Compressing Strings of the Kernel

Wolfram Sang

Consultant

21.8.2014, LinuxCon14

The origin: CEWG project[1]

From the proposal:

Attempts have been made in the past to compress printk messages to save kernel runtime footprint. There is an option to disable all printks, but many embedded developers do not use it, even when they find the space savings attractive, because they still would like to see kernel debug messages.

...

Timothy Miller did some work on this in 2003

...

[1] `http://elinux.org/Compressed_printk_messages`

Timothy's approach[2]

1. Compile kernel and keep `.i`-files
2. Filter them for printk strings
3. Compress those strings using tokenization
4. Create copies of the source files
5. There, replace strings with tokens
6. Compile again

[2] `http://lwn.net/Articles/28935/`

Further notes

- no code was made public, only the description, a codebook and some results
- not even sure depacking at printk was ever made
- `allyesconfig` was used for the tests
- based on 2.4.20 and 2.5.68

Author asks the golden question, too

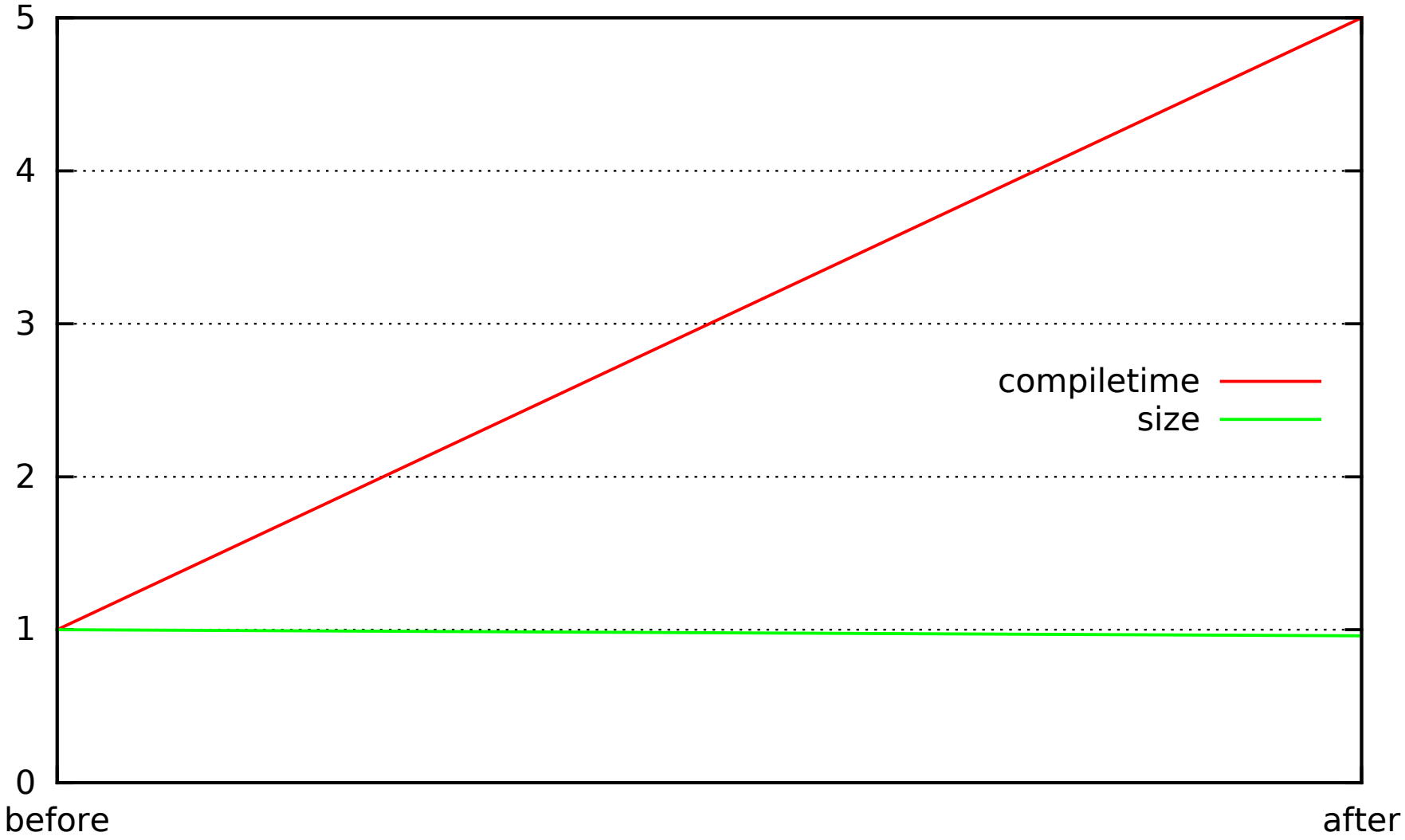
Timothy Miller:[3]

