Faults in Patched Kernel

Alexandre Lissy
Mandriva
alissy@mandriva.com
Laboratoire d'Informatique de l'Université de Tours
alexandre.lissy@etu.univ-tours.fr

Stéphane Laurière
Mandriva
slauriere@mandriva.com

Patrick Martineau
Laboratoire d'Informatique de l'Université de Tours
patrick.martineau@univ-tours.fr

Abstract

Tools have been designed to detect for faults in the Linux Kernel, such as Coccinelle, Sparse, or Undertaker, and studies of their results over the vanilla tree have been published. We are interested in a specific point: since Linux distributions patch the kernel (as other software) and since those patches might target less common use cases, it may result in a lower quality assurance level and fewer bugs found. So, we ask ourselves: is there any difference between upstream and distributions' kernel from a faults point of view ? We present an existing tool, Undertaker, and detail a methodology for reliably counting bugs in patched and non-patched kernel source code, applied to vanilla and distributions' kernels (Debian, Mandriva, openSUSE). We show that the difference is negligible but in favor of patched kernels.

1 Introduction

A survey of the number of "faults" in the Linux kernel has been conducted in 2001 by [1]. In this paper, authors showed the relatively large number of bugs in drivers of the kernel: the `drivers` subdirectory of the kernel accounts for 7 times the number of bugs found in source files compared to the other directories. And this is not a matter of code length, as the figure is given by the following ratio:

$$r = \frac{err\_rate_{drivers}}{err\_rate_{non-drivers}}$$

Ten years later, an updated version of the survey on more recent kernels was contributed in [4]. One of the major outcomes of this second paper is that it shows drivers are less prone to errors: the first paper pointed out methods of improvement in the kernel, and a large effort has been conducted in this area to make things better indeed. Now, the `arch` subdirectory is the one where faults are most concentrated! No reason other than "people and institutions focus on drivers quality" is given to explain the improvement. This result also emphasizes the need to periodically check and follow the status of bugs: basically, actions need a continuous assessment of their usefulness. So the community (be it either researchers, companies, etc.) needs feedback to know where the effort is needed. Some effort in this area is already proposed by Phoronix.com[1]: the website regularly checks the status of the current versions of the kernel using its Phoronix Test Suite, checking mainly for regressions (performance, power consumption, etc.). It should also be noted that the authors of [4] have a tool, Heorodotos [3], which they use for tracking detected bugs life cycle: this is already a first step towards the tracking of "where we need to work". Our motivations are further detailed in section 2, then the tool used in section 3. The methodology is presented and explained in section 4 and we present then discuss results in section 5.

2 Motivations

As we just exposed, there have been several studies of the fault density in the kernel, with several tools. Chou et al. [1] used `xgcc`, itself introduced by [2], which is able to find 12 types of errors in the source code of Linux (and OpenBSD). The output has been inspected

[1] http://www.phoronix.com

by hand to check for false positives, etc., and the kind of errors checked for are: deadlock of spinlock, NULL pointer dereference, large stack variable, NULL pointer assumptions, array boundary checking, lock release and no double-lock, use of already-free'd memory, no floats, memory leak, user pointer dereference and correct size allocation.

As stated in [4], the Linux kernel has evolved a lot since this first study, and thus the authors propose to update the results: one first work was to reuse the same checkers, by re-implementing them after explaining how they understood what the original authors meant. To find the faults corresponding to those checks, they used a tool of their own, Coccinelle. They also used Herodotos to track the faults among the versions of the kernel. One common point of those two studies is that they only follow the upstream Linux kernel, they have no interest in *forks* or derivatives. This is the case of kernel used by distributions: they backport features, they fix bugs, etc., and in our opinion, it is interesting to study whether, from a faults point of view, there is a difference between distribution's kernels and the upstream one.

Another motivation is to study another type of faults, those measured by Undertaker: a tool that checks for configurations faults. Of course, in their papers [6, 5], the authors show results but once again, only on upstream kernel. Following the same idea as previous authors, we would like to have a clear point of view of the status of our distribution's kernel and where it needs focus for improvements: there is no obvious reasons that it will be the same than upstream.

