Increasing Automation in the Backporting of Linux Drivers Using Coccinelle

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

September 10, 2015

What is backporting?

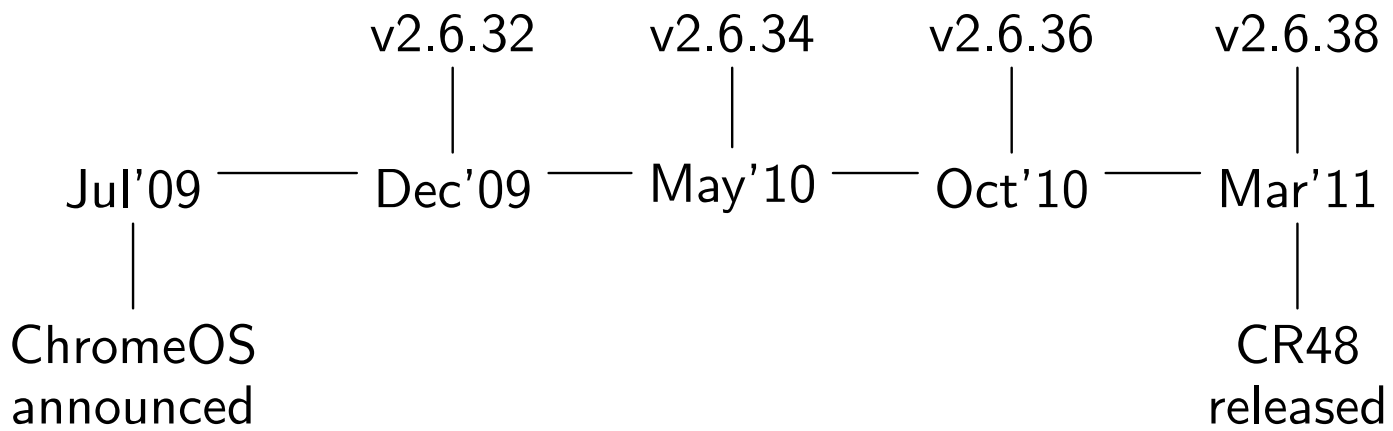
What is backporting?

What is backporting?

Why would we want to do that?

The latency of product development

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 compatibility with older kernel versions.

Backporting

Goal:

- Slightly modify modern drivers for compatibility with older kernel versions.

Issues:

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

What version to start with?

Our problem:

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

What version to start with?

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 Torvalds' git tree.

What version to start with?

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 Torvalds' git tree.
- Advantages
  - Driver developed once, modernized by kernel maintainers.
  - Solves our second problem.
- Inconveniences
  - Coding style constraints.
  - What about backporting?

What version to start with?

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 Torvalds' 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 backport modifications?

Typical strategy: `#ifdefs` by kernel versions.

An artificial example:

```
A_new();
```

How to express backport modifications?

Typical strategy: `#ifdefs` by kernel versions.

An artificial example:

```
#if (LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,29))
A_new();
#endif
```

How to express backport 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();
#endif
```

How to express backport 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: (01.09.2015)

```
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 old 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;
```

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;
```

5 lines of #ifdefs, 5 lines of C code, per driver

Backports via a compatibility library

The Linux backports project

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

Backports via a compatibility library

The Linux backports project

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

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 compatibility library

The Linux backports project

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

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.

Compat library-based approach

```
--- a/drivers/net/usb/usbnet.c
+++ b/drivers/net/usb/usbnet.c
@@ -1446,7 +1446,7 @@ usbnet_probe (...
      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;
```

Backports each driver, with a one-line change.

Scalability

Current status of the backports project:

- 766 ethernet, wireless, bluetooth, NFC, ieee802154, media, and regulator drivers.
- Backported from their linux-next, release candidate, and recent stable versions.
- 23 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 and improve the dependability of the transformation part.

Coccinelle to the rescue

Backport modifications 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);
```

Still, a change needed for every file

- Needs to be redone when nearby context changes.

Coccinelle:

- Semantic patches
- Generalizes 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);
```

