Real-Time Android: Deterministic Ease of Use

Wolfgang Mauerer,[1,*] Gernot Hillier,[1] Jan Sawallisch,[1] Stefan Hönick,[2] and Simon Oberthür[2]

[1] *Siemens AG, Siemens Corporate Research and Technologies, Otto-Hahn-Ring 6, 81739 Munich, Germany*
[2] *Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany*

The rapid ascent of Android to one of the most influential platforms for mobile devices and tablets shows that the platform meets the preferences of end-users and developers with consistent usability and a convenient development environment targeted at the needs of the many instead of a specialised few. Being based on the Linux kernel, it inherits the rich and mature feature set which made Linux the number one embedded operating system in just a few years.

The Android stack, however, only uses and provides a small subset of those features. Real-time capabilities, which enabled Linux for a much broader embedded audience, were not considered in the Android design. By introducing a real-time capable Android appliance, we add a crucial Linux building block combining the benefits of both realms.

Besides presenting the software architecture, we discuss our efforts in augmenting the Android stack with RT capabilites in a minimally invasive way, provide effort measurements, and present a performance evaluation based on a prototype implemented using a Motorola Xoom tablet featuring our architecture extensions.

I. INTRODUCTION

In stark contrast to the user-centric and -friendly world of Android that also spreads into the application developer realm, solid real-time (RT) capabilites are the almost exclusive concern of Linux in the industrial embedded domain. Frugality dominates the many solutions that bear no resemblance to the colourful Android universe. This is a significant drawback when devices require user interaction. While the learning curve for an Android phone is deemed minimal even for technically unenthusiastic users, comparatively simple tasks (like Stroustrup's infamous use of his own telephone) can become daunting with ill-designed human-machine interfaces [14]. Analogously, programmers versed in desktop application development face considerable obstacles when turning their attention to embedded Linux work.

The combined real-time Android system is supposed to provide remedies for both, users and programmers of embedded real-time systems. Besides, our work also has applications in machine consolidation: Steadily increasing computational power available in embedded systems, fuelled by a trend towards multi- and many-core systems, make it likely that the traditionally independent tasks of RT control and human-machine interface (HMI) provision will be served by a single hardware instance. This necessitates appropriate basis software architectures, which we present in this paper.

Additionally, we report on practical experiments with an prototypal implementation of our architecture on a Motorola Xoom tablet with Android 3.1. Equipped with an Nvidia Tegra 2 T20 (dual core ARM Cortex-A9), a ULP GeForce GPU, 1 GiB of main memory and 32 GiB of eMMC NAND flash memory, it ranges among the most powerful embedded ARM platforms available in the first generation of Android tablets.

* wolfgang.mauerer@siemens.com

II. ARCHITECTURE

A. Kernel

RT support by the operating system kernel is obviously a mandatory requirement for RT architectures. With Android being irrevocably tied to the Linux kernel, this suggests using one of the stock real-time solutions for Linux kernels. A cornucopia of approaches was introduced over time (*e.g.*, RTAI [18], RTLinux [19], Xenomai [21], preempt_rt [4]). We have given focus to preempt_rt, mostly because the code is predicted to appear fully integrated into mainline Linux sometime in the near future.

It is important to emphasise that preempt_rt is not a RT kernel in the classical academic sense; owing to the complexity of the Linux kernel, it is impossible to prove upper bounds on all operations under all possible circumstances. However, statistical verification and test mechanisms have been established [6] that allow to determine upper latency bounds. The approach has gained considerable acceptance in industry.

B. Userland

The Android userland is heavily based on the Dalvik virtual machine (DVM), which is essentially a Java virtual machine founded on a register- instead of a stack-based architecture [2]. Two possible approaches to augment the system with RT capabilities are therefore [12]:

- ❏ Replace the Dalvik VM with a classical, RT capable JVM (see Ref. [16]), and adapt the Android userland where necessary.
- ❏ Augment the DVM with real-time capabilities, leveraging techniques and design patterns from existing RT-JVM implementations.

Following a code analysis, we estimated that the effort for any of these approaches exceeds not only our constrained developer resources, but would also in general provide an insufficient return on investment. Instead, we decided to largely separate the Java and RT components, as the architecture diagram in Figure 1 summarises.