3 Undertaker: Finding Configurations Defects

Software configurability in the Linux kernel is a giant and growing space: currently, more than 10000 *features* are available. A vast majority of software allows for compile-time configuration, which in the case of Linux is handled as preprocessor macros. Many of the options have dependencies between themselves, and tools (Kconfig) allow the user to manipulate the *variability model*. These macros are then used in the C code to enable/disable some features. In [6, 5] the authors present Undertaker, a tool that is able to analyze and check for consistency between configurations models and their use in the source code; not only because it can leave dead code blocks, but also because of incorrect selection of code. Imagine a code path that *must not be followed* in case of `CONFIG_NOT_FOLLOW` is set, but source code uses `#ifdef CONFIG_FOLLOW_NOT`, this will lead to inconsistencies. The authors link a *real-life* case with CPU hotplug: this leads to the feature, hotplug of CPU, which has been broken for more than six months before being fixed. Undertaker makes uses of LIFE (*LInux Feature Explorer*) to extract configurations options as a model from *Kconfig* files, making re-use of *Sparse* (static analysis tool for Linux). Both configuration and implementation models are then transformed to Boolean formulas which can be checked thanks to SAT solvers (Undertaker use *PicoSAT*). Thanks to this, they identified, reported and fixed several issues: 14 were accepted and waiting to be applied to 2.6.34 and a total of 90 have been identified. Please note that we have not been able to work with kernel prior to 2.6.30 due to a bug in Undertaker.

4 Methodology

As stated before, our goal is to be able to check whether quality hypothesis that are considered for upstream kernel versions are also applicable to kernel source code used to create a distribution's package: it can differ in a non-insignificant way. For example, Mandriva packages the Kerrighed kernel (Kerrighed is a SSI system on top of Linux), so basically, it is a giant patch over the upstream kernel. Several distributions also package XEN (which allows efficient virtualisation and paravirtualisation) which is also itself a big patch over the kernel. To measure the faults, we decided to use Undertaker for several reasons:

- It is easy to use, so we can have results rapidly
- It aims at an interesting and newer problem compared with other tools

However, as explained in the section 3, Undertaker only treats the problem of configurability. Although the kernel has a lot of configurable features, configurability is not the only ground for errors, and so we would like to continue and extend this study by integrating other tools to have a better view of the error status for the whole kernel. Using Undertaker is a preliminary step towards wider experiments. Several kernel sources from different distributions (Mandriva, openSUSE and Debian) were used to run the tool over as similar as possible versions. When it analyzes the kernel, Undertaker identifies several kinds of issues:

- Dead code: no configuration item can enable this code
- Undead code: no configuration item can disable this code. Of course, only code between `#if...#endif` is considered

The resulting issues can have different scopes: code is either dead or undead globally, i.e. on all architectures, or it can be dead or undead on several architectures but good on at least one.

In our case, the dead or undead differentiation is not very interesting, as it is a fault in both cases. However, the architecture scope becomes interesting: distributions often target several architectures so we limit ourselves to globally dead or undead code. As in previous papers [1, 4, 6, 5] we consider the number of faults per subdirectories, e.g., `kernel/`, `drivers/`, `mm/`, etc. However, since we are interested in non-upstream kernel, there is one more thing to consider: when analyzing a patched kernel, how can we take into account the differences brought by patches? We do not wish to analyze each kernel patch by hand. The number of lines of ANSI C code present in each subdirectory analyzed by Undertaker is counted, thanks to the *sloccount* tool, and we consider the number of faults in a directory divided by the number of ANSI C lines of code:

$$faultRate_{dir} = \frac{Faults in dir}{ANSI\ C\ LOC in dir}$$

This provides a fault rate per single line of code, which gives an idea of the "quality" status (limiting ourselves to what Undertaker is able to diagnose) in the kernel, independent of the size of the code base. This allow us to compare the impact quality of patches.

