop. Should the user switch monitors, or video cards, the “Display Settings” tool is easy to recall during the system boot. By selecting option 7 “Run the Display Setup Wizard” from the loader splash screen, the Xorg configuration tool will be run again, allowing new tweaking of any display settings.

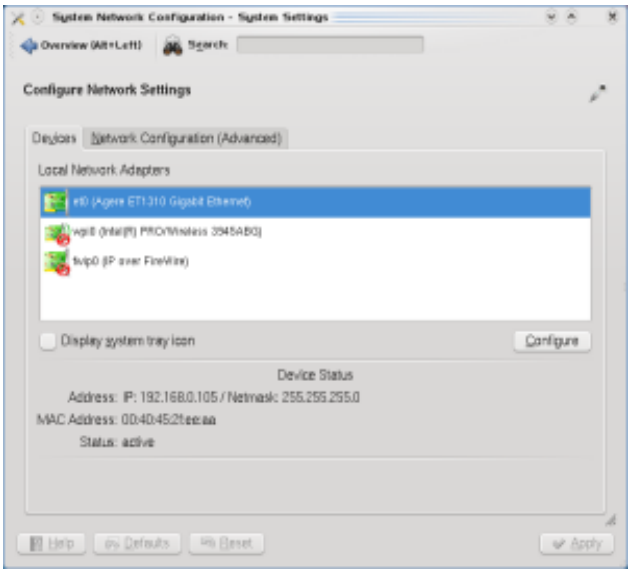

Figure 3: The Network Device Manager

In addition to tools such as an Xorg setup GUI being necessary, other utilities for setting up services such as networking are vitally important to the desktop user. When running standard FreeBSD, networking is normally set up by editing /etc/rc.conf by hand, or if using a wireless device, by editing the /etc/wpa_supplicant.conf file. While this does allow greater flexibility in configuration, it is often beyond the capability of a traditional desktop user, who is used to clicking on a few buttons in order to establish a working network connection.

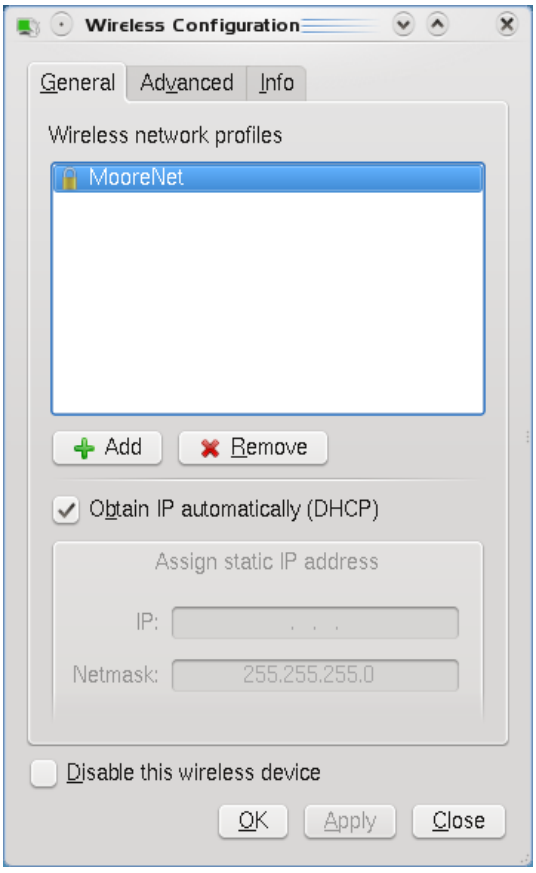

Figure 4: The Wireless Profile Manager

To solve this dilemma, PC-BSD includes an extensive network configuration toolset, for both wired and wireless devices. These tools act as a easy-to-use front-end to the /etc/rc.conf and wpa_supplicant.conf files. The network manager behaves simply as a desktop user would expect, with a listing of all the network devices on the system, and configuration options for each.

For users on laptops or who are connecting via wireless, an extensive

front-end to wpa_supplicant.conf is also provided. This utility provides support for multiple wireless profiles simultaneously, which allows the user to roam with their laptop, and connect to the best available network. Security options are also supported, such as WEP, WPA-P, and WPA-E. For monitoring of the wireless connection strength, a small tray application also is available.

