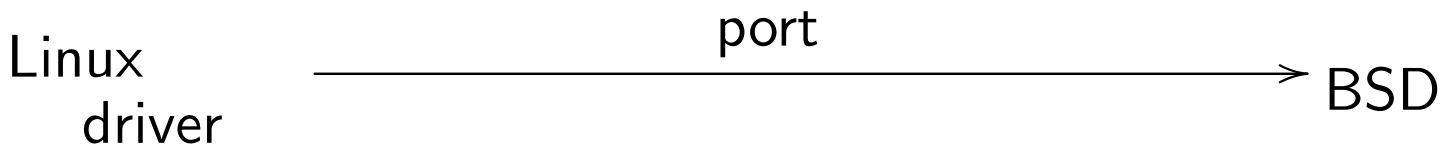
Increasing Automation in the Backporting of Linux Drivers Using Coccinelle

Luis R. Rodriguez, (SUSE Labs)
Julia Lawall (Inria/LIP6/UPMC/Sorbonne University-Whisper)

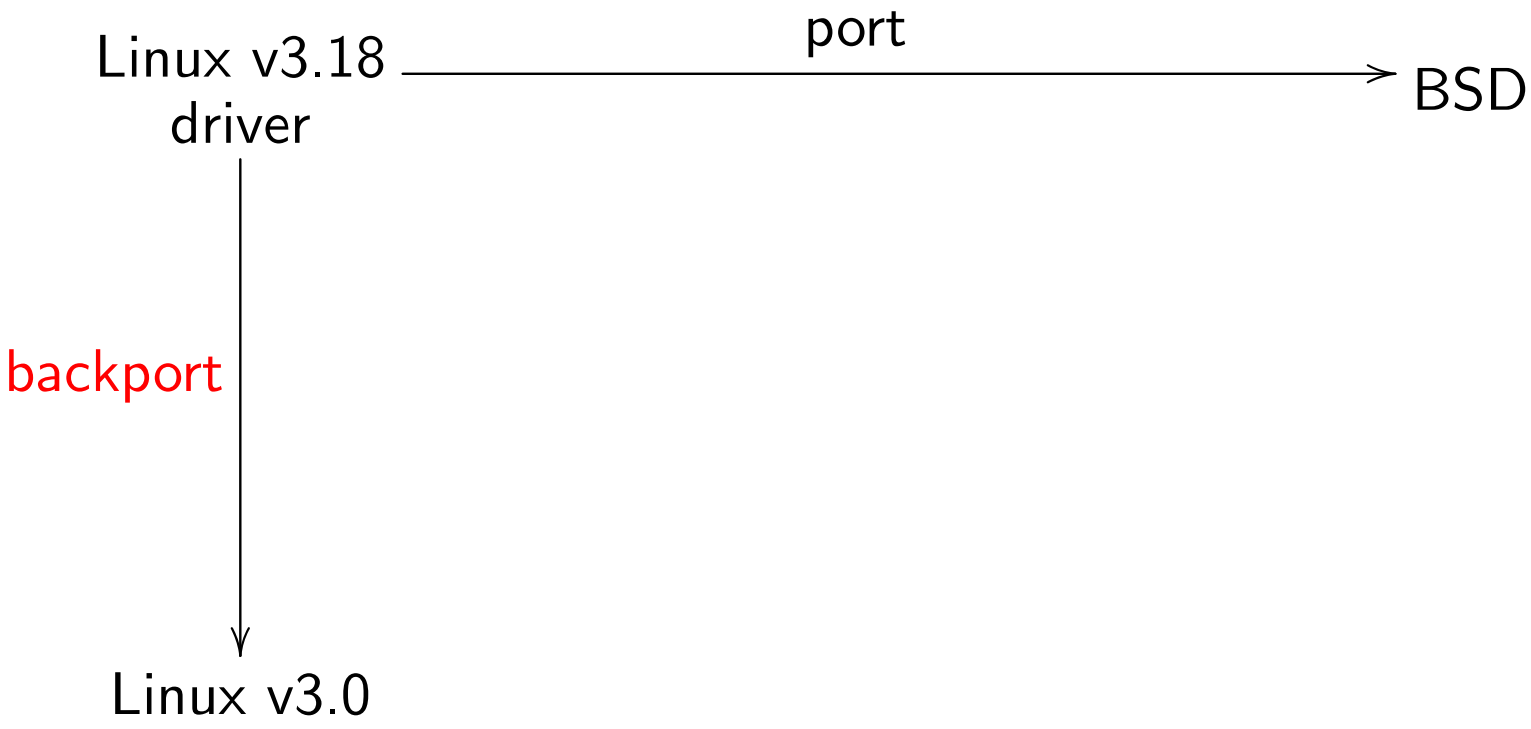
January, 2015

(Unpublished work)

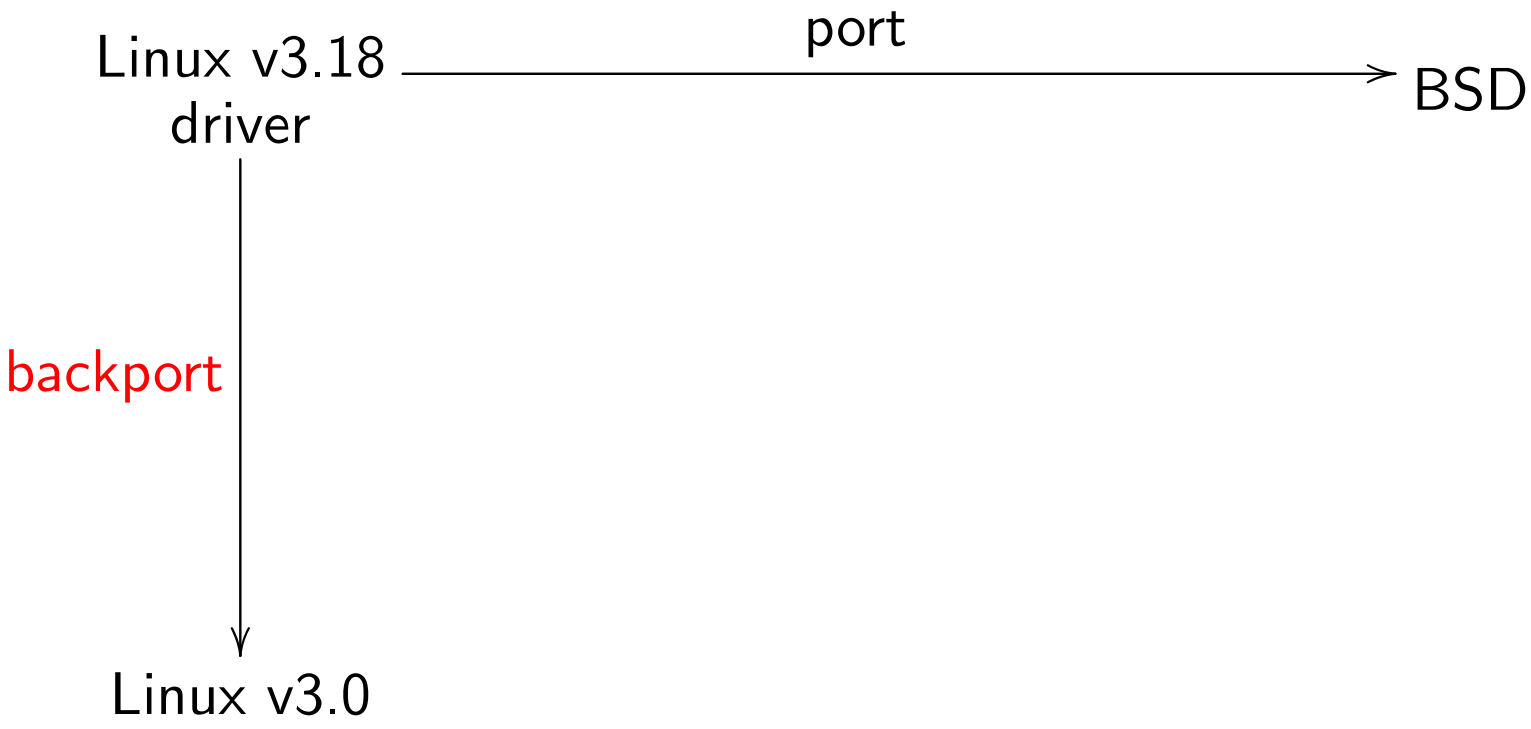
What is backporting?



