# IP Only Server

Muli Ben-Yehuda[1] Oleg Goldshmidt[1] Elliot K. Kolodner[1] Zorik Machulsky[1]
Vadim Makhervaks[2] Julian Satran[1] Marc Segal[3] Leah Shalev[1]
Ilan Shimony[1]

*IBM Haifa Research Laboratory*

[1]{muli,olegg,kolodner,machulsk,satran,leah,ishimony}@il.ibm.com
[2]vadim.makhervaks@gmail.com
[3]marcs@cs.technion.ac.il

## Abstract

Present day servers must support a variety of legacy I/O devices and protocols that are rarely used in the day to day server operation, at a significant cost in board layout complexity, reliability, power consumption, heat dissipation, and ease of management. We present a design of an IP Only Server, which has a single, unified I/O interface: IP network. All of the server's I/O is emulated and redirected over IP/Ethernet to a remote management station, except for the hard disks which are accessed via iSCSI. The emulation is done in hardware, and is available from power-on to shutdown, including the pre-OS and post-OS (crash) stages, unlike alternative solutions such as VNC that can only function when the OS is operational. The server's software stack — the BIOS, the OS, and applications — will run without any modifications.

We have developed a prototype IP Only Server, based on a COTS FPGA running our embedded I/O emulation firmware. The remote station is a commodity PC running a VNC client for video, keyboard and mouse. Initial performance evaluations with unmodified BIOS and Windows and Linux operating systems indicate negligible network overhead and acceptable user experience. This prototype is the first attempt to create a diskless and headless x86 server that runs unmodified industry standard software (BIOS, OS, and applications).

## 1 Introduction

Present day server systems support the same set of I/O devices, controllers, and protocols as desktop computers, including keyboard, mouse, video, IDE and/or SCSI hard disks, floppy, CD-ROM, USB, serial and parallel ports, and quite a few others. Most of these devices are not utilized during the normal server operation. The data disks are frequently remote, and both Fibre Channel and iSCSI now support booting off remote disk devices, so directly attached hard disks are not necessary for operating system (OS) boot either. Removable media devices, such as floppies and CD-ROMs, are only used for installation of OS and applications, and that can also be avoided with modern remote storage management systems.

Moreover, there are no users who work directly on the server, using keyboard, mouse, and display — normal administrative tasks are usually performed over remote connections, at least while the server is operational. Remote management is done via protocols such as Secure Shell (SSH) and X for Linux/UNIX systems, Microsoft's Windows Terminal Services, and cross-platform protocols such as the Remote Framebuffer (RFB, see [6]), used by the popular Virtual Network Computing (VNC) remote display scheme. However, local console access is still required for some operations, including low-level BIOS configuration (pre-OS environment) and dealing with failures, such as the Windows "blue screen of death" and Linux kernel panics (post-OS environments). Local console is usually provided either via the regular KVM (keyboard-video-mouse) interface or a serial line connection.

The legacy protocols and the associated hardware have non-negligible costs. The board must contain and support the multitude of controllers and the associated auxiliary electrical components each of them requires. This occupies a significant portion of the board real estate that could otherwise contain, say, an additional CPU or memory. The multitude and complexity of the legacy components also reduce the mean time between failures (MTBF).

We propose that future servers will only need CPUs, memory, a northbridge, and network interface cards (NICs). All the legacy I/O that is done today, e.g., over PCI, will be done over a single, universal I/O link — the ubiquitous IP network. All communication with storage devices, including boot, will be done over iSCSI, console access will also be performed over the network. Protocols such as USB can also be emulated over IP [5], pro-

viding a variety of remote peripherals such as CD-ROM, printers, or floppy if they are needed.

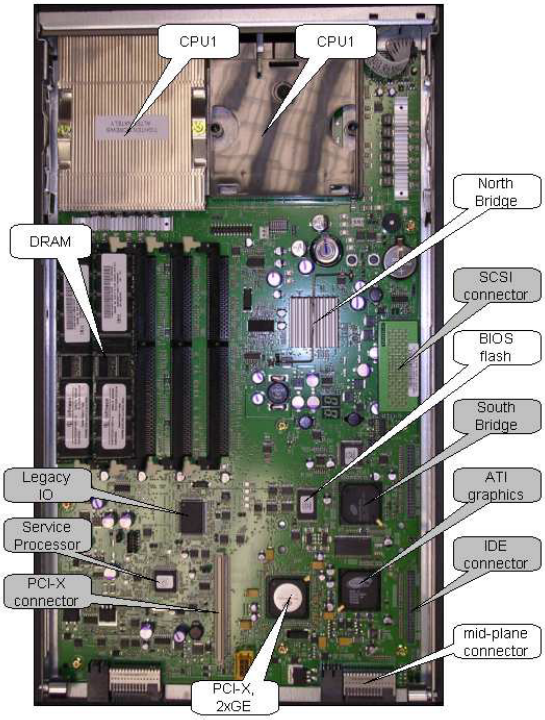


Figure 1: Components of an IBM HS20 blade server.