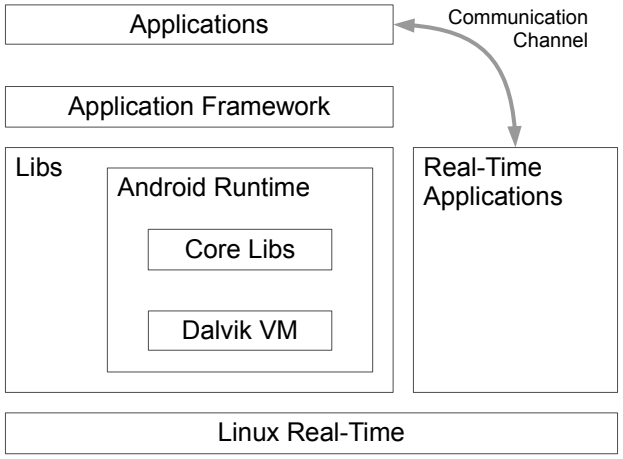

FIG. 1. Architecture diagram of our real-time Android implementation.

The architecture mandates that RT components are written in C (or any other low-level language that can target the machine directly without intermediate translation by a non-RT VM). We have used transactional techniques [1, 7] to allow for data exchange between the application domain (based on Java and the Android APIs) and the real-time domain.

C. Data exchange

When setting up a data exchange path between real-time and non real-time tasks, special care needs to be taken to not introduce unbounded delays in the RT domain. One of the simplest ways to communicate between two processes is to set up a dedicated region of shared memory that is available to both partners, and to use mutual exclusion to avoid data corruption. This scenario, however, can easily leads to unbounded latencies: By locking the shared region in the non-RT task, the RT task be be delayed indefinitely when the lock needs to be acquired there[1]. We therefore employ a synchronisation algorithm that is similar to Linux sequence locks (seqlocks) combined with a publish-subscribe pattern. This allows for lock-free communication in both directions and is always wait-free for the real-time part. Data is stored in a special shared memory region which is locked into physical RAM.

When the real-time process wants to modify the shared data, it increments an atomic counter before and after the write access – odd counter values therefore signal an on-going write operation. The reader retries any interleaved read operations until an identical counter value is observed before and after the read access (when an odd value is observed before the read, a wait cycle can be inserted since it is clear that the data are currently being modified). In the opposite direction, when the non-RT part acts as producer, it creates a modified copy of the data in a second buffer. Finally, it checks whether the copy source is still consistent and unchanged, and atomically switches between the buffers.

This ensures that the HMI layer can issue directives to the control part (*e.g.*, to adapt algorithmic parameters), while the RT layer can pass data (*e.g.*, information about noteworthy events) to the HMI part.

III. IMPLEMENTATION AND EVALUATION

A. Basis Component Integration

While the vanilla kernel neither fully supports real-time aspects nor all the Android extensions, the corresponding patch stacks are provided as source additions (respectively as git repositories [3]) to selected versions of the Linux kernel

Unfortunately, preempt_rt is only available for a limited number of base kernels, see Table I. Patches required for the Xoom board (and supplied by Motorola/Google [13] in their board support package (BSP) are based on Linux 2.6.36 [20], with no ports for other kernel releases.[2] The Android patch stack is available for nearly every recent kernel release.

[1] This can even lead to *priority inversion* when the non real-time task is itself blocked by a third task with a priority in-between both communication partners. Effectively, the mid-prio task now blocks the high-prio (RT) task. There are solutions for such situations—however, avoiding them in the first place is always desirable.

[2] Notice that the generic Nvidia Tegra *platform* infrastructure is integrated into the mainline Linux kernel, and is at the time of writing being actively developed by Google [9]. However, support for the Xoom *board* (on top of the platform code) is not mainlined.

Component	Version	Release Date
Linux Kernel	2.6.33	24. Feb. 2010
	2.6.36	20. Oct. 2010
	2.6.39	19. May 2011
	3.0	22. Jul. 2011
preempt_rt	2.6.33	
patch stack	3.0	
	3.2,4	
Android	2.6.25,27,29,32	Immediately
patch stack	2.6.35–39	after kernel
	3.0	releases
	3.3,4	
Xoom BSP	2.6.36	

TABLE I. Versions and release dates of relevant basis components.