What is backporting?



What is backporting?

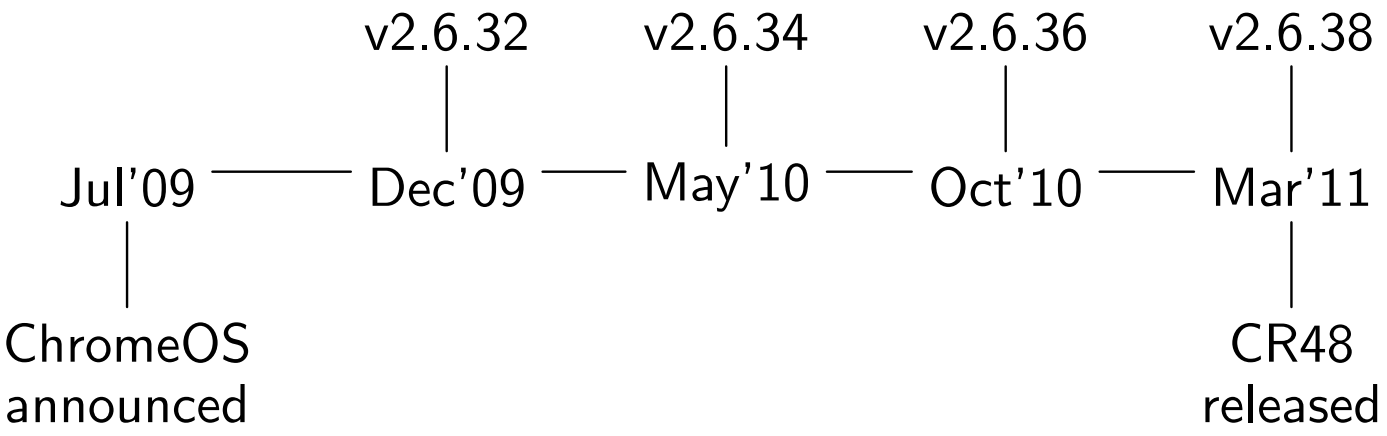


Why would we want to do that?

The latency of product development

Athalon wireless device

The latency of product development



- ChromeOS based on Linux v2.6.32.
- New devices appear all the time.
  - Eg, Atheros IEEE 802.11n wireless chipset.
- Ath9k driver developed for Linux v2.6.38, not Linux v2.6.32

Possible solutions

Make an ath9k driver for Linux v2.6.32?

- Lots of work, error-prone.
- Atheros may not be motivated.
- ChromeOS may modernize to eg Linux v2.6.36.

Possible solutions

Make an ath9k driver for Linux v2.6.32?

- Lots of work, error-prone.
- Atheros may not be motivated.
- ChromeOS may modernize to eg Linux v2.6.36.

Modernize ChromeOS to Linux v2.6.38?

- Not in the short term.
- May prefer using a stable kernel.

Possible solutions

Make an ath9k driver for Linux v2.6.32?

- Lots of work, error-prone.
- Atheros may not be motivated.
- ChromeOS may modernize to eg Linux v2.6.36.

Modernize ChromeOS to Linux v2.6.38?

- Not in the short term.
- May prefer using a stable kernel.

Ensure Linux v2.6.38 drivers run out of the box on Linux v2.6.32?

- Hinders advancement.
- Not in the Linux philosophy.

Backporting

Goal:

- Slightly modify modern drivers for compatability with older versions.