To illustrate this point, Figure 1 identifies the various components of an x86 server (an IBM HS20 blade server chosen for its unobstructed layout). The CPUs, the DRAM, the northbridge, the BIOS flash, and the network hardware are necessary, while the southbridge, the SCSI and IDE controllers, the graphics adapter, and the integrated legacy I/O chip (implementing the keyboard and mouse controllers, various timers, etc.) can be removed and their functionality can be emulated over the network.

This "remoting" of I/O can be achieved without any modifications at all to the applications, the OS, or the BIOS of the server if the protocol emulation is done in hardware. Substituting a single hardware component for all the legacy controllers, capturing all the bus transactions involving the legacy devices, remoting the transactions over the IP network, and performing the actual I/O at remote systems will be completely transparent to the BIOS and the OS, and thus to the applications.

While many software based alternatives exist for remoting I/O when the OS is up and running (see Section 2), doing the protocol emulation in hardware is essential for supporting the pre-OS (e.g., BIOS or bootloader) and post-OS environments.

We developed a research prototype of such an "IP Only Server" that uses IP for all of its I/O needs. We designed and implemented a legacy I/O emulator based on a COTS FPGA. The FPGA, connected to the host via the PCI bus, serves as the local keyboard, mouse, and VGA controller. All keyboard, mouse, and VGA traffic reaches the FPGA and is sent to a remote station over IP. The user is able to perform all the management operations — throughout the lifetime of the server, i.e., during boot, BIOS, OS initialization, normal operation, and post-OS stages — from the remote station. No changes were needed for the software running on the host. In particular, neither the BIOS nor the OS were modified.

The performance of the prototype is acceptable for the usual server management tasks. The only significant network load may come from the remote iSCSI storage, the network utilization due to remote management is small, and the user experience is acceptable.

## 2 Related Work

The concept of allowing the user to interact with a remote computer over a network had a long history. Today there exists a wide variety of thin clients, e.g., SSH/X11, VNC, SunRay, Citrix Metaframe, and Windows Terminal Server. Our approach differs from all these solutions in two major ways: we allow remoting of legacy I/O over the network a) without any host software modifications, and b) from the moment the computer has powered on until it has powered off, including when no operating system is present (BIOS stage as well as OS crash).

The Super Dense Server research prototype [7] presented servers with no legacy I/O support. It used Console Over Ethernet, which is OS dependent, supported text mode only, ran Linux, and used LinuxBIOS [4] rather than a conventional BIOS. In contrast, the IP Only Server runs unmodified OSes and BIOSes and supports graphical VGA modes as well as text based modes. Attempting sweeping changes in the BIOS (e.g., switching to LinuxBIOS) while requiring to support a wide variety of boards and many different software stacks would adversely affect reliability, availability, serviceability, and system testing.

"USB over IP" [5] is used as a peripheral bus extension over an IP network. USB/IP requires a special OS-specific driver and thus is only available when the OS is operational, while the IP Only Server does not require OS modifications — it listens on the PCI bus for transactions using a hardware component.

Baratto et al. presented THINC [3], a remote display architecture that intercepts application display commands at the OS's device driver interface. This approach looks promising for remoting the display while the OS is running, but does not handle either pre-OS or exception (post-OS) conditions. THINC could be used together

with modifications to the system's BIOS to build a pure software IP Only Server. However, by using specialized hardware the system can be remoted at all times.

IBM's JS20 PowerPC-based blades do not contain any video/mouse/keyboard components. Instead Serial-over-LAN (tunneling text-only serial console messages over UDP datagrams) is used. KVM over IP products (e.g., Cyclades AlterPath KVM/net) provide an easy way to remote all operating environments. However, such products carry a non-negligible price tag, and servers using KVM over IP still require a full complement of hardware components.

## 3 Design

The IP Only Server was designed with several guidelines in mind.

1. The server must run unmodified software, including OS and BIOS.

2. Remote access is needed at all times, from the BIOS boot stage through the OS's lifetime, and even post-OS environments such as the "blue screen of death" or a Linux kernel oops. This does not preclude a more effective remote access method when the OS is operational, such as X-Windows, or Windows Terminal Server.

3. The server should have the minimal amount of local state required for disconnected operation. The hard drives should be remoted over IP, including boot.

4. The IP Only Server must be able to work even when no remote management station is connected, or when one has been connected and then disconnected. Obviously, the remote storage that is necessary for boot and normal operation of the server must be available at all times.

5. Text (console) and graphical mode support must be provided. There is no requirement to provide more than plain VGA mode support — the IP Only Server is not aimed at users who need accelerated graphics.