The components that are most difficult to port across kernel versions are the BSP and preempt_rt. This, essentially, suggests two integration strategies:[3]

- ❏ Based on the preempt_rt release 2.6.33, back-port the BSP from 36 to 33, and forward-port the Android patch set from 32 to 33.
- ❏ Based on the preempt_rt release 3.0, forward-port the BSP from 2.6.36 to 3.0, and integrate the Android patches into the result.

Code analysis and experiments revealed that the latter option can be realised with less effort than the first alternative. This is especially because during the 8 months long development cycle between 2.6.33 and 2.6.36, much crucial infrastructure for the Tegra platform was merged into the mainline kernel and would need to be back-ported. The overall substantial amount of changes continuously faced by the Linux kernel [11] forms another strong argument against a back-port of large bodies of code across multiple kernel releases.

A graphical outline of our integration strategy is shown in Figure 2. Combining the preempt_rt and Android patches turned out to be uninvolved; the stacks are mostly orthogonal. Section III G provides quantitative details on the porting efforts.

B. Shared-memory

Android doesn't use classical POSIX shared memory, but provides ashmem, the Android "Anonymous Shared Memory Subsystem". Ashmem is specifically designed to allow shrinking when the system is in a low-memory situation. For real-time operation, however, locking shared pages in physical memory is desirable, which is not supported by Ashmem.

Therefore, we introduced a custom shared memory provider for Android.

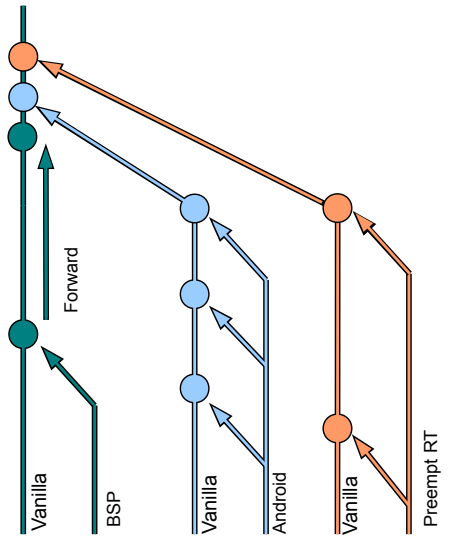

FIG. 2. Source code integration strategy for kernel components required to support the preempt_rt extensions and the Android userland.

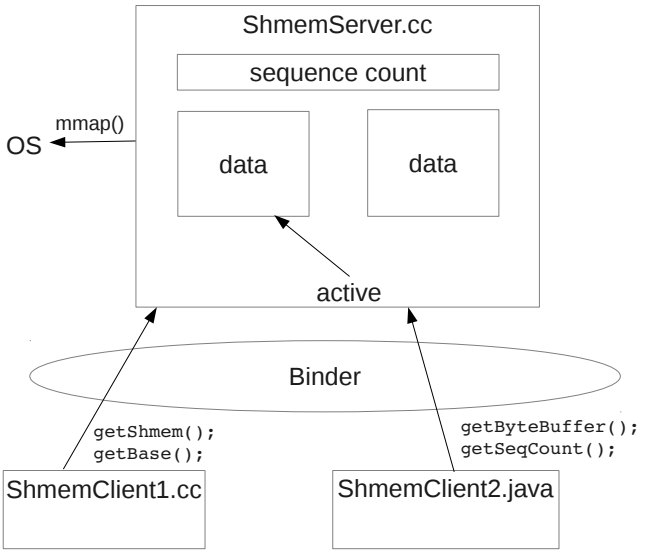

FIG. 3. The shared-memory server *ShmemServer* allocates a shared region. Native as well as Java processes can get a reference to the region via Binder remote method invocation and afterwards access it directly for data exchange.