Backporting

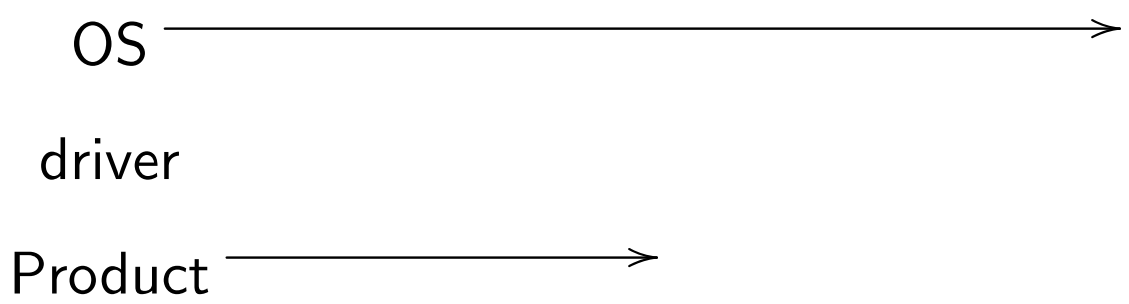
Goal:

- Slightly modify modern drivers for compatability with older versions.

Issues:

- Where to start?
- How to express modifications?
- Scalability.
  - 10,000 or so Linux drivers.
  - Code arrives/modified every day.

Where to start?



- Device manufacturer targets an OS version relevant to potential customers.
- OS always moves ahead
  - New Linux release every 2.5-3 months.
- Products may modernize as well.
- A driver targeting any specific release is always left behind.

Upstream-first development

Our problem:

- A driver is too modern for existing clients,
- And too old fashioned for future clients.

Upstream-first development

Our problem:

- A driver is too modern for existing clients,
- And too old fashioned for future clients.

Upstream-first development:

- Driver integrated with HEAD of Linus's git tree.

Upstream-first development

Our problem:

- A driver is too modern for existing clients,
- And too old fashioned for future clients.

Upstream-first development:

- Driver integrated with HEAD of Linus's git tree.
- Advantages
  - Driver developed once, modernized by kernel maintainers.
  - Solves our second problem.
- Inconveniences
  - Coding style constraints.
  - What about backporting?

Upstream-first development

Our problem:

- A driver is too modern for existing clients,
- And too old fashioned for future clients.

Upstream-first development:

- Driver integrated with HEAD of Linus's git tree.
- Advantages
  - Driver developed once, modernized by kernel maintainers.
  - Solves our second problem.
- Inconveniences
  - Coding style constraints.
  - What about backporting?

To make upstream-first development attractive, we need an "industrial-strength" solution to backporting.

How to express modifications?

Typical strategy: `#ifdefs` by kernel versions.

An artificial example:

```
#if (LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,29))
A_new();
#elif (LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,25))
A_older();
#else
A_very_old();
#endif
```

A real example

Linux v2.6.28 code: drivers/net/usb/usbnet.c

```
net->change_mtu = usbnet_change_mtu;
net->get_stats = usbnet_get_stats;
net->hard_start_xmit = usbnet_start_xmit;
net->open = usbnet_open;
net->stop = usbnet_stop;
net->watchdog_timeo = TX_TIMEOUT_JIFFIES;
net->tx_timeout = usbnet_tx_timeout;
```

Current code: (12.12.2014)

```
net->netdev_ops = &usbnet_netdev_ops;
net->watchdog_timeo = TX_TIMEOUT_JIFFIES;
```

Issues

Given `net->netdev_ops = &usbnet_netdev_ops;`, must:

- Find the definition of `usbnet_netdev_ops`:

```
static const struct net_device_ops usbnet_netdev_ops = {
  .ndo_open             = usbnet_open,
  .ndo_stop             = usbnet_stop,
  .ndo_start_xmit       = usbnet_start_xmit,
  .ndo_tx_timeout       = usbnet_tx_timeout,
  .ndo_set_rx_mode      = usbnet_set_rx_mode,
  .ndo_change_mtu       = usbnet_change_mtu,
  .ndo_set_mac_address  = eth_mac_addr,
  .ndo_validate_addr    = eth_validate_addr,
};
```

- Find the names of the corresponding fields.
  - Some perhaps didn't exist.
- Remove the definition of `usbnet_netdev_ops`.
- Construct the new code.

