doi: 10.15496/publikation-29062

Feeding the Masses: DNBD3

Simple, efficient, redundant block device for large scale HPC, Cloud and PC pool installations

Simon Rettberg Dirk von Suchodoletz Jonathan Bauer

eScience Department, Computer Center, University of Freiburg, Freiburg, Germany

In computer center operations many sites operate large PC lecture pools or HPC clusters which can require similar or identical operating system images and software packages. Booting over the LAN allows instantaneously usable systems but requires the efficient provisioning of the root file system. Traditionally, general purpose file systems like NFS are used, but read-only Network Block Devices like the presented DNBD3 provide a range of attractive features, which can outperform alternatives across a range of situations. DNBD3 not only allows for caching and proxying at various levels, but it comes with a built-in performance monitor, versioning, and failover functionality. DNBD3 has been under development at Freiburg University for the past few years. It is released under a GPLv2 license, and consists of a Linux kernel module for the clients, and a user space executable for the servers. It is running in production for two highly heterogeneous use cases: as a distributed setup of campus-wide computer pools with more than 400 connected machines, and in the 1000+ node compute cluster backing the Freiburg HPC and Clouds. Aggressive local caching might even allow the use of mobile clients on WLAN infrastructures in stateless Linux operation.

1 Motivation

Management of large scale desktop computer pools, HPC clusters, and private clouds all usually incur high administrative costs or operational expenses that can easily surpass the figures for capital expenditures such as hard- and software purchases. One strategy to reduce the Total Cost of Ownership (TCO) is to employ

doi: 10.15496/publikation-29056

the use of remote boot options utilizing pre-existing high bandwidth local networks. While per-GB cost of centralized reliable storage is usually higher, maintaining such a system often requires less manual work, especially if the storage is shared with other projects and scaled accordingly.[1]

Exporting the root file system (rootfs) of many clients via NFS has been shown to be an inefficient solution due to significant file system overhead and serialized round trips for access operations. While iPXE initial network boot is very efficient (Bauer, Messner et al., 2019), highly concurrent access to a central NFS server from many machines booting simultaneously can result in bandwidth bottlenecks resulting in poor end-user experience (Stirenko et al., 2013).

Both the ability to provide failover in the case of a server outage and to provide load-balancing is a requirement in large scale setups of 100+ machines. With NFS versions prior to 4.0, providing NFS server redundancy was cumbersome[2] and active load-balancing impossible. This improved marginally in version 4.1, which includes Parallel NFS. More promising results were produced with Network Block Devices (NBDs): During initial development, only a compressed squashfs file system image was distributed by NBD (Schmelzer et al., 2014). Since NBDs work only at the block level, they have significantly less overhead compared to NFS.

Both in PC lecture pools and compute clusters (HPC, Clouds), the machines share a broad software base, which is ideally highly redundant between the use cases for operator convenience. Stateless operation, and thus easy exchange of clients between operating roles, is significantly more important than the individualization of single clients.

2 Previous experiments and alternate developments

Block devices besides traditional local disks and optical storage started to emerge by the end of the 1990s in the Linux kernel.[3] After the original NBD, derived variants like GNBD or DRBD emerged (Kim et al., 2001). iSCSI could be seen as

[1] In case of computer pools, compare swapping a disk in your redundant central storage vs. locating a workstation somewhere on campus and replacing its only disk.

[2] NFSv3 is stateless, so it allowed one server in »hot standby« to detect another server's disappearance and simply take over its IP address. However, this is rather abusing the statelessness of NFSv3 than an elegant integrated solution, often resulting in client-side hangs approaching 10+ seconds.

[3] The Linux Network Block Device was developed in 1997 by Pavel Machek and first included into the standard Linux kernel with version 2.1.101

falling into a similar category but is not made for highly concurrent access.[4] As a result of iSCSI being implemented largely by wrapping SCSI commands in TCP/IP, it is comparatively inefficient due to the additional overhead. This led to various extensions to solve the underlying issue, such as multipath routing, which uses several TCP/IP connections to parallelize operations against multiple endpoints, or iSER, which can greatly speed up bulk data transfers by leveraging RDMA on suitable transports, like InfiniBand. Since iSCSI supports read and write operations for disk targets, the implementation is more complex than required for the use case envisioned. Our goal was to fully saturate a traditional 10GbE link without requiring excessive resources or special hardware on either end, apart from the network card and fast storage or enough RAM to deliver data at that speed.