A system service creates the shared memory region via the *mmap()* system call and registers itself at Binder,

[3] Another alternative is to start with the BSP kernel 2.6.36, and forward-port preempt_rt from 33 to 36. Unfortunately, the preempt_rt extensions do not just modify crucial core components of the kernel, but are also only available as one large patch file for 2.6.33, which means that the required effort would by far exceed the other variants.

the Android inter process communication central. All programs which want to use the shared memory region need to obtain a reference via Binder calls. Native clients receive a pointer they can use directly, while a JNI library provides access methods for Android apps that make data and the sequence counter available as Java data types (*ByteBuffer* and *int*).

C. Demonstration

A simple setup to ensure that responses to external stimuli can indeed be satisfied under hard temporal constraints is given in Figure 4. Although the Tegra platform offers real-time capable GPIO pins in abundance, they are unfortunately physically inaccessible on the Xoom board. The "official" communication interface is based on USB, which is an inherently problematic protocol in real-time contexts.

To circumvent the problem, we abuse two GPIO pins integrated into the HDMI video cable: One is supplied with periodic signals from a signal generator. Upon signal reception, the system is notified via interrupt. The threaded interrupt handler (subjected to regular kernel scheduling, but running at real-time priority) sets up a timer, which upon expiration emits a signal via the second GPIO pin. This pin, in turn, is connected to an oscilloscope. Given stimulus frequency and timer duration, it is therefore possible to determine system latencies with an external measurement.

The demonstration exposes two main requirements of a real-time Android platform: Appropriate hardware to implement the HMI concept (especially accelerated graphics chips and a multi-touch capable display), and a sufficient amount of real-time capable input/output ports, for instance, serial lines or GPIO ports.

Unfortunately, no commercially available platform known to us satisfies all requirements,[4] so an appropriate co-design of hardware and software is required from the very onset of a real-time Android implementation project.

D. Challenges

The major aim of our work was directed at merging RT Linux techniques and the Android HMI capabilities. Unfortunately, evaluating Android development platforms as provided by several major ARM manufacturers exposed numerous hardware deficiencies regarding the prerequisites for a proper Android user experience, especially a suitable form factor and state-of-the-art touch screen capabilities. Consequently, we focused on Android consumer tablets as development platform. Owing to GPL [10] provisions, the kernel trees need to be made available in source form, including necessary hardware drivers.

Under the assumption of stable hardware support and minimal driver porting efforts, we planned most person-power on integrating the Android mechanisms with RT features. Contrariwise, it turned out that Android and preempt_rt integrated cleanly, while forward ports of the BSP patch sets exceeded the estimations by orders of magnitude. One of the prevalent issues was to isolate BSP patches from (often sub-optimally tended) manufacturer kernel trees.[5]

For the eventually selected Xoom platform, porting the BSP required less, but still substantial involvement—the numbers discussed in Section III G clearly indicate that the manual effort for this task considerably outweighs the RT-Android integration work.

It is worth emphasising that the ease of porting depends not only on the kernel, but on the availability of the source code for the *complete* platform: The Android userland is largely governed by BSD-like licenses, which implies that Google is not bound to release them to the general public [10]. While the latest version of the Android platform is conveniently available in source form (even including proper revision control history) at the time of writing, the situation was different when this work was performed: Only binary images of Android 3.1 were available. As an illustrative example, ioctl ABI changes in the graphics subsystem forced us to provide compatibility code so that the userland was also operational on kernel 3.0. With the source code available, the issue could have been solved by a simple recompilation.

E. Architectural Flexibility

Interestingly, we did not observe any potential issues with the Android stack that would put any particular real-time implementation for Linux into advantage. Ipipe/Xenomai [21], or also more focused implementations as presented in Ref. [5] would serve as equally apt basis technologies. While details of the transactional data exchange between RT and non-RT parts of the system would likely require some adaption, the core concept can remain identical.

---

[4] Some system vendors offer evaluation platforms that formally satisfy both, but they typically require considerable polishing to reach commercial grade hardware usability.