3.3 Base System Modifications

Even though PC-BSD is one hundred percent FreeBSD under the hood, it does include some modifications to the installed system in order to be more “desktop-ready”. Starting in PC-BSD 7.1, the included KDE4 desktop and related packages have been moved from the /usr/local directory, and into the /PCBSD/local directory by modifying the LOCALBASE variable at build time. This change allows the user to install both PBIs and FreeBSD ports, all without touching or possibly breaking their installed desktop. Also, this modification enables online updates to be applied selectively to the /PCBSD/local base, without corrupting a user's installed ports in /usr/local.

This change to using /PCBSD/local for installed ports is done by setting a few options during the build of PC-BSD.

Options for /etc/make.conf

```
X11BASE=/PCBSD/local
LOCALBASE=/PCBSD/local
```

Environment Variables

```
PKG_DBDIR="/PCBSD/var/db"
export PKG_DBDIR
PORT_DBDIR="/PCBSD/var/db"
export PORT_DBDIR
```

These options are set before compiling the PC-BSD base set of packages, such as Xorg, KDE4, HAL, and others. After the build is complete, /usr/local is left empty, and running the pkg_info command shows no ports installed on the user's system. This change provides additional freedom for advanced users to work with the FreeBSD ports tree, without fear of conflicts with the installed PC-BSD base packages.

4 PBI – A desktop-friendly packaging system

A desktop which is easy to install and manage is a good start, but without the ability for a user to easily install applications it becomes meaningless. To solve this problem, PC-BSD introduced a new package management system known as PBI (Push Button Installer), which is quite different than traditional methods of software disbursement on other open-source operating systems. First, we will take a look at the concept differences between the PBI and traditional packaging system.

4.1 Design Differences

At its core, a PBI software package is built upon some very different premises than a traditional port, RPM, or deb file. When installing software on a Linux system, or FreeBSD using the ports tree, each program is dependent upon several things in order to function. First, each application is dependent upon the the base operating system itself to function, such as the kernel, some basic libraries such as libc, and a few others. These are fairly consistent on a system, and usually only change between major system updates, such as moving from FreeBSD 6.x to 7.x.

The next step up from the base operating system is the “application layer”, which consists of software

packages, shared libraries, and data. On traditional FreeBSD or Linux, packages are installed in this layer, with a system of inter-dependencies upon other packages. For an example, lets take a look at a visual diagram.

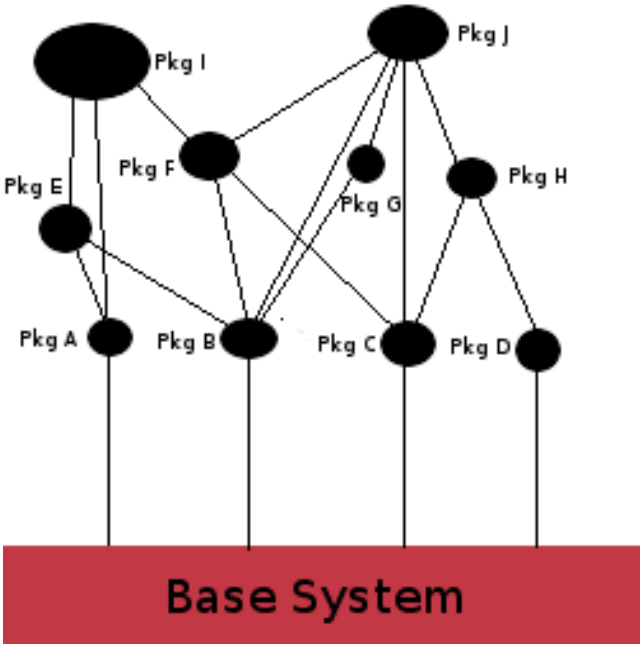

Figure 5: Traditional Package Management Model

In this simplified example, we see the base operating system along with a series of installed packages. At the first layer, Pkg A through Pkg D all depend solely upon the base system to function. However, as you move up the tree to Pkg E through Pkg J, you'll notice that they depend upon the presence of one, or several other packages to function. Should any package at a lower level be removed, other software which depends upon it will cease to function. By the same token, should the user wish to update a package at the top of the tree, it may require an updated version of a package lower down, which in turn may require the updating of other packages which depend upon it.