When two kernel versions are compared, for each directory, it is simply the difference between the first kernel's values and the second one. So, for a chart comparing 2.6.33.7 and 2.6.33.7-mdv1 (Figure 4), the computed difference is as follows:

$$diff = faultrate_{2.6.33.7} - faultrate_{2.6.33.7-mdv1}$$

Hence a **positive** difference means an **improvement** in Mandriva's kernel, whereas a **negative** difference suggests **new bugs introduced**. Also, it should be noted that the naming of kernel versions follows some distinct conventions for openSUSE and Mandriva: in the first case, it uses the pattern `rpm-kver` where `kver` is the version of the kernel; for Mandriva the pattern is `vkver-mdvX` where `kver` is the version of the kernel and `X` is the release number of the kernel package.

5 Results and discussion

Some preliminary results are now presented and discussed regarding our objectives. In section 5.1 a first run of some "verification against upstream and literature" is done, to check that the methodology gives comparable results to what has already been done, especially in [4, 6, 5]. Then some specific cases are studied: Debian kernel in section 5.2, Mandriva kernel in section 5.3 and openSUSE 11.2 kernel in section 5.4.

5.1 Verification with Upstream

First, the goal is to verify that the approach gives comparable results with [4], not because exact figures are necessary (since measurements are not done over exactly the same things and in the same way, it is meaningless compare directly), but to check that it "reflects" the same tendencies. Our results are present in Figure 1 and target kernel from 2.6.32 to 2.6.38. Similar data is visible in [4, p. 13, Figure 9] where "fault rate per directory" is plotted for kernel 2.6.5 to 2.6.33. It is possible to estimate the fault rate of the `drivers` directory at 0.3, and the fault rate of `arch` at 0.4, compared with [4, p. 13, Figure 9]. In our results, values are respectively of 0.57 and 0.76. This leads to the following ratios:

$$ratio_{coccinelle} = \frac{0.3}{0.4} = 0.75$$

and

$$ratio_{undertaker} = \frac{0.57}{0.76} = 0.75$$

Even if the values are not the same, what they indicate is similar. Also, they lead to convergent interpretations:

- The overall fault rate is decreasing
- `arch` directory shows a higher fault rate than `drivers`

5.2 Debian's kernel

Starting with the Debian kernel, we focus on two 2.6.37 releases provided within the distribution: 2.6.37-1 and

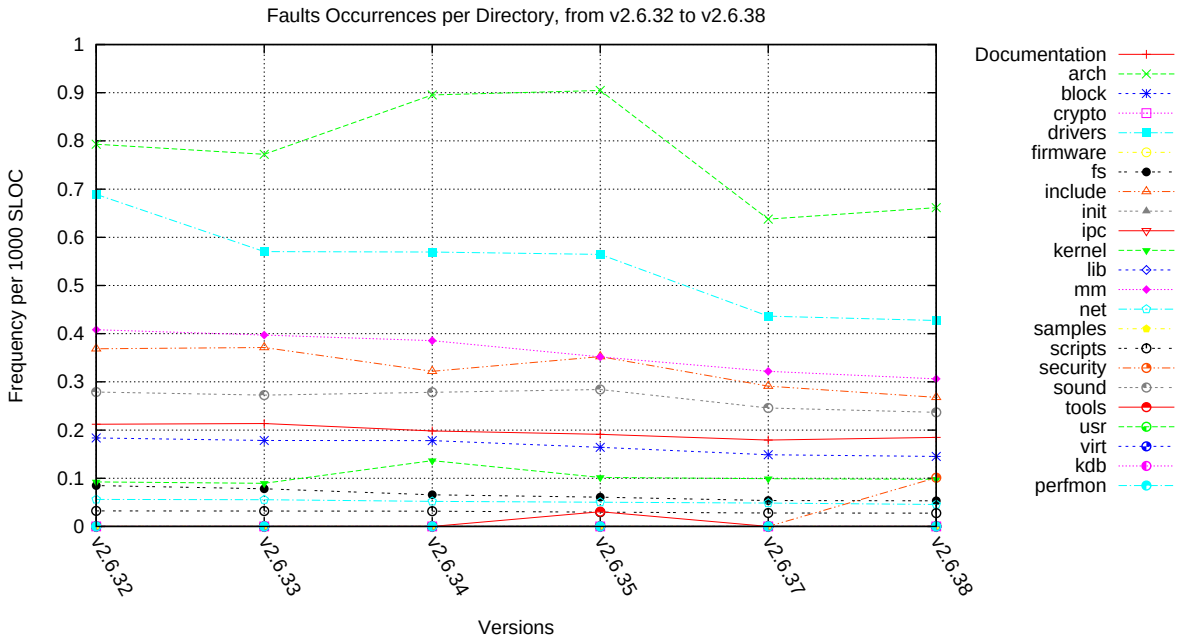

Figure 1: Evolution of the fault rate in the Vanilla kernel, from 2.6.32 to 2.6.38

