Tux on the Air: The State of Linux Wireless Networking

John W. Linville
Red Hat, Inc.
linville@redhat.com

Abstract

"They just want their hardware to work," said Jeff Garzik in his assessment on the state of Linux wireless networking in early 2006. Since then, more and more of "them" have their wish. Lots of hardware works, and most users have little or no trouble using their Linux laptops at their favorite cafe or hotspot. Wireless networking no longer tops the list of complaints about Linux. Of course, some problems persist... and new things are on the horizon.

This paper will discuss the current state of Linux wireless networking, mostly from a kernel perspective. We will explore where we are, some of how we got here, and a little of why things are the way they are. We will also preview some of what is to come and how we plan to get there.

1 Where have we been?

The original wireless LAN devices were what are now called "full MAC" devices. These devices go to a great deal of effort to present themselves to the host processor very much like an ethernet device. They tend to have large firmware images that create that illusion, and only add the requirement of configuring parameters particular to dealing with wireless LANs such as specifying a Service Set IDentifier (SSID).

Devices with "full MAC" designs are not particularly prone to being forced into modes that do not comply with governmental regulations. As a result, vendors of such devices tend to be more open to supporting open source drivers. Examples include the old Prism2 and the early Prism54 devices, which are well supported by drivers in the Linux kernel. Unfortunately, the requirement for large firmware images tends to increase the costs associated with producing this kind of device.

Newer consumer-oriented wireless LAN devices tend to utilize what is known as a "half MAC" or "soft MAC" design. These devices minimize the work done using firmware on the devices themselves. Instead, only critical functions are performed by the device firmware, and higher functions like connection management are transfered to the host processor. This solves problems for hardware manufacturers, but makes life more difficult for open source software in more ways than one.

The chief problem created by the shift to "soft MAC" designs is the need for software to perform those functions on the host processor that had previously been performed by firmware on the wireless LAN device. The early Intel Centrino wireless drivers used a component called "ieee80211" to perform these functions. The `ieee80211` component used code adapted from the earlier `hostapd` driver for Prism2 devices.

Unfortunately, the early Centrino wireless hardware designs were not sufficiently general to apply the `ieee80211` code directly to other drivers. In response to that need, Johannes Berg developed an extension to that code to support true "soft MAC" designs. The original `bcm43xx` and `zd1211rw` drivers used this "ieee80211softmac" code successfully, as did a few more drivers that never got merged into the mainline Linux kernel. However, many developers (including Johannes Berg) felt that this combination of code was poorly integrated and the source of many problems.

About this time, Devicescape released source code to what would eventually become the `mac80211` component. Most of the active wireless developers rallied around this code and so all new development for "soft MAC" devices was shifted to this codebase. Unfortunately, many core Linux networking developers quickly identified systemic problems with the code from Devicescape. Thankfully, Jiri Benc adopted the Devicescape code and began working to resolve these problems. After many months, a great deal of code pruning, and some help from Michael Wu, Johannes Berg, and others, the new `mac80211` component was initially merged into the Linux kernel version 2.6.22.

The remaining drivers were merged later, and finally there was much rejoicing.

Now things are much better. Of course, the Linux kernel continues to support "full MAC" designs, and now it has infrastructure to support "soft MAC" devices as well. Thanks to this new infrastructure, it is possible to add new features to a whole set of related drivers with a minimal set of changes. Linux is well on its way to being a first-class platform for all forms of wireless LAN networking.

2 Where are we now?

The past is the past. What is the situation now? What drivers are available? Which ones are coming soon? How fast is wireless LAN development progressing? Where can I get the latest code? Where can I find current information? And what is coming next?

2.1 Driver status

Several wireless LAN drivers have been developed and merged into the mainstream kernel over the past few years. Still more are currently under development. Sadly, a few may never appear.

2.1.1 Current Drivers

Table 1 represents the 802.11 wireless LAN drivers available in the 2.6.25 kernel which were originally merged in 2006 or later. As you can see, nearly all of them are based upon the `mac80211` infrastructure. The notable exception is `libertas`. That driver was developed for use in the One Laptop Per Child's XO laptop, which relied on extensive firmware both for power management and for implementing a pre-standard version of mesh networking. The other exception is `rndis_wlan`, which is primarily an extension of the existing `rndis_host` ethernet driver to also support configuration of wireless devices which implement the RNDIS standard. Aside from these two, all of the other drivers are for "soft MAC" devices.