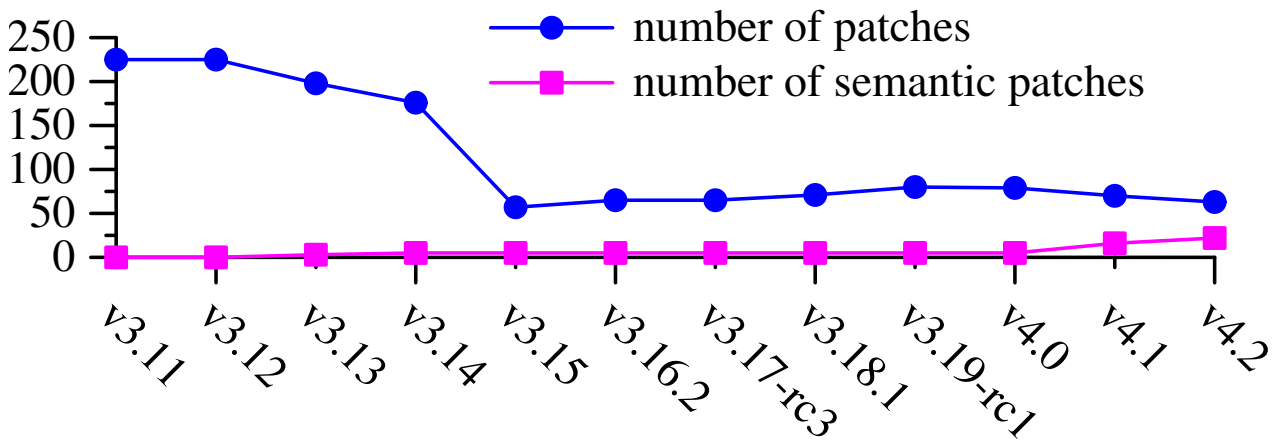
Backporting netdev_ops with Coccinelle

```
@@
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.

Status

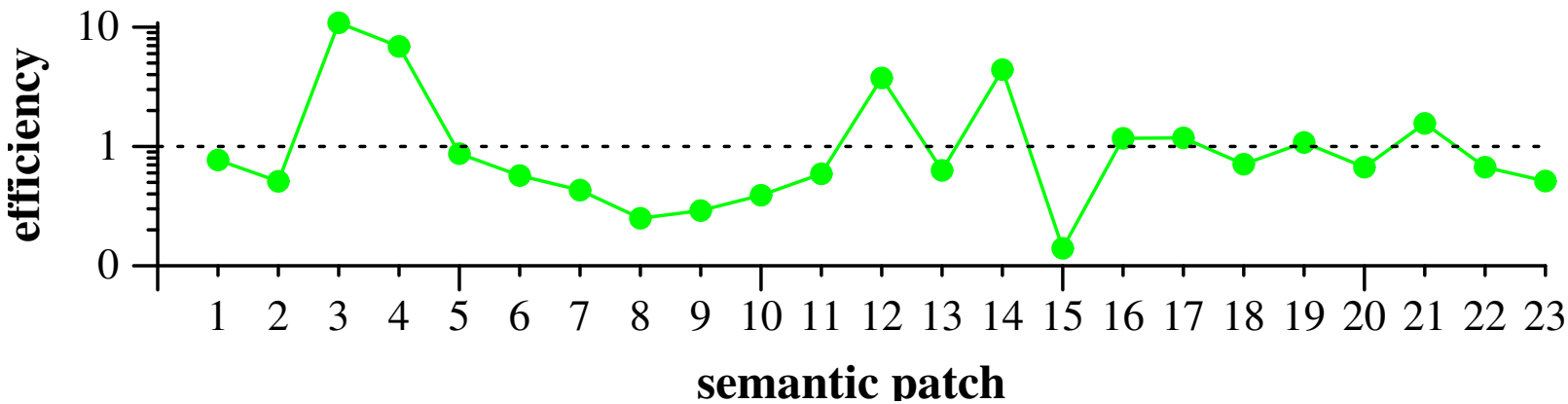
Patches and semantic patches used by backports:



- 3.11: 661 backported drivers
- 4.2: 766 supported drivers
- Patch diff contribution: 43.5414%
- Coccinelle diff contribution: 56.4586%

Development efficiency

$$\text{Development efficiency} = \frac{\text{Added/removed lines}}{\text{Semantic patch lines}}$$



Semantic patches apply "for free" to later versions.

- Over the last almost 2 years, 28 changed lines for the 23 semantic patches (622 LOC).
- 3.82 patches per patch for the 70 remaining standard patches.

Other benefits of using Coccinelle

- Consistent backporting
- Orthogonal backporting steps
- Further improve the dependability of the backporting process

Correctness and performance issues

Coccinelle design is pragmatic, no formal correctness guarantees

- Results are compile tested.
- Users provide feedback.
- Backporting changes typically involve global information.
  - Little ambiguity and changes rarely

Coccinelle must parse and analyze the code

- Less efficient than patch application, but more naturally parallelizable.
- Keyword indexing for fast file selection.
- Can be faster than sequential patch application.

Conclusion

6201 lines of standard patch code converted to:

- 23 Coccinelle semantic patches (622 lines of code)
  - Replaces 3501 lines of standard patch code, 56% of total.
- 2700 lines of standard patch code for one-off changes.

Total of 766 drivers supported by the backports project.

Future work:

- Make Linux code more backport friendly.
- Infer semantic patches, or even backports library code.
- Introduce new strategies for checking correctness.

Conclusion

6201 lines of standard patch code converted to:

- 23 Coccinelle semantic patches (622 lines of code)
  - Replaces 3501 lines of standard patch code, 56% of total.
- 2700 lines of standard patch code for one-off changes.

Total of 766 drivers supported by the backports project.

Future work:

- Make Linux code more backport friendly.
- Infer semantic patches, or even backports library code.
- Introduce new strategies for checking correctness.

"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]