2.6.37-2. A first look at the chart given in Figure 2 shows that both releases do not present the same fault rate: 2.6.37-2 has a higher fault rate in `kernel` directory than the 2.6.37-1 release. Taking a look at the difference between 2.6.37-1 and 2.6.37 from upstream, we can see in Figure 3 that there is a general **improvement**.

5.3 Mandriva's Kernel

For the Mandriva case, we have made a direct comparison of fault rate for each directory on only one version of the kernel, 2.6.33.7, whose results are available in Figure 4. As stated before, this is computed by doing the difference of fault rate between upstream and the distribution's kernel. From this chart, we can get two interesting points: the number of faults is generally lower in the patched kernel, and the fault rate is generally lower too. But the differences in fault rate are very tight, and even if they are in favor of patched versions, it can be considered as negligible.

	2.6.33.7		2.6.33.7-mdv1	
Directory	Faults	Rate	Faults	Rate
Documentation	2	0.213493	2	0.213061
arch	1240	0.772214	1241	0.772731
block	2	0.178380	2	0.178380
drivers	2558	0.570437	2552	0.567989
fs	50	0.078163	52	0.080378
include	92	0.371393	92	0.370956
kernel	9	0.089347	9	0.089326
mm	18	0.397263	18	0.396922
net	23	0.055458	23	0.055406
scripts	1	0.031829	1	0.031809
sound	115	0.272438	115	0.271721

Table 1: Mandriva Kernel 2.6.33.7 versus Upstream 2.6.33.7

5.4 openSUSE's Kernel

Looking in the openSUSE release gives another insight. Figure 5 shows the status over all RPM released kernels of the openSUSE 11.2 distribution, going from 2.6.30