While this method of package management does have some advantages such as saving disk space, it can also have many drawbacks for the average desktop user. A user will often not realize that the desktop itself, such as KDE or Gnome, is apart of this “package tree”. After his system is installed and the user decides to update to the latest version of a particular application, he could begin to run into problems. The new application may in turn require updated versions of libraries further down the tree. These updated libraries may or may not be apart of his desktop interface, and have the potential of causing breakage, or requiring an update of some (seemingly) unrelated package to function. In the strict sense, a user's desktop interface is simply another piece of software on the system, and may be subject to updates or can be broken simply through the process of installing or updating any third party application.

Most modern open-source desktops try to compensate for this design by implementing powerful software managers which keep track of all packages and the inner-connecting dependencies they require. Thus, when a user requests to install or update a software package, the manager will map out all the required dependencies, and try to warn the user of any potential conflicts. While these managers are improving in their ability to monitor changes, fix conflicts, and prevent breaks, they still do not change the underlying design of the system. There still may be ten, twenty or even one hundred potential points of failure in the process of trying to install an application, depending upon the size and scope of the application's dependency tree.

With the introduction of the PBI format, the PC-BSD system has implemented a new method of package management, which attempts to reduce the potential for failure. The PBI format does this through the packaging of applications, complete with their required data and libraries together into a self-contained directory. This structure keeps applications separate

from the desktop and base system itself. By using this method, there are a few advantages and disadvantages.

The main disadvantage is that applications will require a bit more disk space, since they each include the required libraries within the package. However, this is quickly becoming less of an issue, since storage capability is expanding at an exponential rate. The same disadvantage may apply to downloading of applications, since a program like FireFox may be a few megabytes larger in PBI form, vs a traditional package with no included libraries. However, this also is becoming a lesser issue as time goes on, with the advent of high-speed connections becoming faster and more common place.

Even with the disadvantages, as increasingly irrelevant as they are becoming, the advantages to the desktop user are easily apparent. First, by including the dependencies within each package, the applications themselves no longer have multiple points of failure with which to complicate an install. A user need not worry about which packages are installed on their system, such as the versions of KDE, QT, or GTK. By the same token, now the user can freely remove or update an installed PBI, without the potential of a failure in their desktop, or some other seemingly unrelated application. This provides a level of reliability that desktop users want and need in their systems. The average desktop user or business does not want to waste time, manpower, or money resolving potential software conflicts.

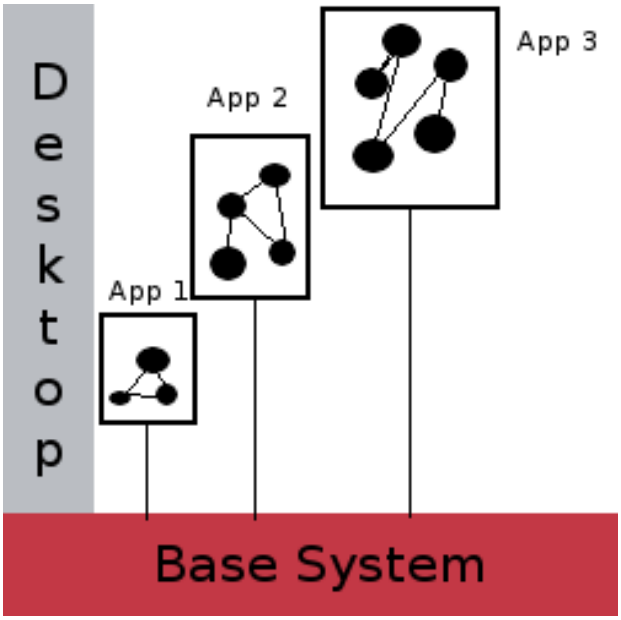

Figure 6: The PBI Application Model

4.2 PBI in practice

Now that we've taken a look at the design differences between a traditional package and a PBI, let us now explore how a PBI is built in order to achieve containment. Starting with the introduction of PBI Schema v2 in December of 2008, building a PBI from ports is easier, and more reliable than ever. By using the PBI Builder Software[3], a target port is compiled within a chroot environment, with a custom LOCALBASE option that will determine the location of the installed PBI on the client system.