All of these drivers support infrastructure mode (a.k.a. "managed mode") and most of them support IBSS mode (a.k.a. "ad-hoc mode") as well. These drivers can reasonably be expected to work well with NetworkManager and similar applications as well as the traditional wireless configuration tools (e.g., `iwconfig`).

Driver	Hardware Vendor	Uses mac80211?
`adm8211`	ADMTek	Y
`ath5k`	Atheros	Y
`b43`	Broadcom	Y
`b43legacy`	Broadcom	Y
`iwl3945`	Intel	Y
`iwl4965`	Intel	Y
`libertas`	Marvell	N
`p54pci`	Intersil	Y
`p54usb`	Intersil	Y
`rndis_wlan`	Various	N
`rt2400pci`	Ralink	Y
`rt2500pci`	Ralink	Y
`rt2500usb`	Ralink	Y
`rt61pci`	Ralink	Y
`rt73usb`	Ralink	Y
`rtl8180`	Realtek	Y
`rtl8187`	Realtek	Y
`zd1211rw`	ZyDAS	Y

Table 1: New drivers since 2005

2.1.2 Drivers In Progress

A number of drivers have been started but are not yet completed or mergeable for one reason or another. Reasons include questionable reverse-engineering practices, incomplete specifications, or simply a lack of developers working on producing a mergeable driver.

The `tiacx` driver for the Texas Instruments ACX100 chipset has been around for a long time. At one time this driver was working well and had been successfully ported to the `mac80211` infrastructure. Unfortunately, questions were raised about the reverse engineering process used to create the initial version of this driver. It is possible that TI could clarify this driver's legal status. Otherwise, work similar to that done by the SFLC to verify the provenance of the `ath5k` driver will be required to remove the legal clouds currently preventing this driver from being merged into the mainstream Linux kernel.

The `agnx` driver for the Airgo chipsets[1] is based upon a set of reverse-engineered specifications[2] as well. To date, this team has maintained a rigid separation between reverse engineers and driver authors. This is the same technique used with good results to implement the `b43` and `b43legacy` drivers. Unfortunately, the main

[1] `http://git.sipsolutions.net/?p=agnx.git`
[2] `http://airgo.wdwconsulting.net/mymoin`

reverse engineer has had to leave the project for personal reasons. Worse, the driver is still not fully functional. Until the `agnx` reverse engineering team is reconstituted, this driver remains in limbo.

The `at76_usb` driver supports USB wireless chipsets from Atmel. This driver has been ported to the `mac80211` infrastructure and it works reasonably well. It is possible that it will be merged as early as the 2.6.27 merge window.

A driver has appeared recently for the Marvell 88w8335 chipset. These are PCI (and CardBus) devices manufactured a few years ago, and Marvell has shown little interest in supporting an upstream driver for them. While these devices were marketed under the "Libertas" brand, these devices bear little or no resemblance to those covered by the `libertas` driver. The driver for these devices is called `mrv8kng` and it has been posted for review. Hopefully that will lead to it being mergeable as early as the 2.6.27 merge window as well.

Marvell has shown interest in supporting a driver for their current TopDog chipset. They have released a driver based on the `net80211` infrastructure from the Madwifi project, and they have provided some minimal documentation on the firmware for their device. Unfortunately, no one is known to be actively working to port this driver to `mac80211` or to make it mergeable into the mainstream kernel.

2.1.3 Unlikely Drivers

Hardware vendors come and go, and some products are more successful than others. Nowadays most Linux wireless drivers are reverse engineered, and reverse engineering takes lots of both motivation and skill. While it is certainly possible that a motivated and skilled reverse engineer will apply his craft to produce a driver for an uncommon hardware device, the odds are against such an occurrence. So wireless devices that enjoyed limited market penetration and are no longer in production are unlikely to ever get a native Linux driver. One example of such a device is the InProComm IPN2220. Unfortunately, NDISwrapper will likely remain the only viable solution for making this hardware work under Linux.