[5] As a specific example, take Samsung's BSP for the Galaxy Tab using Linux 2.6.32.9: The revision seems tantalisingly close to kernel 2.6.33 supported by the preempt_rt patch set. We learned, though, that the BSP was based on an internal Samsung kernel tree intermingling crucial device and board support patches with hundreds of unrelated modifications from different source trees. It also contained a considerably modified USB core based on Linux 2.6.29. Irregardless of any other tasks, extracting all mandatory patches from this kernel tree already exceeded the predicted cumulative porting efforts.

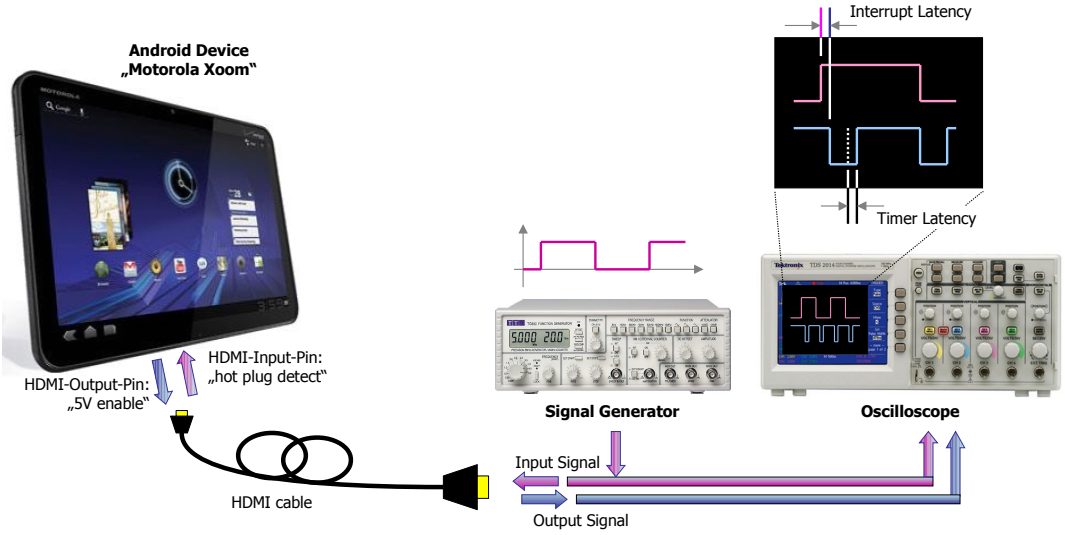

FIG. 4. Experimental setup to demonstrate time-constrained reactions of the real-time Android prototype to an external stimulus. See the text for details.

F. Latency Measurements

To ensure that no code paths leading to unacceptable latencies remain in our architecture, we have performed the usual measurements based on synthetic tests (see, *e.g.*, Ref. [6]). To obtain more comprehensive information about how the overall observed latency is composed of contributions by individual kernel actions, additional instrumentation based on the Ftrace framework [17] was used in the data collection phase. The results of a typical measurement run are visualised in Figure 5.[6]

The measurement is based on a userland application with RT priority that registers timers on a temporally equidistant grid. The difference between the moment when the timer is supposed to expire and the moment when the test program reacts to the expiration is observed as *latency*, which has to be bounded to fulfil the RT premise. Several contributions (illustrated in Fig. 5) sum up to the total latency :

- ❏ The *Expiration delay* measures the difference between the moments when the hardware timer was scheduled to expire, and when it actually expired.
- ❏ The duration between timer expiration and the start of high resolution timer (HRT) processing[7] is denoted as *HRT handler delay*.
- ❏ The time required to process the HRT handler is the *HRT duration*
- ❏ After HRT handler processing, control needs to be passed to the userland RT application. *Schedule in* accounts for the duration between the in-kernel decision to request control for the task, and the actual invocation.
- ❏ To measure the difference between expected timer expiration and current time, the application needs to use the `getclock` mechanism to learn the current time. The overhead of this operation is labelled as *Getclock*.

The bottom portion of Fig. 5 shows a boxplot (*i.e.*, a non-parametric statistical summary, see, *e.g.*, [15]) of the latency distribution for each contribution; the individual events are represented by jittered dots. By connecting the contributions of the 10 largest total latencies, we see that they are consistently composed of outliers in each category, which allows us for inferring that the upper bounds are solely caused by general system effects, not by any asynchronous events that would necessitate code modification.