"So, I ask... is this a useful savings? Is there any chance anyone would bother to increase their compile time by a factor of 5 in order to shave off 4% or 100k bytes?"

[3] `https://lkml.org/lkml/2003/6/6/207`

The Graph!

The Graph!



Three problems identified

1. Extract printk format strings
2. Compress printk format strings
3. Replace printk format strings

Extracting

Problem: Find all printk-strings

- There are *lots* of functions/defines embedding printk/vprintk_emit
- They are nested in all ways you can think of
- Moving target, there will be more

```
<new_subsys>_dev_err, …
```

Extracting: Options I

Scan the source files

- needs to know *all* printk-emerging functions
- misses merging of literals
- handle all ways of string concatenation

Extracting: Options II

printk strings to own section

- scales a bit better (only base functions need to be converted)
- no knowledge where strings came from
- needs changes to core functions

Extracting: own section

```
+#define __printk(fmt, args...) \
+do { \
+    if (__builtin_constant_p(fmt)) { \
+        static const __attribute__((section("__printk"))
+            char __f[] = fmt; \
+        printk(__f, ##args); \
+    } else \
+        printk(fmt, ##args); \
+} while (0)
```

Extracting: own section II

```
 #define pr_emerg(fmt, ...) \
-    printk(KERN_EMERG pr_fmt(fmt), ##__VA_ARGS__)
+    __printk(KERN_EMERG pr_fmt(fmt), ##__VA_ARGS__)
 #define pr_alert(fmt, ...) \
-    printk(KERN_ALERT pr_fmt(fmt), ##__VA_ARGS__)
+    __printk(KERN_ALERT pr_fmt(fmt), ##__VA_ARGS__)
     ...
```

BTW don't redefine `printk`. Really, don't!

Extracting: own section III

```
Author: Wolfram Sang <wsa@the-dreams.de>

    WIP: move printk strings to a special section

    Only pr_*, dev_*, BUG, and WARN are supported.
    Kernel doesn't fully build yet.

    Signed-off-by: Wolfram Sang <wsa@the-dreams.de>

 drivers/base/core.c                 | 36 ++++------------------------
 include/asm-generic/bug.h           | 19 ++++++++++++----
 include/asm-generic/vmlinux.lds.h |  5 ++++
 include/linux/device.h              | 48 ++++++++++++++++++++++------------------
 include/linux/printk.h              | 27 +++++++++++++++-------
 5 files changed, 75 insertions(+), 60 deletions(-)
```

Upstreaming forecast

Upstreaming forecast

Compressing

Problem: Algorithm

- lots of small strings
- should be instantly available

  not somewhere in the middle of packed data
- no significant overhead

Compressing

Problem: Algorithm

- lots of small strings
- should be instantly available

  not somewhere in the middle of packed data
- no significant overhead

Conclusion

- no sliding window algos (LZ and friends)
- no variable length encoding (Huffman and friends)
- no frequency based compression (stats3)

Compressing II

- tokenization is actually a good option
- BytePairEncoding works, too
- both achieve $\approx 50\%$ of compression
- still $\approx 4\%$ of the kernel size gained