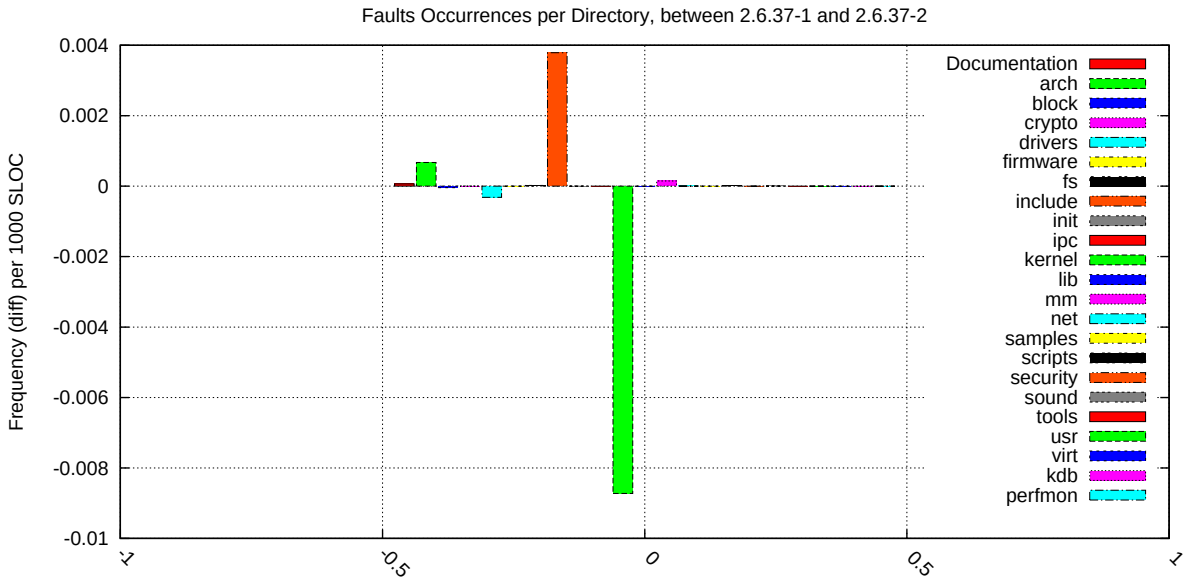

Figure 2: Fault rate differences between Debian 2.6.37-1 and 2.6.37-2

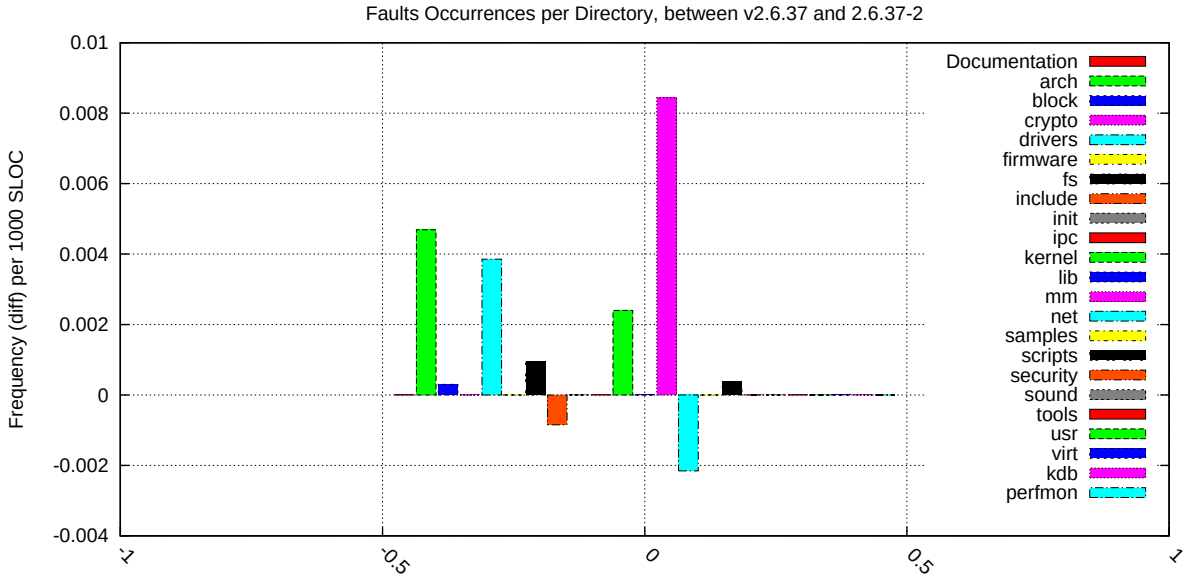

Figure 3: Vanilla 2.6.37 and Debian 2.6.37-2

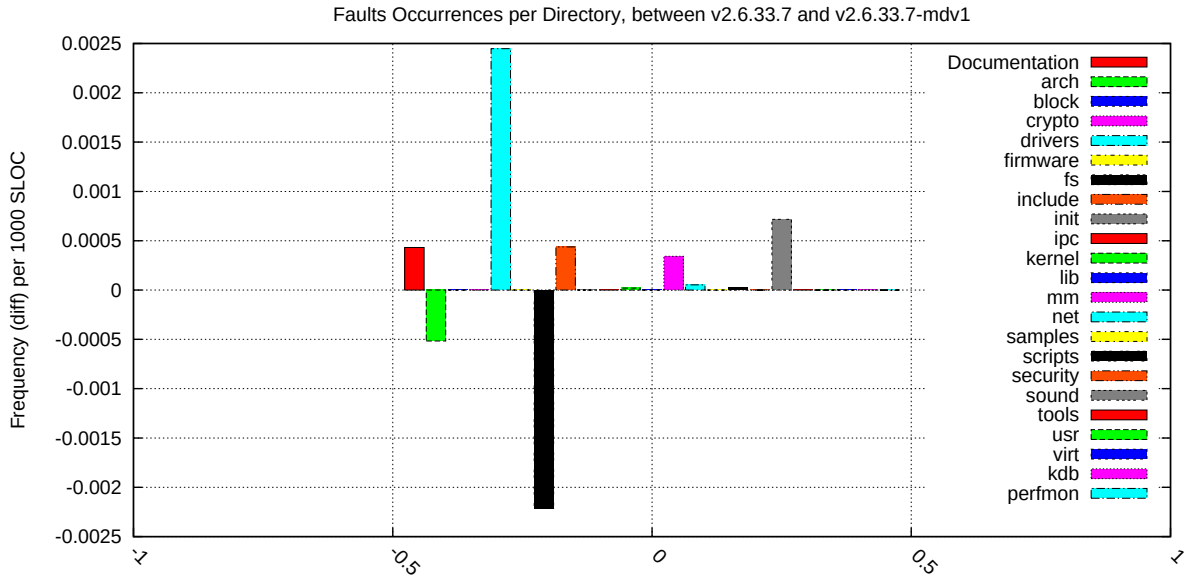

Figure 4: Mandriva versus Vanilla, 2.6.33.7

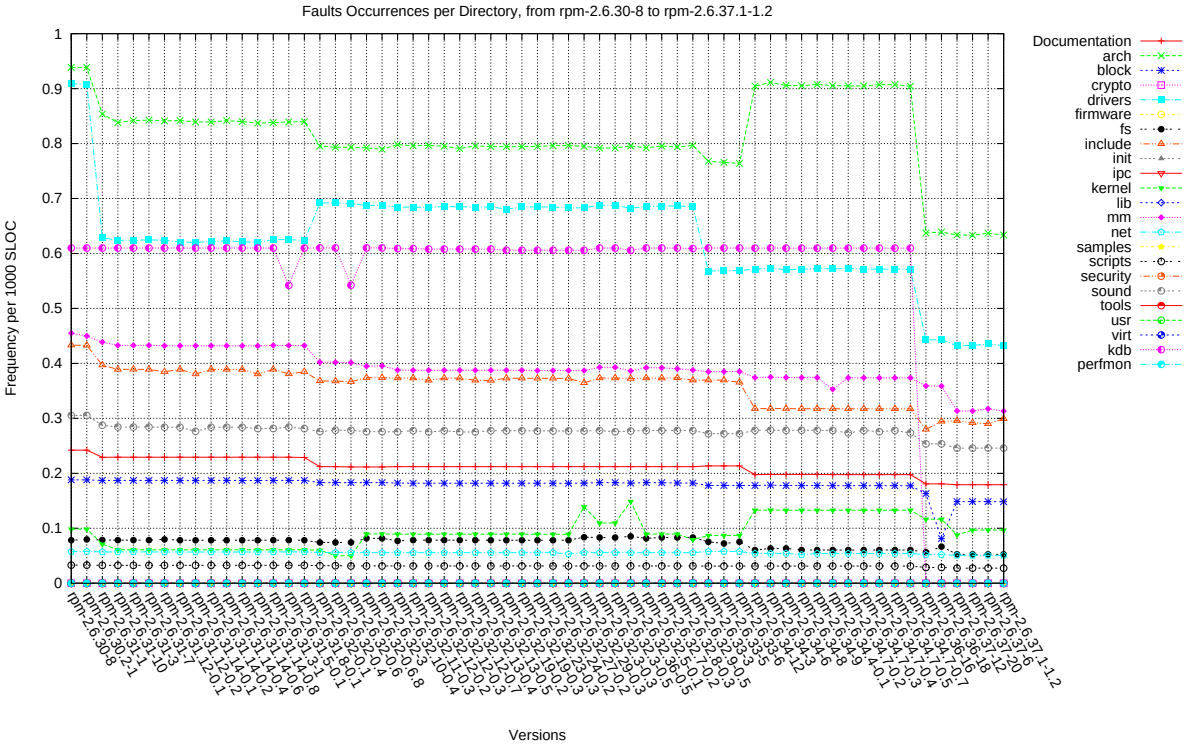

Figure 5: openSUSE Kernel, from 2.6.30 to 2.6.37

	rpm-2.6.37-20		2.6.37	
Directory	Faults	Rate	Faults	Rate
Documentation	2	0.179308	2	0.179308
arch	1145	0.633151	1133	0.637841
block	2	0.148566	2	0.148865
drivers	2261	0.432644	2256	0.436499
fs	36	0.052706	36	0.053648
include	81	0.292085	79	0.291239
kernel	11	0.096745	11	0.099146
mm	16	0.313535	16	0.321970
net	23	0.050789	22	0.048632
scripts	1	0.027578	1	0.027952
sound	118	0.245929	118	0.245929