Before the described state was reached, some kernel optimisations were required; the most notable modifications are needed for the display subsystem which, in its original form, relies heavily on flushing all system caches, causing unacceptable latencies of tens of milliseconds.

G. Efforts

To provide an indication on how much effort is required to construct a basis kernel for a real-time capa-

[6] Owing to the unavailability of means to automatically stimulate hardware touch-screen events, the length of the measurement interval used to obtain the data is not yet fully satisfactory.

[7] High resolution timers are the backbone of preempt_rt time handling; they improve upon classical Linux timers whose resolution is insufficient for RT requirements.

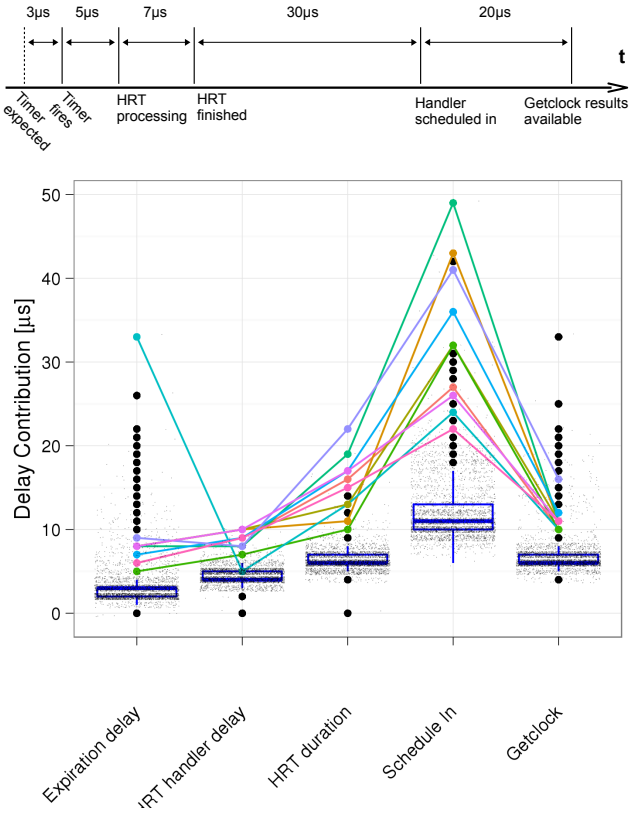

FIG. 5. Component-resolved latency measurements (top: illustration, bottom: results). Parallel to executing the synthetic benchmark, the system was subjected to substantial non-realtime load. See the text for a detailed discussion.

ble Android system, Figure 6 shows how many patches (classified by component) were required on top of vanilla Linux 3.0. While the individual patches exhibit considerable variation in size and complexity, there is no bunching of higher or lower complexity in any particular component. Taken cum grano salis, the numbers offer reasonably accurate guidance.

[1] J. Anderson, S. Ramamurthy and K. Jeffay, *Real-Time Computing with Lock-Free Shared Objects*, ACM Transactions on Computer Systems, 1997

[2] D. Bernstein, *Dalvik VM Internals*, Google I/O Conference, 2008

[3] S. Chacon: *Pro Git*, Apress, 2009

[4] H. Egger, C. Emde und Th. Gleixner, *Linux: Embedded für alle*, Elektronik Embedded 11/2011

[5] R. Graf, *Transparent Hard Real Time Behavior, a new Linux-Based Approach*, Proc. Emb. World 2010

[6] C. Emde, Echtzeit im Echtheits-Test, Elektronik 4/2011

[7] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*, Morgan Kaufmann, 2008

[8] Git repository for Android kernel contributions: `https://android.googlesource.com/kernel/common.git`

[9] Git repository for Tegra platform support development: `https://android.googlesource.com/kernel/tegra.git`

[10] A. M. St. Laurent, *Understanding Open Source & Free Software Licensing*, O'Reilly, 2004

[11] W. Mauerer, *Professional Linux Kernel Architecture*, Wiley/Wrox, 2008

[12] C. Maya et al., *Evaluating Android OS for Embedded Real-Time Systems*, Proc. 6th Int. WS on OS Plat. for Emb. RT Appl., 2010.