Compressing II

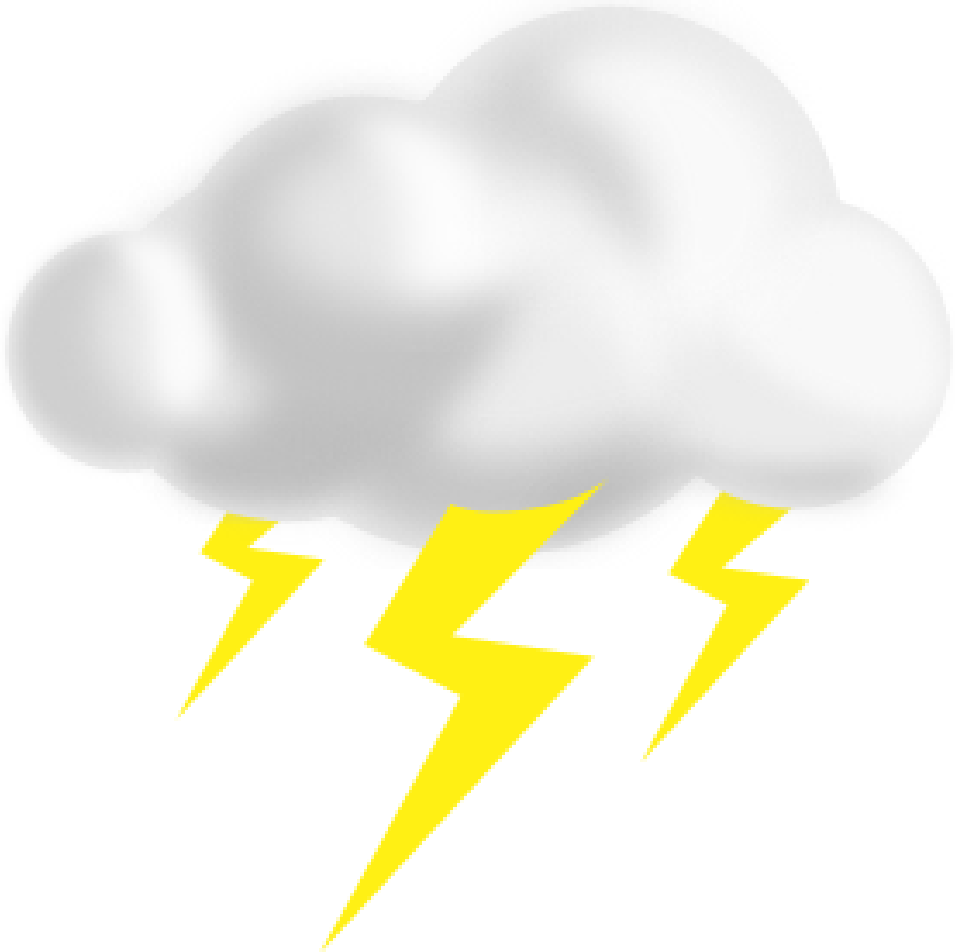
- tokenization is actually a good option
- BytePairEncoding works, too
- both achieve $\approx 50\%$ of compression
- still $\approx 4\%$ of the kernel size gained

Problem: UTF8

Both approaches need 'empty' symbols which might collide with UTF8 encoding.

Upstreaming forecast

Upstreaming forecast



Compressing III

Problem: Codebook

- `allyesconfig` is unrealistic for tiny systems
- smaller kernels, smaller pool for codes
- what about modules?
  - share codebook from kernel → tied to that build
  - own codebook → overhead eats gain

Brainstorming

- predefined codebook?
- could save second kernel compile, too

Replacing

Scanning

run source files through filter before compiling

Own section

Work on the section directly?