Result, part 1

```
--- a/drivers/net/usb/usbnet.c
+++ b/drivers/net/usb/usbnet.c
@@ -1151,6 +1151,7 @@
 }
 EXPORT_SYMBOL_GPL(usbnet_disconnect);

+#if (LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,29))
 static const struct net_device_ops usbnet_netdev_ops = {
    .ndo_open               = usbnet_open,
    .ndo_stop               = usbnet_stop,
@@ -1160,6 +1161,7 @@
    .ndo_set_mac_address    = eth_mac_addr,
    .ndo_validate_addr      = eth_validate_addr,
 };
+#endif
 /*-------------------------------------------------------------------*/
```

Result, part 2

```
@@ -1229,7 +1231,15 @@
      net->features |= NETIF_F_HIGHDMA;
 #endif

+#if (LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,29))
    net->netdev_ops = &usbnet_netdev_ops;
+#else
+   net->change_mtu = usbnet_change_mtu;
+   net->hard_start_xmit = usbnet_start_xmit;
+   net->open = usbnet_open;
+   net->stop = usbnet_stop;
+   net->tx_timeout = usbnet_tx_timeout;
+#endif
    net->watchdog_timeo = TX_TIMEOUT_JIFFIES;
    net->ethtool_ops = &usbnet_ethtool_ops;
```

Assessment

- Added: 5 lines of C code, 5 lines of `#ifdefs`.
- Changes maintained as patches
  - Allows changes upstream.
- 61 `netdev_ops` fields possible.
- All field names change.
- `ndo_set_mac_address` and `ndo_validate_addr` removed.
  - Have default values.
- Bug? What happened to `ndo_set_rx_mode`?

484 `netdev_ops` initializations in 434 files.

Backports via a compatability library

Observations:

- The code to modify is copious but repetitive.
  - Remove a structure, because its type is not available.
  - Copy structure field values.

Backports via a compatability library

Observations:

- The code to modify is copious but repetitive.
  - Remove a structure, because its type is not available.
  - Copy structure field values.

These changes can be encapsulated in a library:

- Define the missing type.
- Define a function to perform the structure copy.

Backports via a compatability library

Observations:

- The code to modify is copious but repetitive.
  - Remove a structure, because its type is not available.
  - Copy structure field values.

These changes can be encapsulated in a library:

- Define the missing type.
- Define a function to perform the structure copy.

Proposition of the Linux backports project

- Initiated in 2007 by Luis R. Rodriguez, to backport 802.11 wireless drivers.

Compat library-based approach

```
--- a/drivers/net/usb/usbnet.c
+++ b/drivers/net/usb/usbnet.c
@@ -1446,7 +1446,7 @@ usbnet_probe (struct usb_interface *udev
      net->features |= NETIF_F_HIGHDMA;
 #endif

-   net->netdev_ops = &usbnet_netdev_ops;
+   netdev_attach_ops(net, &usbnet_netdev_ops);
    net->watchdog_timeo = TX_TIMEOUT_JIFFIES;
    net->ethtool_ops = &usbnet_ethtool_ops;

--- a/drivers/net/wireless/ath/ath6kl/main.c
+++ b/drivers/net/wireless/ath/ath6kl/main.c
@@ -1289,7 +1289,7 @@ static const struct net_device_ops ath6k

 void init_netdev(struct net_device *dev)
 {
-   dev->netdev_ops = &ath6kl_netdev_ops;
+   netdev_attach_ops(dev, &ath6kl_netdev_ops);
    dev->destructor = free_netdev;
    dev->watchdog_timeo = ATH6KL_TX_TIMEOUT;
```

Backports two drivers, in one line each.

Scalability

Current status of the backports project:

- 800 ethernet, wireless, bluetooth, NFC, ieee802154, media, and regulator drivers.
- Backported from their linux-next, release candidate, and recent stable versions.
- 18 earlier releases as backport targets.
- linux-next and linux-stable evolve every day.
- Changes maintained as patches, which become out of date.
- 2-6 iterations of tests, refinements, compiles for all supported versions.
  - Patches are fragile.