[13] Motorola Xoom BSP home page: `http://sourceforge.net/motorola/xoom/home/Home/`

[14] D. Norman, *The Design of Everyday Things*, Perseus books, 2002

[15] R. Peck and J. L. Devore, *Statistics*, Brooks/Cole, 2011

[16] D. Rollella and J. Gosling, *The Real-Time Specification for Java*, IEEE Computer 33(6), 2000

[17] S. Rostedt, *Finding Origins of Latencies Using Ftrace*. Proc. RT Linux WS., 2009

[18] RTAI Project home page: `http://www.rtai.org`

[19] RTLinux project home page: `http://www.rtlinuxfree.com`

[20] Vanilla kernel distribution web site: `http://www.kernel.org`

[21] Xenomai Project home page: `http://www.xenomai.org`

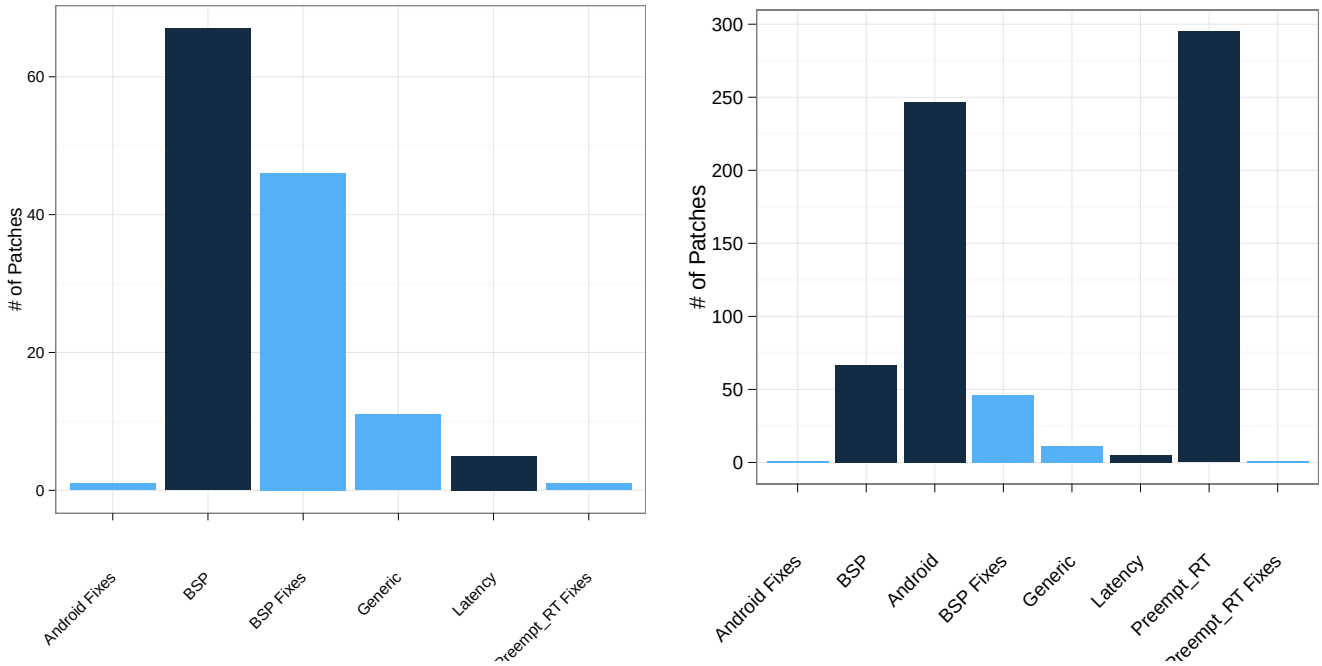

FIG. 6. Number of patches required to augment the vanilla Linux kernel 3.0 with capabilities for both, real-time and Android support. To emphasise the manual effort (red) required beyond the mechanic application of readily available, but large patch stacks (blue) for preempt_rt and the Android specific components, the left hand side omits these data. A complete summary is provided on the right hand side; please note the considerably larger ordinate span. The patches were categorised by a subjective manual selection.