Upstreaming forecast

Upstreaming forecast

Further issues

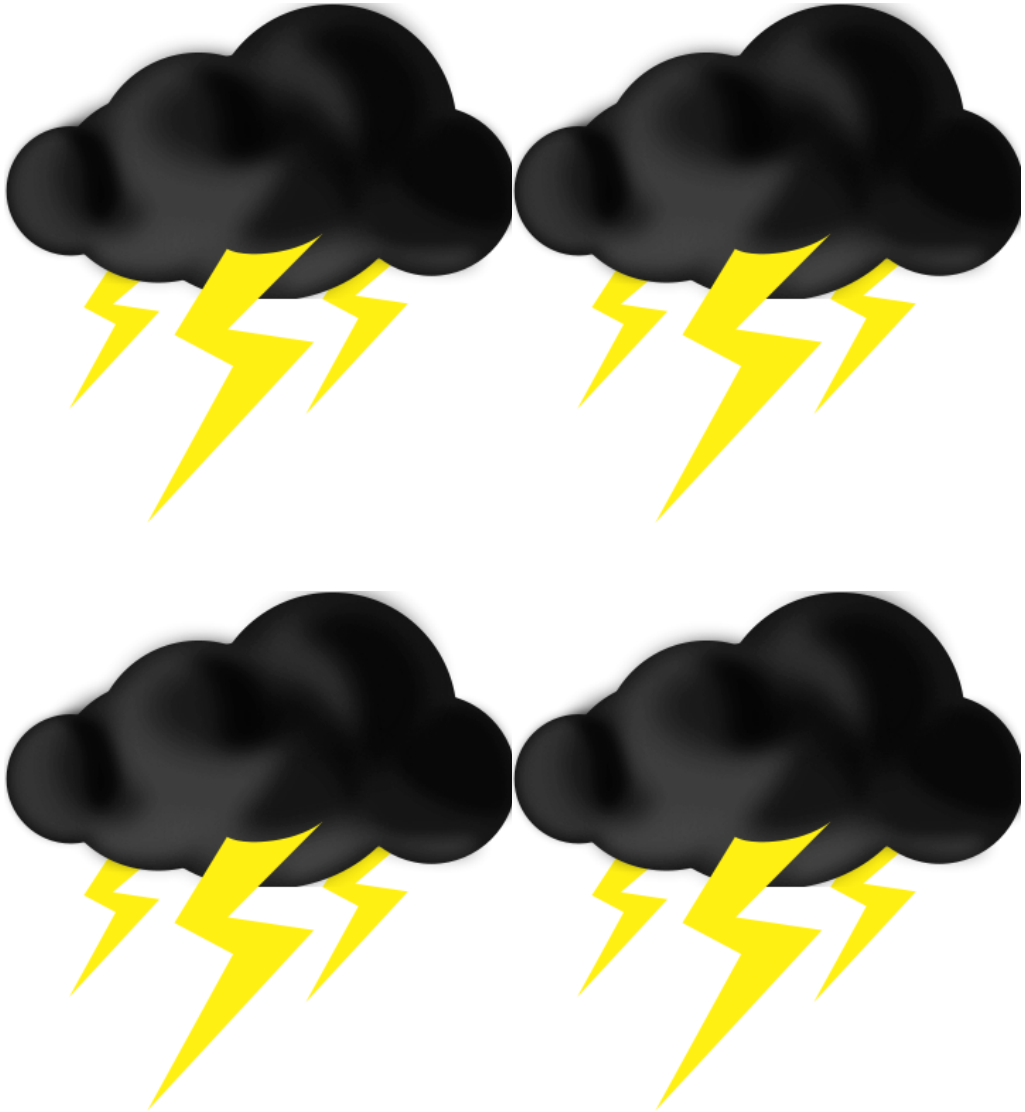
- printk strings are only a subset
- devicetree uses a lot of strings!

  which should be easier to tackle since they are accessed via of_* functions
- address this problem at a higher level

  all strings? all .rodata?

Summary

Summary



This quote still makes sense

From: Managing Gigabytes[4]

"We find ourselves in the midst of a practically important and intelectually fascinating convergence between the desire for more and better compression and the need to learn about what 'structure' there is in data."