2.2 Patch Activity

Take it from the author, someone who knows: wireless LAN is currently one of the fastest developing segments of the Linux kernel. New drivers or new features arrive nearly every month, and the large portions continue to undergo extensive refactoring as lessons are learned and functionality is extended. In fact, some might say that too much refactoring continues to occur! Nevertheless, the wireless developers continue to be prolific coders.

This is not simply anecdotal—Linux Weekly News (LWN) regularly documents patch activity in the kernel as it nears each release. Because of the role the author plays as wireless maintainer, the number of Linux kernel Signed-off-by's for the author provides a good proxy for the level of activity around wireless LAN in the kernel. For 2.6.24, LWN placed the author as the #5 "gatekeeper" for patches going into the linux kernel.[3] This represented over 4% of the total patches in that release, or more than 1 out of every 25. For 2.6.25, it was a full 5%, or 1 out of every 20 patches.[4] This is all the more impressive when one considers that no wireless vendor other than Intel provides direct developer services.[5] Given the amount of work remaining, I see no reason to believe that this level of production will change significantly in the near future.

2.3 How does someone get the code?

Given the quick pace which continues around the Linux kernel in the area of wireless LAN development, not all distributions are shipping with current wireless code. All of this progress does one no good if you do not have the code. So how is a user to go about getting it?

2.3.1 Development Trees

Ongoing development is done using `git`, now the standard revision control system for the Linux kernel. Multiple trees are used in order to accomodate various needs relating both to distributing patches to other kernel maintainers and to facilitating further development.

[3] `http://lwn.net/Articles/264440/`

[4] `http://lwn.net/Articles/275954/`

[5] In fact, nearly all of the prominent wireless developers are university students!

Most of these trees are of no interest either to end users or to casual developers.

The tree that is of interest is the wireless-testing tree, `git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-testing.git`. This tree is generally based on a recent "rc" release (e.g., 2.6.25-rc9) from Linus. It also includes any patches destined for the current release that have not yet been merged by Linus, as well as any patches queued for the next release. This tree might also include drivers that are still in development and are not yet considered stable enough for inclusion even in the next kernel release. Consequently, this tree is intended as the base for any significant wireless LAN development. Users or developers seeking an up-to-date wireless LAN codebase should use this tree.

2.3.2 The compat-wireless-2.6 project

Not everyone is interested in running a "bleeding edge" kernel. The compat-wireless-2.6 project[6] was created to fill this need. This project provides a means to compile very recent wireless code in a way that is compatible with older kernels. At the time of this writing, kernels as old as 2.6.21 are supported. Even older kernels may be supported by the time you read this.

The compat-wireless-2.6 project maintains a set of scripts, patches, and compatibility code. These are combined with code taken from a current wireless-testing tree. The resulting code is compiled against the user's running kernel, resulting in modules that can be loaded to provide current wireless bugfixes and updated hardware support. Users of enterprise distributions or others who need to run older kernels may find this project to be quite useful.

2.3.3 Fedora

Obviously the easiest way to get kernels is through a distribution. This is especially true for users who may not be kernel hackers or even software developers at all. Perhaps unfortunately (or possibly by design), distributions have varying policies regarding kernel updates. Consequently, not all distributions have an easy means for users to run kernels with current wireless LAN code. One distribution that does provide such kernels is Fedora.

The Fedora build system is called Koji,[7] and all official Fedora packages are built there. As a Fedora contributor, the author ensures that current wireless fixes and updates make their way into the Fedora kernel on a timely basis. The normal Fedora update process typically makes kernel updates available within a few weeks. Those too impatient to wait can retrieve later kernels directly from Koji. Picking the latest kernel built by the author is usually the best way to find the kernel with the latest wireless bits:

`http://koji.fedoraproject.org/koji/userinfo?userID=388`

2.4 Website

Those simply wanting a starting point for information about current wireless LAN developments would do well to vist the Linux Wireless wiki,[8] graciously hosted by Johannes Berg. This site has information organized for users, hardware vendors, and potential developers. Since it is a wiki, the site is easily updated as old information becomes obsolete, and it is open to a wide variety of potential contributors who may or may not be actual software developers. The Linux Wireless wiki is a good first stop for anyone seeking more information about wireless LAN support in Linux.