Table 2: openSUSE 2.6.36-20 versus Vanilla 2.6.37

to 2.6.37 releases: thus, we can compare it roughly to Figure 1 targeting upstream releases between 2.6.32 and 2.6.38. It can be seen that the evolution is quite similar once again. Figure 2 show that the number of added fault is similar to what has been observed in previous cases, and that the fault rate is also lower.

5.5 Overall analysis

A general tendency can be drawn from these results: distributions' kernels, even if patched, shows a slightly lower fault rate. It is interesting, not because it shows that the packaged kernels are somewhat "better" than upstream, but because it shows that they are not worse and that patches do not have a real impact, regarding the configurability faults measured by Undertaker. It should be noted that the difference in fault rate is not important, so there is an impact of patches in those distributions, but it is negligible if considering its importance. As shown in [4], the faults in Linux are decreasing over the time thanks to the efforts being put on producing tools to identify as much as possible of these faults, allowing to fix them.

6 Conclusion

Our initial objective was to study whether, regarding the Undertaker analyzing tool and its target, there were differences between a distribution-patched kernel source code and its upstream counterpart. We presented the way Undertaker works, and we also explained how we used it to measure fault rates. The methodology given also explains the notion of fault rate we used. After checking that we obtain comparable results with previous studies, confirming that Undertaker's specifics can be avoided and that the fault rate it computes gives a similar idea of the quality status of the code base as Coccinelle [4], we ran the process over some kernels from different distributions: Debian, Mandriva and openSUSE. We have been able to show that, whatever distribution we consider, there is a constant pattern: the computed fault rate is often lower on distribution-patched kernels, while the number of faults is often higher. However, the figures also give an important detail: if we look at the deltas, the differences are always very low; for example, in the `arch` directory of openSUSE's 11.2 kernel release 2.6.37-20, we have 1145 faults were identified, while 1133 were in the corresponding upstream version, as illustrated by Table 2. Figures presented in Table 1 are roughly the same for Mandriva's 2.6.33.7 versus Vanilla 2.6.33.7. And regarding the number of faults, the deltas are negligible.

Further work should target a more precise analysis of the faults: it would be interesting to check whether the faults identified and which are more present in distributions' kernels are evenly distributed or whether they concentrate around hot spots. Another improvement would be to integrate other detailed measures such as what is done from Coccinelle in [4]: that would allow us to perform the same checks, i.e. whether there are hot spots in patched-code, not limiting ourselves to the study of configurability.

References

[1] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating system errors. pages 73–88, 2001.

[2] Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem. Checking system rules using system-specific, programmer-written compiler extensions. pages 1–16, 2000.

[3] Nicolas Palix, Julia Lawall, and Gilles Muller. Tracking code patterns over multiple software versions with herodotos. In *AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, pages 169–180, New York, NY, USA, 2010. ACM.

[4] Nicolas Palix, Suman Saha, Gaël Thomas, Christophe Calvès, Julia Lawall, and Gilles Muller.

Faults in Linux: Ten Years Later. Research Report RR-7357, INRIA, 08 2010.

[5] Julio Sincero, Reinhard Tartler, Christoph Egger, Wolfgang Schröder-Preikschat, and Daniel Lohmann. Facing the Linux 8000 Feature Nightmare. In ACM SIGOPS, editor, *Proceedings of ACM European Conference on Computer Systems (EuroSys 2010), Best Posters and Demos Session*, 2010.

[6] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. Feature Consistency in Compile-Time Configurable System Software. In European Chapter of ACM SIGOPS, editor, *Proceedings of the EuroSys 2011 Conference (EuroSys '11)*, 2011.