DRBD, like iSCSI, implements a read/write block device, but it is focused on live replication of written data amongst secondary DRBD hosts. It focuses on providing quick failover of a service to another host rather than providing redundancy to a multitude of clients, making it unsuitable for the use case. Previous projects with similar goals (ENBD, ANBD, GNBD) appear to be dead and are no longer maintained.

The first iteration of the Distributed Network Block Device (DNBD1) implemented multicast-like features to allow access to a read-only NBD over shared media such as WLANs.[5] To cope with the intrinsic communications overhead of wireless networks, DNBD1 aimed to avoid sending out the same data over and over again to every single client. This was achieved by connecting wireless clients and servers in a multicast network on the IP layer. DNBD1 clients take advantage of this overhead by pre-caching data requested by other DNBD1 clients. In ideal situations, all redundant requests can be prevented, greatly reducing bandwidth consumption on the shared medium. The development of the superseding versions of DNBD2 and DNBD3 catered to the fact that the wireless future didn't arrive as quickly as envisioned and brought replication and scalability features to the standard unicast LAN environment.[6]

The latest version of the Distributed Network Block Device, DNBD3, exclusively focuses on providing a redundant and fast, read-only block device which greatly

[4] Setting a read-only target allows multiple clients to connect, though.

[5] Since in 2006, it was assumed that soon everybody would exclusively use laptops or similar mobile devices.

[6] See `https://lab.openslx.org/attachments/download/19/ba-dnbd2-dileo.pdf` (visited on 10.01.2019).

reduces implementation and operational complexity, especially on the server side. The kernel module compiles with all major Linux kernels. The server is a user land executable written in C, mostly compatible to standard POSIX environments.[7] DNBD3 is a complete rewrite of previous iterations, the main objective of which is to combine the advantages of the standard NBD with caching, versioning, and wide area distribution features.[8] The optimal utilization of the available bandwidth is achieved through strong parallelization and efficient handling of I/O intensive operations within the Linux kernel. Resilience is realized through deploying redundant servers and allowing clients to distribute across the available servers. The clients independently search for the fastest server and respond dynamically to bottlenecks, ensuring the best possible performance even when used in wide area networks.

3 The DNBD3 architecture

Since DNBD3 was designed to suit the use case of many read-only clients, a great deal of complexity supporting distributed read-write access could be completely omitted. Emphasis was instead laid on supporting quick failover in case of connectivity issues, and load-balancing in case of apparent network congestion (Figure 1). Since the operational requirements for this project included not only providing the root file system to workstations but ensuring quick response times, i. e. no multiple second long client hangs while reconnecting to another server, it was made of utmost importance that the protocol incurs as little overhead as possible, especially regarding connection establishment.

3.1 Server and client setup

A typical setup involves a primary server, which exports one or more images from a configurable file system location. Images are addressed by their relative path within the exported directory. Additional DNBD3 servers function as proxies, pointed at the primary server, initially not storing any images or data. In this setup, a client can connect to any of these servers and request an image. When a client requests an image from a proxy that the proxy server does not know about, it will transparently

[7] Currently tested on Linux and FreeBSD, could be rather easily ported to similar systems, too, given they have a comparable mechanism to sendfile(2).

[8] The source is available at `https://git.openslx.org/dnbd3.git/` (visited on 10. 01. 2019).