Goal: Automate the transformation part.

Coccinelle to the rescue

Our transformations have a lot in common:

```
-   net->netdev_ops = &usbnet_netdev_ops;
+   netdev_attach_ops(net, &usbnet_netdev_ops);

-   dev->netdev_ops = &ath6kl_netdev_ops;
+   netdev_attach_ops(dev, &ath6kl_netdev_ops);
```

Similar, but one per file.

Coccinelle:

- Semantic patches, generalizing over unimportant details.
- Used for over 2000 Linux kernel patches.

Backporting netdev_ops with Coccinelle

```
- net->netdev_ops = &usbnet_netdev_ops;
+ netdev_attach_ops(net, &usbnet_netdev_ops);
```

Backporting netdev_ops with Coccinelle

```
- dev->netdev_ops = &ops;
+ netdev_attach_ops(dev, &ops);
```

Backporting netdev_ops with Coccinelle

```
@@
struct net_device *dev;
struct net_device_ops ops;
@@
- dev->netdev_ops = &ops;
+ netdev_attach_ops(dev, &ops);
```

Backporting netdev_ops with Coccinelle

```
@@
struct net_device *dev;
struct net_device_ops ops;
@@
- dev->netdev_ops = &ops;
+ netdev_attach_ops(dev, &ops);
```

6 lines to backport this change for all drivers.

Performance

Patch

- Applies to a specific file and line number.
- No parsing required.

Performance

Patch

- Applies to a specific file and line number.
- No parsing required.

Coccinelle

- Parses semantic patch and C code,
- Searches for positions where the semantic patch matches,
- Performs the tranformation.

Performance

Patch

- Applies to a specific file and line number.
- No parsing required.

Coccinelle

- Parses semantic patch and C code,
- Searches for positions where the semantic patch matches,
- Performs the tranformation.

Coccinelle optimizations

- Parallelism, by file.
- Keyword indexing.
- Can be faster than sequential patch application.

A more complex example

Threaded IRQs introduced in Linux v2.6.31.

- Adds an extra handler to normal `request_irq` call.
- Need somewhere to store this handler.

A more complex example

Threaded IRQs introduced in Linux v2.6.31.

- Adds an extra handler to normal `request_irq` call.
- Need somewhere to store this handler.

Solution

- Use device's private structure.
- Need to find structure type name, extend structure definition.

Extending the private structure using Coccinelle

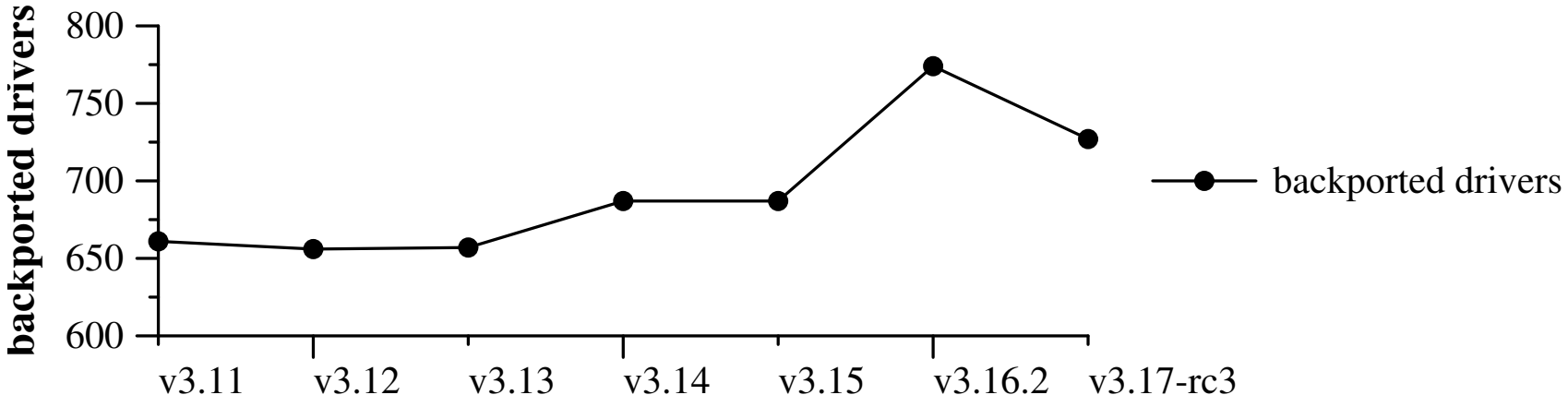
```
@ threaded_irq @
identifier ret; type T; T *private;
expression irq, irq_handler, irq_thread_handler, flags, name;
@@
+#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,31)
ret = request_threaded_irq(irq, irq_handler,
          irq_thread_handler, flags, name, private);
+#else
+ret = compat_request_threaded_irq(&private->irq_compat,
+         irq, irq_handler, irq_thread_handler,
+         flags, name, private);
+#endif

@ modify_private_header depends on threaded_irq @
type threaded_irq.T;
@@
T {
+#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,31)
+       struct compat_threaded_irq irq_compat;
+#endif
...
};
```