/etc/make.conf in chroot build

```
LOCALBASE=/Programs/FireFox3.0.5
```

By setting our port make options to this target directory, all binaries and libraries are compiled to link back to this directory for their execution. After the port make and install are finished, the PBI Builder then reads through a developer-provided configuration module[4] to determine which files need to be copied and placed into the compressed PBI archive. There usually are a very small number of files

necessary for program execution, since a majority of the installed data is only required for the compiling phase of the port make.

After the program data is accounted for, the PBI Builder then proceeds to include some module-provided configuration data. This data may include desktop or start menu icon configurations, mime-type association of files, custom installation graphics, and setup / removal scripts. Once this configuration data has been copied, the entire archive is compressed using tar and lzma. This archive is then appended to a small binary loader and second tar archive which contains the name, version, and other details about the final application. Once this is complete, the PBI is now ready to be installed.

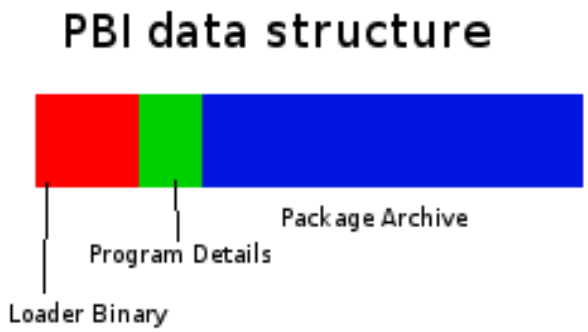

Figure 7: The PBI file layout

When the user clicks, or "runs" the PBI, the loader binary extracts the program details, prompts to switch to root for the install, parses any command-line flags, and then brings up the appropriate installation wizard. A PBI file includes both a command-line mode along with the graphical installation wizard. After the user confirms a few choices, the installer then decompresses the included package archive, usually to /Programs/<PBI Name + PBI Version>. If the user selected a different installation location, the installer will extract it to the appropriate location, and then create a symlink back to /Programs, ensuring that the application's internal linking is still valid.

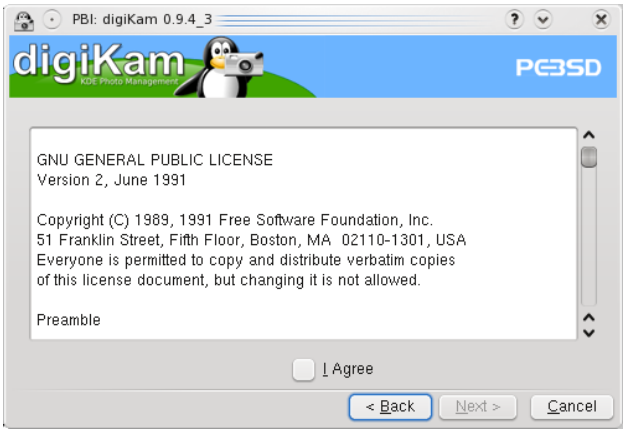

Figure 8: The PBI Graphical Install Wizard

After this installation is finished, the program is now ready to be run. The user may simply click the created program icon, or run it via the command-line if the installed application includes a command-line functionality. Should the user wish to remove this PBI, it may be easily accomplished through the Software Manager, or via the command-line by using "pbidelete". The removal tool then deletes the program data, cleans up any created icons, and runs any developer provided uninstall scripts. Often these scripts exist simply to ask if the user wishes to remove personal data created by the application.

5 Summary

We have taken an in-depth look at some of the challenges to making FreeBSD on the desktop a reality, including the steps PC-BSD has taken to accomplish this goal. While these enhancements to FreeBSD by themselves may only seem to be small improvements, together they form the basis of a easy-to-use desktop operating system, which has the potential to greatly increase the number of systems on which FreeBSD is deployed. By introducing the PBI package management system in PC-BSD, our goal is to be accessible to those who may have never touched an open-source system before, as well as provide convenience to technical users who appreciate the simplicity and reliability that it offers.

1 http://www.qtsoftware.com/
2 http://www.freesbie.org/ or in ports: sysutils/freesbie/
3 http://www.pcbsd.org/content/view/45/30/
4 http://trac.pcbsd.org/browser/pbibuild/modules/