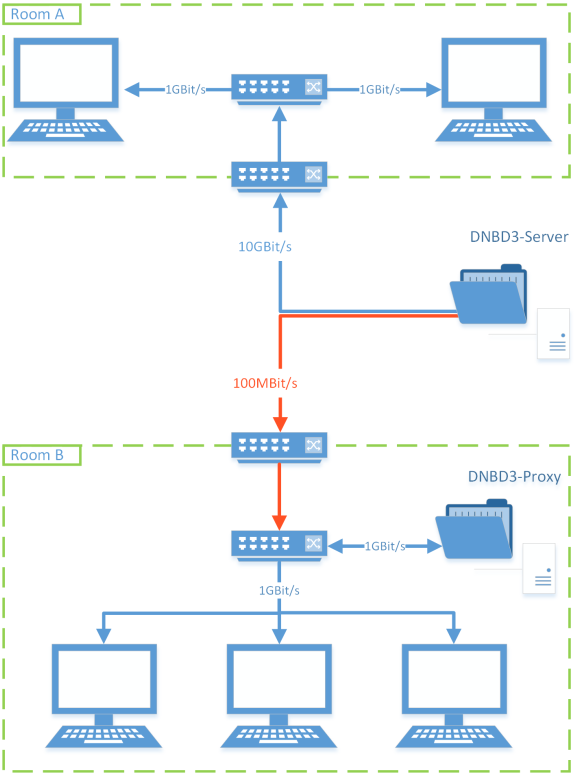


Figure 1: Distributed cache and proxy cascade for performance to deal with low bandwidth links.

forward the requests to the primary server, relaying all replies to the client. Simultaneously, it will cache those replies locally for future requests. Proxy servers will automatically evict the least recently used image from the cache when the proxies run out of space.

To (dis)connect a DNBD3 client device, the user space binary `dnbd3-client` is executed. It can be passed one or more server addresses to work with, and requires the desired image name and revision number, which are then passed to the kernel module via `ioctl`. Once the kernel module has established a connection to a server, it will request a list of suggested servers[9] to use for failover and load balancing. This means the server list used by the client can be pre-populated when initializing the client (e. g. embedded in initramfs and randomized on each boot) and optionally

[9] From the client's viewpoint servers and proxies are functionally the same.

extended later on by querying servers for additional addresses. During operation, DNBD3 does not do any additional caching on the client, but merely acts as a plain old block device. Since the general use case means ultimately mounting a file system from the device, caching will eventually happen higher up the stack, using the global page cache.

3.2 Load balancing and failover

Load balancing is kept fairly simple in DNBD3; each client individually measures and decides which of the available servers to use for requesting blocks. While it is tempting to design an algorithm where the involved servers coordinate distributing clients among them according to their NIC speed, this approach falls short as soon as the network infrastructure between servers and clients is any more complex than one or two switches. Individual clients need to accurately track bottlenecks that only affect clients in certain subnets for globally optimal performance. Simply distributing clients evenly across servers will certainly not be sufficient, especially under fluctuating network loads.

Instead of going down a rabbit hole of designing a highly sophisticated algorithm to master this task, we opted for clients to periodically measure the speed of every known server. If another server's response time is lower by 33% compared to the current server, the client will make a switch. To account for occasional spikes, the average of four such measurements is used when comparing servers.

A server's speed is determined by establishing a connection, requesting the image currently in use by the client, and reading the first 4096 bytes of that image.[10] This ensures enough network round trips to produce a meaningful result without consuming too much bandwidth. Requesting the currently used image from the server instead of having a synthetic benchmark feature in the protocol has two advantages: first, this ensures that the server being measured is actually able to serve the image in question, and second, since the full handshake has already happened after the measurement is done, there is zero networking overhead when deciding to actually switch over to that server, other than having to re-queue any in-flight requests that the old server didn't reply to yet. Switching to another server merely requires closing the current connection and then re-initializing the sending and

[10] Those first bytes will almost certainly reside in the server's cache in memory and thus omit the delays of actual disk reads, but it is generally a good enough proxy for the requirements of the protocol.

receiving threads with the new socket descriptor already established during the measurement.

After startup, clients measure every four seconds for the first 30 seconds in order to quickly switch to another server, should the initially selected server be overloaded. If no switch occurs within 30 seconds, the current server is assumed to be a good choice, and clients lower measurements to every 20 seconds. This approach, combined with the averaging over four measurements, results in a pretty stable distribution of clients among servers in both, the comparably spread out topology of bwLehrpool, and the much larger but more homogeneous network infrastructure of the HPC/NEMO setup.