[4] Witten/Moffat/Bell, 1st edition, p. 385

Analyzing the data

- observations from 3.16-rc5:
  - x86-64 allyesconfig
  - arm-cortexa8 customer kernel
- maybe a bit biased for device drivers

Print from central locations

Proposal

- strings should be emitted from as centralized locations as possible
- bonus: consistent messages

Examples

- OOM error message removal
- `devm_ioremap_resource()`

Print from central locations II

Possibilities

- have basic functions not printing messages
- have a convenience function suitable for most cases printing error messages
- `devm_get_*_optional` are also good candidates
- Removing error strings for `devm_clk_get` saved 20K instantly
- lots of other candidates

Prefixes

Observation

- usually done by a literal prefixing the format string
- creates unique strings
- ...which have redundancy in them
- use `dev_*` and friends if possible

```
$ strings ubi.o
...
3UBI error: %s: negative values
3UBI error: %s: bad alignment
3UBI error: %s: alignment is not multiple of min I/O
...
```

Prefixes: Dead simple tinyfication

How `pr_fmt` gets applied:

```
#define pr_alert(fmt, ...) \
    printk(KERN_ALERT pr_fmt(fmt), ##__VA_ARGS__)
```

Prefixes: Dead simple tinyfication

How `pr_fmt` gets applied:

```
#define pr_alert(fmt, ...) \
    printk(KERN_ALERT pr_fmt(fmt), ##__VA_ARGS__)
```

How to simplify it:

```
-#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt
+#define pr_fmt(fmt) "%s" fmt, KBUILD_MODNAME ": "
```

That saved $\approx$ 15% (or 250 byte) for sn9c20x. Applied to all 900 instances in the kernel, it saved about 30K (or 4‰). So, works only for some places.

Prefixes: More easy stuff

```
 /* UBI error messages */
-#define ubi_err(fmt, ...) pr_err("UBI error: %s: " fmt "\n",      \
-                 __func__, ##__VA_ARGS__)
+#define ubi_err(fmt, ...) printk("%s%s: " fmt "\n",      \
+                KERN_ERR "UBI error: ",  __func__, ##__VA_ARGS__)
```

That saved $\approx$ 15% (or 2.5K). UBIFS, JFFS2, SCSI layer seem also to be promising candidates.

Copy'n'paste

```
switch (sd->sensor) {
case SENSOR_OV9650:
    ov9650_init_sensor(gspca_dev);
    if (gspca_dev->usb_err < 0)
        break;
    pr_info("OV9650 sensor detected\n");
    break;
case SENSOR_OV9655:
    ov9655_init_sensor(gspca_dev);
    if (gspca_dev->usb_err < 0)
        break;
    pr_info("OV9655 sensor detected\n");
    break;
case SENSOR_SOI968:
    soi968_init_sensor(gspca_dev);
    if (gspca_dev->usb_err < 0)
        break;
    pr_info("SOI968 sensor detected\n");
    break;

/* ... 7 more ... */
```

Copy'n'paste II

And in the init functions:

```
if (gspca_dev->usb_err < 0)
    pr_err("OV9650 sensor initialization failed\n");
...
if (gspca_dev->usb_err < 0)
    pr_err("OV9655 sensor initialization failed\n");
...
if (gspca_dev->usb_err < 0)
    pr_err("SOI968 sensor initialization failed\n");
...
```

Copy'n'paste: A little love <3

- proper cleanups are most sustainable
- will not only remove strings but code as well
- somebody up to it? I donate the camera :)

Layer 8

Be aware when adding strings

- be precise with printk level

  adds possibility to compile out based on level

- be conservative with non debug strings

  we need rules of thumb here...

- try to be consistent with the strings you create
- strings cost (a little), so they should be worth it

Conclusion I

Compressed printk-strings, only for corner-cases

- the price for compressed printk-strings is still high
- lots of side effects and increase of build time
- looking at printk-strings is not enough

Conclusion II

General improvements, benefit for all

- there is quite some potential to simply reduce # of strings, especially throught centralization
- ...which mostly make them more consistent, too
- should be the first goal before stepping further

The End

Thank you for your attention!

Questions? Comments?

- right now
- anytime at this conference
- wsa@the-dreams.de