6. The remote management station should not require a custom or proprietary client, e.g., the KVM-over-IP (Keyboard/Video/Mouse-over-IP) protocol should be based on open standards.

7. A single remote station should be able to control multiple IP Only Servers concurrently.

The IP Only Server can be based on any standard architecture (such as x86). The CPU, memory, northbridge, and BIOS flash are not modified. The server will include at least one network interface. Other peripheral components will not be needed. The functionality of the peripheral components can be emulated by dedicated logic that presents the legacy I/O interfaces to the host (via a PCI bus), and remotes them over an IP based protocol to the remote station. The logic may be implemented as an ASIC or an FPGA depending on the cost/programmability trade-off.

The IP Only Server will not include any local disks. Instead, it boots from a remote boot device, such as an iSCSI disk via iBOOT [1] or PXE. Alternatively, disk access can be remoted like the other legacy I/O protocols. A mixture of the two approaches is possible in principle: the emulation hardware can include an implementation of a boot-capable iSCSI initiator. This will leave the BIOS flash as the only local state.

For the prototype described below we designed an FPGA that presented itself as a VGA/keyboard/mouse device. The server's BIOS and OS accessed the FPGA using their standard drivers. The FPGA received all host accesses as PCI transactions, and handled them appropriately.

We experimented with two different approaches to remoting these PCI transactions to a remote station. The first approach, the Internet PCI Remote Protocol (iPRP) was essentially "PCI over IP": PCI transactions were wrapped by IP packets, sent to the remote station, and processed there. Responses were sent back as IP packets as well and passed them to the local PCI bus. Clearly iPRP does not satisfy design guideline 4 above, and was used mainly as an intermediate debugging tool. It is described in Section 4.1.

The second approach, using the RFB protocol, is described in Section 4.2. In this scheme the emulation FPGA translates keyboard, mouse, and video PCI transactions into the high level RFB protocol that allows using any VNC client (and any OS) on the remote station. PCI transactions are processed locally by the FPGA, while the display and user inputs are handled by the remote station.

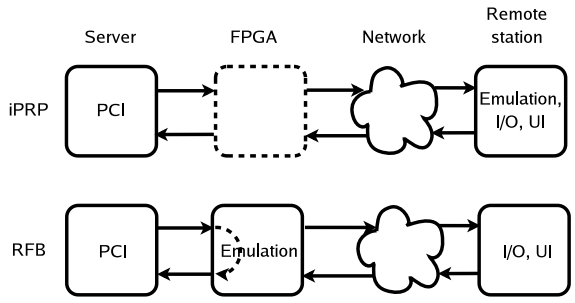


Figure 2: Comparison of iPRP and RFB implementations of an IP Only Server.

The difference between the two schemes is highlighted by Figure 2: with iPRP the FPGA is essentially transparent, and the emulation is done in the remote station; with RFB the remote station exists for the user interface only, the emulation is done in the FPGA, and some PCI transactions (reads) are handled locally inside the FPGA.

Thus, in the RFB design, if the user is not interested in interacting with the server, the server can operate without a remote station. On the other hand, a user can open VNC sessions against multiple IP Only Servers simultaneously. Clearly, this approach supports design guidelines 4 through 7.

# 4 Implementation

For the prototype we used a Dini-group DN3000k10s FPGA evaluation board with a Metanetworks Inc. MP1000TX Ethernet prototyping plug-in board. The FPGA is a Xilinx Virtex-II x2v8000. The design ran at 50 MHz, used 378 KB of local memory, and 8400 logic slices (equivalent to about 1M gates).

The FPGA firmware was divided into four main modules: a PCI Interface Module, a BIOS expansion ROM, a Network Interface Module, and a Transaction Processing Module.

The PCI interface module answers to keyboard controller addresses, VGA controller, VGA memory, and implements an expansion ROM base address register.

The device identifies itself as a PCI-based VGA adapter. Upon discovery of the VGA add-in card, standard BIOSes consider it as the default VGA adapter, and configure the chipset in such a way that all VGA I/O and memory transactions are routed to the device. Having keyboard controller I/O addresses routed to the device is trickier, and requires additional northbridge and I/O bridge configuration done by the BIOS. The expansion ROM is size 32kB, and contains the VGA BIOS routines. It was based on the GPL VGA BIOS included in the Bochs x86 emulator (`http://bochs.sourceforge.net`).

The network interface consists of a Fast Ethernet MAC and a DMA engine. The amount of the management traffic is small, but it may be beneficial to use a separate port for other reasons — for instance, if the main interface is down or saturated (e.g., due to heavy load or to a denial of service attack). The iSCSI storage interface may also be separate for performance and/or security reasons.

In our prototype implementation we used the FPGA's Ethernet interface for KVM emulation, and the server's regular Ethernet interface for the other traffic, including iSCSI.