3.3 sysfs interface

DNBD3 exports some simple statistics via sysfs: the currently connected server's address, the list of known servers, their measured response times, the current image, and its revision number.

```
# ll /sys/block/dnbd0/net/
total 0
drwxr-xr-x 2 root root    0 Jun 27 14:38 ./
drwxr-xr-x 9 root root    0 Jun 27 08:38 ../
-r--r--r-- 1 root root 4096 Jun 27 14:38 alt_server_num
-r--r--r-- 1 root root 4096 Jun 27 14:38 alt_servers
-r--r--r-- 1 root root 4096 Jun 27 14:38 cur_server_addr
-r--r--r-- 1 root root 4096 Jun 27 14:38 cur_server_rtt
-r--r--r-- 1 root root 4096 Jun 27 14:38 image_name
-r--r--r-- 1 root root 4096 Jun 27 14:38 rid
```

Example output of known server list:

```
# cat /sys/block/dnbd0/net/alt_servers
10.4.128.240,5003,426,0
10.16.0.22,5004,252,0
10.8.8.88,5003,452,0
10.23.4.2,5003,1071,0
```

This shows the list of servers known to serve the image currently in use. The columns are IP address, port, response time in µs and a counter which tracks how many times this server was consecutively found inaccessible.

3.4 Versioning

Versioning is an optional feature available in DNBD3. It allows for multiple versions of the image files to co-exist on the server by appending .r[0-9]+ to the image name, representing the revision number of the image. Virtually, the image is exported without this suffix, and clients can indicate which revision of an image they want when connecting to a server. The revision number 0 has a special meaning in this context – it refers to the highest revision number. This way, a client already running can explicitly request the revision number of an image that it is currently running on, while a freshly booted client can simply request revision 0 to get the latest version.

4 Client-side copy-on-write for r/w access

Creating a writable file system for the client OS on top of DNBD3 can be achieved in two ways. Up until recently, the source file system got packed with squashfs before it got exported as an image. On the client, the squashfs was mounted in read-only mode and then a tmpfs top layer was added via AUFS[11] to make the file system virtually writable. Squashfs offers great compression ratios and since copy-on-write (CoW) happens on the file system layer, the size of the rootfs is determined individually on the client side through the size of the tmpfs. Nevertheless, adding a single byte to a large file meant fully re-copying it into the tmpfs layer.

The second approach is exporting a disk image in any container format, e. g. qcow2, and then providing a CoW backing on the client with qemu-nbd on the block level. It is also possible to use the Linux kernel device mapper to create a writable layer for the image, but this requires exporting a raw image file which has all the unused space preallocated. The advantage of this approach is that it is possible to directly export the disk image of a virtual machine without the tedious squashfs packing process. Unfortunately, since this operates on the block layer, the size of the root file system will be predetermined during image creation, so using a

[11] Special unioning file system which never made it into the kernel because of significant complexities and resulting issues. Compared to OverlayFS (which wasn't part of the mainline kernel when bwLehrpool started), it offers much more flexibility, e. g. adding/removing layers after mounting.

small virtual disk during image creation would make it possible to run out of disk space on the client, even if plenty of space were available for the CoW file.[12]