3 What is coming?

The road behind us was a hard slough, and now we stand on steady ground. Yet we are far from home! What new features are coming to Linux wireless LANs? A new and better means for configuring wireless devices is on the way, and new ways for using those devices to communicate are coming as well.

3.1 Replacing Wireless Extensions with CFG80211

Traditionally Linux wireless interfaces have been configured using an Application Programming Interface (API) known as Wireless Extensions (WE). This API

[6] `http://www.linuxwireless.org/en/users/Download`

[7] `http://koji.fedoraproject.org`

[8] `http://wireless.kernel.org`

is based on a series of `ioctl` calls, each of which specifies the parameters of a specific attribute for wireless LAN configuration. This API mapped sufficiently well to the designs of wireless LAN devices that were prevalent when WE was produced, and it continues to remain at least minimally serviceable for modern designs. However, WE has many shortcomings. Chief of these is that it fails to specify a number of details about what default behaviors should be, what the proper timing should be, or order of configuration steps, or even what the exact meaning is intended to be for a number of parameters. Further, reliance on individual configuration actions for what otherwise might be considered atomic operations introduces the possibility of race conditions when configuring devices. Also, WE has proven to be difficult to extend without breaking the kernel's pledge of userland Application Binary Interface (ABI) consistency between releases. Finally, the in-kernel WE implementation is mostly transparent, forcing individual drivers to reimplement a number of features of the API which might otherwise be shared.[9] All of this, coupled with the general disdain for `ioctl`-based interfaces which is now prevalent amongst kernel developers, makes it difficult to extend or even adequately maintain WE going forward.

In order to address this, work has begun on `cfg80211`. This is a component intended to replace WE for configuration of wireless interfaces. The `cfg80211` component should provide a much cleaner API both to userland applications and to drivers. The userland interface (as implemented in the Netlink-based `nl80211` component) will provide a logical grouping of configuration parameters so that logically atomic operations are actually handled atomically within the kernel. On the driver side, `cfg80211` will provide an interface that minimizes the amount of configuration handling that drivers need to do on their own.[10] Finally, the designers of `cfg80211` have attempted to observe the API design lessons learned over a decade of continuing Linux development. The `cfg80211` component should remain both extensible and maintainable for a long time to come.

[9] This leads to differing behaviors between drivers and is a potential source of bugs that might otherwise be avoided.

[10] This should serve to provide much more consistent wireless driver behavior observable by userland applications like NetworkManager.

3.2 "Access point" mode

Most people interested in wireless LAN technology know that there is a difference between a wireless client device and a wireless access point. The latter is usually a small box with antennas on it that plugs into the wired LAN (e.g., the Ethernet jack on the wall or the back of a cable modem or DSL modem). Access points are wireless infrastructure devices that coordinate wireless LAN traffic, and they require somewhat different software to implement this coordination behavior. Further, many early wireless device designs made it impossible to implement an access point no matter if you had the required software or not. This is why the list of devices traditionally supported by the `hostapd` software is rather short.

The dirty little secret is that there is not really anything special about the physical wireless LAN hardware used in an access point. Usually only the software and/or firmware controlling it prevents or enables it to be used as an access point. With older designs, this meant that access points needed devices with firmware which allowed access point functions to work. With the "soft MAC" designs which are now prevalent, this means that software in the kernel can allow a wide variety of devices to be used to implement an access point.

It turns out that the `mac80211` component already contains most of the code needed to enable this access point behavior. However, it is currently disabled in stock kernels. This is because that behavior needs a stable API for control by userland software (e.g., `hostapd`), and such an API has not yet been agreed upon. Work is in progress (and far along) on implementing support for such an API in the `cfg80211` and `nl80211` components. The current maintainer of the `mac80211` component, Johannes Berg, has a series of patches for both the kernel and for `hostapd` that enables using Linux as an access point. This support may be merged into the mainline kernel as early as version 2.6.27.