Update some IRQ oprations accordingly.

Conclusion

Status



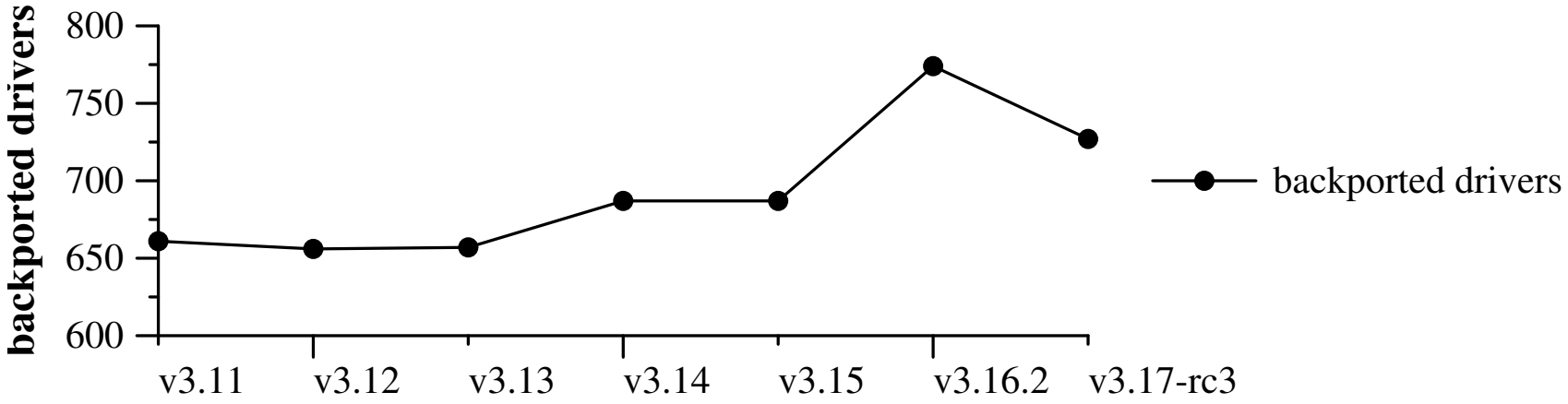
Currently 5 semantic patches, representing 471 lines of code.

Future work:

- Make Linux code more backport friendly.
- Infer semantic patches, or even compat library code.
- Address correctness issues - currently, only compilation.

Conclusion

Status



Currently 5 semantic patches, representing 471 lines of code.

Future work:

- Make Linux code more backport friendly.
- Infer semantic patches, or even compat library code.
- Address correctness issues - currently, only compilation.

"All the patches that broke often in the early days are now using coccinelle or are removed because they were only needed for the older kernel versions." [Hauke Mehrtens, 10.23.2014]