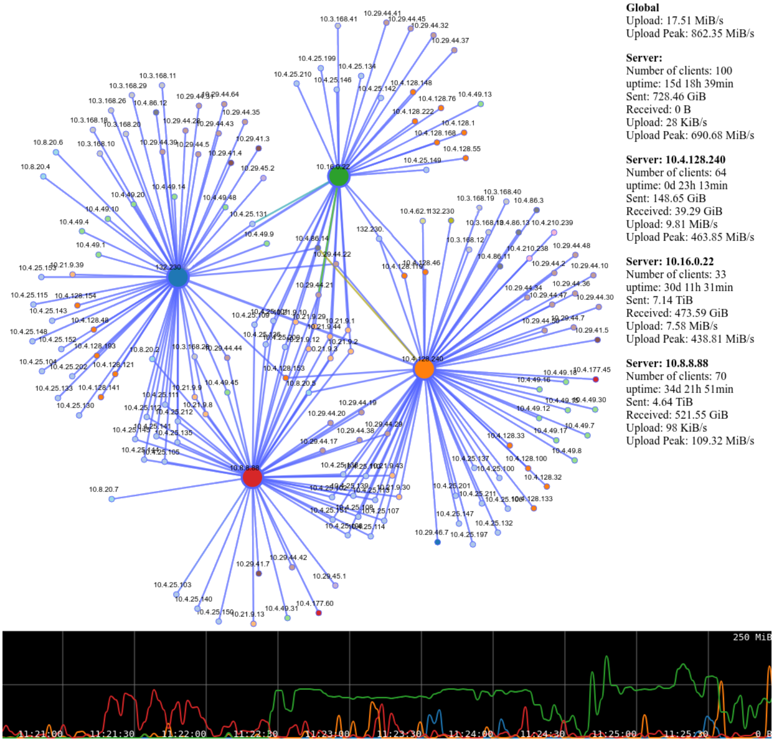
5 Practical experience

DNBD3 has been powering the bwLehrpool infrastructure since late 2013 (Figure 2). bwLehrpool (Suchodoletz et al., 2014) is a stateless Linux remote boot project offering various hypervisors and containerization on top to host »user created content«. It enables the flexible and efficient deployment of virtual teaching and laboratory software environments in computer lecture rooms. bwLehrpool offers instructors at cooperating educational institutions the possibility to quickly, simply, and independently create teaching environments for a wide range of courses. In bwLehrpool both the base operating system, as explained above, and, additionally, the various VM images are provided through DNBD3. Since it is desirable that the environment looks exactly the same every time the students start the VM complementing a lecture, any changes that happen on the block layer are temporarily written to a copy-on-write backing file on the client and discarded when the individual session ends.

Using DNBD3 for providing disk images to clients enables risk-free updating of DNBD3 servers and proxies without any service interruption. It even makes it possible to experiment with different server features and configurations, with the advantage of always being able to test against a real world workload.

While the virtualizers' own mechanisms are still used to realize the CoW layer for VM images, a writable layer was required for the rootfs images provisioning bare-metal machines. Since AUFS was never officially supported by the Linux kernel, it was cumbersome to patch its code to compile against newer kernel versions. The qcow2 format proved to be a good alternative to AUFS to implement CoW directly on the block layer, even though this relies on user-space tools to handle the format. Recent experiments with the device mapper framework were promising and have inspired us to work towards realizing the CoW layer entirely in the kernel-space.

[12] Actually, the block device can be enlarged when creating the CoW layer, and given a suitable partitioning and file system choice inside the container, the file system of the rootfs could be extended on the client, but it is usually not worth the effort. Making the base image large enough is usually much simpler and sufficient.



Global
Upload: 17.51 MiB/s
Upload Peak: 862.35 MiB/s

Server:
Number of clients: 100
uptime: 15d 18h 39min
Sent: 728.46 GiB
Received: 0 B
Upload: 28 KiB/s
Upload Peak: 690.68 MiB/s

Server: 10.4.128.240
Number of clients: 64
uptime: 0d 23h 13min
Sent: 148.65 GiB
Received: 39.29 GiB
Upload: 9.81 MiB/s
Upload Peak: 463.85 MiB/s

Server: 10.16.0.22
Number of clients: 33
uptime: 30d 11h 31min
Sent: 7.14 TiB
Received: 473.59 GiB
Upload: 7.58 MiB/s
Upload Peak: 438.81 MiB/s

Server: 10.8.8.88
Number of clients: 70
uptime: 34d 21h 51min
Sent: 4.64 TiB
Received: 521.55 GiB
Upload: 98 KiB/s
Upload Peak: 109.32 MiB/s

Figure 2: Example of the bwLehrpool DNBD3 distribution structure. The four big nodes represent DNBD3 servers, small nodes are connected clients. Some clients are currently using more than one image, so they can be connected to multiple servers at the same time. Blue edges are idle links, green edges represent network traffic. The graph at the bottom shows aggregated egress traffic for all four servers over the last 20 minutes.

Another DNBD3 use case, in production since the start of the new HPC cluster in 2016, is the provisioning of the root file system of NEMO in Freiburg. Over 1000 nodes are booted simultaneously while two DNBD3 servers provide their root file system. During the boot process, the 10 GbE network interfaces of both servers are mostly saturated while generating minimal CPU load, achieving short boot-times and stable operation. Updates are run gradually without interrupting the cluster operation: Node are drained and rebooted into a new version of the rootfs. The old version is provided in parallel to the new one as long as nodes continue to request it.

Without delving into the details, both the bwLehrpool and NEMO rootfs images are generated with a combination of packer templates to install the base OS from installation ISOs and ansible playbooks to install and configure the project's respective software stack (Bauer, Suchodoletz et al., 2019). The resulting qcow2 images are exported via DNBD3 and prepared for network boot with the same set of tools. This approach has proved to be flexible enough to cater for the different use cases.