3.3 Mesh networking

Many people realize that there are currently two common modes of communication on wireless LANs: a) *access point* or infrastructure mode, where the wireless client talks to an access point that directs traffic to the rest of the LAN; and b) *ad-hoc* or independent BSS mode, where wireless clients coordinate wireless traffic

amongst each other, but with traffic limited to stations that are physically in range of one another. In recent years a new mode has been under development. This mode, commonly called *mesh networking*, is a bit like a mixture of the two previous modes. Wireless stations coordinate wireless traffic amongst each other within a limited range, but also stations can pass traffic between stations that cannot otherwise reach each other. This enables communication over much larger ranges without requiring lots of infrastructure, and is therefore ideal for underdeveloped or disaster-stricken areas. This mode of communication has seen popular use by the OLPC project in their XO laptops.

The people that developed the wireless mesh firmware for the adapters on the XO laptop have also contributed a pre-standard implementation of mesh networking for the `mac80211` component. The developers from Cozybit seem to be committed to maintaining and improving this code until the 802.11s specification for mesh networking is finalized. This gives Linux a cutting-edge wireless capability not currently seen in other mainstream operating systems. Surely this will prove useful to a great number of people all over the world.

3.4 Multi-Queue Support

The 802.11e wireless standard defines Quality of Service (QoS) mechanisms based on classifying traffic into four queues.[11] All pending traffic in the highest priority queue is transmitted before traffic in the next highest priority, and so on. The `mac80211` component implements support for this by using a custom queueing discipline associated with the physical wireless device.

Modern wired LAN devices have evolved designs which also have multiple queues for supporting QoS applications. Consequently, the Linux kernel's networking infrastructure has been extended to support the concept of multiple hardware queues attached to a single network interface. Now that this exists, it seems sensible to convert the `mac80211` component to make use of this new infrastructure. Work is currently underway to achieve just that.

4 What else is needed?

So, it seems that things are firming up reasonably well. Further, the wireless LAN developers already have the next round of work cut out for them. But surely that is not all that is lacking? There certainly are areas that need to be addressed. These include better power management and taking advantage of performance-enhancing features available to drivers within the Linux kernel's networking layer.

4.1 Better Power Management

Power management is an important issue. Not only are mobile devices continuing to proliferate, but also energy efficiency has become increasingly important even with desktop computers and other fixed-location devices. There are both economic and ecological reasons behind this trend, and it is unlikely to significantly decrease in importance anytime soon—just the opposite is likely. So it behooves wireless LAN devices to be good citizens regarding power usage.

4.1.1 Suspend and Resume

Drivers receive notifications of suspend and resume events from the core kernel. Drivers are expected to save or restore the state of their associated hardware as appropriate for the specific notification. This approach suffices for the vast majority of LAN adapters. Since "full MAC" devices implement the connection management functionality themselves, this approach should work for those devices as well.

In the case of `mac80211`-based devices, the actual hardware driver does not implement the connection management functionality. Since the wireless LAN environment may change radically between when a device is suspended and when it is resumed, there is no way for a `mac80211`-based driver to reliably resume operational state by itself after a suspend.

Unfortunately, the `mac80211` component is currently completely unaware of suspend and resume events. Drivers work around this by unregistering themselves from `mac80211` upon suspend and re-registering themselves upon resume. This method works reasonably well in many circumstances, but it is unreliable and it tends to increase the wait required after a resume before the wireless interface is again usable. The `mac80211` component needs to be aware of suspend and resume events and it needs to handle them appropriately without forcing driver work-arounds.

[11] The queues are designated for voice traffic, video traffic, best-effort traffic, and background traffic.

4.1.2 Power Saving Mode

The 802.11 specification includes a mechanism for a device in infrastructure mode to notify its associated access point that it is entering power-saving mode. The access point then queues frames intended for the power-saving station. Periodically the station returns to full power state long enough to ask the access point if it is holding traffic for the station, and to accept delivery of any such traffic. This can enable a device to save a great deal of power if it is not actively transmitting traffic.

The `mac80211` component currently makes no use of this mechanism. The potential for power savings makes this seem like a "must have" feature. On the other hand, implementing it may not be as simple as it sounds. Still, this would be a welcome addition to the `mac80211` feature set.