6 Conclusion and outlook

DNBD3 has been in production for several years and has gracefully survived multiple server failures without interruption of service for connected clients. Thus, DNBD3 is not just another network block device, but fills a gap in the Linux block device selection as it combines features which are not provided by pre-existing designs. In the short term, some adaptations to newer kernel versions were necessary[13]. Compatibility with new kernel releases is currently maintained by the bwLehrpool team. In the long term, it is planned to do major cleanups and checks whether it still conforms to modern block device designs, which should eventually make it possible to suggest DNBD3 for upstreaming into the official kernel.

To allow a simpler and more efficient updating of user provided base and VM images, the use of a local CoW image on the client will be explored for seamless updating. So, aside from simply discarding the snapshots for stateless operations, the resulting diff-file from the CoW layer could be further processed to handle incremental updates on base images, both for server-side image versioning and even to implement client-side persistence. This would allow copying only the different blocks instead of the entire image whenever changes should be made to the served image. To allow DNBD3 operation in low or fluctuating bandwidth environments, a client-side disk caching feature will be explored in further experiments in order to enable bwLehrpool operation in mobile setups.

Acknowledgement

The work presented in this publication was part of the bwLehrpool project sponsored by the Ministry of Science, Research and the Arts Baden-Württemberg, Germany. NEMO is part of the state-wide HPC research infrastructure, which is supported

[13] As of writing, DNBD3 is known to work on all kernel versions from 3.0 to 4.19.

by both the DFG and the state of Baden-Württemberg. The support is gratefully acknowledged.

Corresponding Author

Simon Rettberg: simon.rettberg@rz.uni-freiburg.de
eScience Department, Computer Center, University of Freiburg
Hermann-Herder-Str. 10, 79104 Freiburg, Germany

ORCID

Dirk von Suchodoletz https://orcid.org/0000-0002-4382-5104
Jonathan Bauer https://orcid.org/0000-0002-5624-2055

References

Bauer, J., M. Messner et al. (2019). »A Sorting Hat For Clusters. Dynamic Provisioning of Compute Nodes for Colocated Large Scale Computational Research Infrastructures«. In: *Proceedings of the 5th bwHPC Symposium. HPC Activities in Baden-Württemberg.* Freiburg, September 2018. 5th bwHPC Symposium. Ed. by M. Janczyk, D. von Suchodoletz and B. Wiebelt. TLP, Tübingen, pp. 217–229. DOI: 10.15496/publikation-29055.

Bauer, J., D. von Suchodoletz, J. Vollmer and H. Rasche (2019). »Game of Templates. Deploying and (re-)using Virtualized Research Environments in High-Performance and High-Throughput Computing«. In: *Proceedings of the 5th bwHPC Symposium. HPC Activities in Baden-Württemberg.* Freiburg, September 2018. 5th bwHPC Symposium. Ed. by M. Janczyk, D. von Suchodoletz and B. Wiebelt. TLP, Tübingen, pp. 245–262. DOI: 10.15496/publikation-29057.

Kim, K., J.-S. Kim and S.-I. Jung (2001). »GNBD/VIA: a network block device over virtual interface architecture on Linux«. In: *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM.* IEEE.

Schmelzer, S., D. von Suchodoletz, M. Janczyk and G. Schneider (2014). »Flexible Cluster Node Provisioning in a Distributed Environment«. German. In: *Hochleistungsrechnen in Baden-Württemberg. Ausgewählte Aktivitäten im bwGRiD 2012.* Beiträge zu Anwenderprojekten und Infrastruktur im bwGRiD im Jahr 2012. Ed. by J. C. Schulz and S. Hermann. KIT Scientific Publishing, Karlsruhe, pp. 203–219. ISBN: 978-3-7315-0196-1. DOI: 10.5445/KSP/1000039516. URN: urn:nbn:de:0072-395167.

Stirenko, S., O. Zinenko and D. Gribenko (2013). »Dual-layer hardware and software management in cluster systems«. In: *Proc. Third Int. Conf. »High Performance Computing« HPC-UA*, pp. 380–385.

Suchodoletz, D. von et al. (2014). »bwLehrpool – ein landesweiter Dienst für die Bereitstellung von PC-Pools in virtualisierter Umgebung für Lehre und Forschung«. In: *PIK – Praxis der Informationsverarbeitung und Kommunikation* 37.1, pp. 33–40. DOI: 10.1515/pik-2013-0046.