4.2 NAPI Interrupt Mitigation

NAPI is a mechanism used by network drivers to mitigate the costs of processing interrupts on a busy network link. The basic idea is to disable interrupts after the first one and schedule a polling routing to keep processing incoming frames. In that way, the kernel only incurs the cost of the first interrupt in a burst of traffic rather than processing interrupts for individual frames.

Originally, NAPI implicitly assumed that a given interrupt source was associated with a single network interface. Because the `mac80211` component supports multiple kernel network interfaces on a single physical wireless interface, `mac80211` drivers were implicitly excluded from using NAPI. This has not been a huge problem, because the transfer speeds used on wireless networks has been relatively slow. However, 802.11n is bringing much faster speeds to wireless LANs. Fortunately, NAPI has been changed to disassociate interrupt sources from specific network interfaces. The `mac80211` component should take advantage of NAPI in order to maximize wireless LAN performance.

5 Other issues

Perhaps more than any other section of the kernel, wireless LAN support is held hostage to non-technical concerns. Coping with these legal and political issues is key to maintaining and improving good support for wireless LANs in the Linux kernel.

5.1 Firmware Licensing

Wireless LAN protocols have stringent timing requirements for a number of operations. Consequently, many adapters have an embedded microcontroller with firmware to handle a variety of operations. This is true even for many "soft MAC" designs, even though they rely on the host processor for most higher-level operations. These devices simply do not work without their required firmware.

In the Windows and OSX worlds, drivers typically include the firmware as data embedded in the driver binaries. In most cases developers have learned how to separate the firmware from those binaries so that they can be loaded by Linux drivers as well. Unfortunately, the copyright status of the resulting firmware images is at best uncertain.

Many vendors have provided liberal licenses for their firmware images which allow those images to be freely distributed for use with Linux drivers. Intel and Ralink are two examples of good citizens in this regard. Other vendors have proven unwilling to cooperate on this issue, with Broadcom as the most clear example of an uncooperative vendor. Making wireless LAN hardware purchasing decisions in support of cooperative vendors is advised as the best approach to resolving this issue.

5.2 Vendor Participation

Many vendors have proven unwilling to provide either support or programming documentation for their wireless LAN devices. This is true even for vendors like Broadcom who have grown accustomed to providing such support for their wired LAN devices. Some vendors cite the spectre of governmental regulation as a reason they cannot participate in the creation of open source drivers. Other vendors (such as Realtek) find ways to provide community developers with information and still others (particularly Intel) provide developers dedicated to the cause. Unfortunately, too many vendors continue to depend on support from reverse engineers and unsupported community developers. Again, economic pressure is advised as the best approach to resolving this situation.

5.3 Regulatory Issues

As noted above, vendors often cite the spectre of governmental regualtors as a reason to provide poor support

for Linux or no support for Linux at all. Unfortunately, these fears are not altogether unfounded. The regulations governing wireless LAN communications are determined by geographical location and political realities. There are literally dozens or even hundreds of different regulatory jurisdictions covering the planet. Each of these jurisdictions can have its own set of rules about which channels are available, what practices (e.g., active scanning) are acceptable on each channel, what power output is acceptable on each channel, whether the rules differ between indoor and outdoor operation, and many other variables.

Complying with all these regulations can be troublesome at best. Add to that the fact that failure to ensure compliance with local regulations can result in local officials refusing to let a vendor conduct business in their jurisdiction. One can understand how a vendor may be hesitant to embrace the loss of total control over their products. Still, some vendors have found ways to overcome these fears. Once again, economic pressure is advised to persuade hesitant vendors to find ways to satisfy regulators while supporting open source drivers, as well as to reward those vendors which have already done just that.

6 Conclusion

Hopefully it is clear to the reader that a great deal of progress has been made for Linux in the wireless LAN arena over the past few years. Not so long ago Linux was a wireless LAN ghetto, with only a few devices working reliably enough for general use. Now most consumer devices are either already supported, or support already exists for a similar device, with reverse engineering efforts continuing in hope of supporting new device versions. Better still, we now see a number of wireless LAN vendors offering some form of Linux support even if that is only providing liberal licensing terms for device firmware. Also, development of fixes and new features continues to make Linux wireless LAN even better. Wireless LAN is no longer the biggest problem for Linux.

Proceedings of the Linux Symposium

Volume Two

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*