The Transaction Processing firmware consists of a single loop that gets PCI transactions from the PCI interface, and either handles them locally or wraps them into the appropriate network protocol (iPRP or RFB — see Sections 4.1 and 4.2 below).

To improve network utilization subsequent write transactions are coalesced into bigger network packets, while read transactions are either handled locally (RFB version) or sent to the network immediately (iPRP version). In the iPRP version read responses are forwarded to the PCI Interface Module that handles the PCI read response. The iPRP version also supports remote keyboard generated interrupt requests (IRQ1). Since this interrupt is reserved for the local keyboard controller we used the chipset's Open HCI USB legacy keyboard emulation feature to emulate it.

## 4.1 iPRP — Internet PCI Remote Protocol

The iPRP protocol uses the UDP protocol to emulate memory and I/O space PCI transactions over an IP network. The protocol was designed for ease of implementation, initial debugging and bring up of the hardware and firmware. It is not very efficient nor robust (a network problem or a remote side failure will cause the host to crash or just lock up).

A *command* is defined is a single I/O command such as memory read, I/O space write, or acknowledgment. Multiple commands may be packed into a single Ethernet frame. A *message* is one or more commands, mapped into a single Ethernet PDU. In a multi-command message only the last command may be a read type command, since the PCI-based system can not proceed until the read data arrives back. Each commands has an attached *sequence number* (SN).

The protocol uses the 'go-back-N' scheme, i.e., there may be up to N unacknowledged commands in flight. An ACK message acknowledges all commands with sequence numbers less the the ACK's SN; in case of a read command the ACK also contains the returned data. A message transmit is triggered by a read, a timeout, or in case the maximum message size is reached.

The remote station software was based on the Bochs open source x86 emulator. We extracted the relevant PCI device emulation code and fed it the PCI transactions received over the iPRP payload as input. For host PCI reads, a return packet with the response is sent back to the FPGA.

## 4.2 RFB — Remote Frame Buffer

To overcome the shortcomings of the iPRP version, especially the fact that the server cannot operate without the remote station, we have to emulate the device controllers in the FPGA firmware rather than in the remote station software. As in the iPRP version we based the implementation on code from the Bochs emulator.

To transfer the keyboard, mouse, and video events between the FPGA and the remote station we chose the Remote Framebuffer (RFB) protocol [6]. To this end, we implemented a VNC server in the FPGA firmware. We choose RFB because it is a well known and widely used open protocol with numerous open source clients.

Our FPGA platform was limited in both space (350 KB of memory total) and speed (50 MHz CPU, no memory caches). Accordingly, we started with a straightforward hardware VGA controller emulation in software, based on the Bochs VGA controller, and optimized it both in space and time to fit our FPGA environment. For instance, by removing support for VGA modes that were not used by either Linux or Windows, we managed to reduce the VGA framebuffer to 128 KB.

Since the RFB protocol is TCP/IP based, we also added a TCP/IP stack to the FPGA firmware. Due to the limited memory and processing resources of the FPGA we had to implement a custom embedded TCP/IP stack. The stack is minimalistic and is specifically designed to fit our firmware environment, but it implements a complete TCP state machine and is streamlined: it has a very low memory footprint and avoids copying of data as much as possible.

Like the iPRP version, the RFB-based FPGA firmware is based on a single looping execution thread. The firmware receives the host's PCI transactions from the PCI interface. Host PCI reads are answered immediately, while PCI writes update the local device state machines. Every so often, the frame buffer updates are sent to the remote station to be displayed. The decision of when to update the remote station is crucial for establishing reasonable performance with our constrained FPGA; we developed heuristics that performed fairly well (cf. Section 5).

## 5 Performance Evaluation and Analysis

The most important performance metric for evaluating the IP Only Server is user experience, which is notoriously hard to quantify. In order to approximate the user's experience, we performed several measurements.

All tests were performed on two identical IBM x235 servers including the PCI-based FPGA evaluation board with a 100 Mb/s Ethernet network interface. The remote station software ran on two R40 Thinkpad laptops with a 1.4 GHz Pentium M CPU and 512 MB RAM each. The servers booted either Windows 2003 Server or Red Hat Enterprise Server Linux 3.0. The iSCSI connection was provided through a separate network interface, and the FPGA was not involved in communication with the iSCSI target at all. The performance of iSCSI storage has been studied independently, and we were primarily interested in the performance of our FPGA-based KVM emulation. Therefore, for the purpose of these measurements we used local disks, to achieve a clean experimental environment.

First, we measured the wall time of a server boot for each of the three scenarios: a server with native legacy I/O peripherals, an IP Only Server with an FPGA using iPRP, and an IP Only Server with an FPGA using RFB. In each scenario, we measured the time from power on until a Linux console login prompt appeared, and from power on until a Windows 2003 Server "Welcome to Windows" dialog showed up.

As depicted in Table 1, an unmodified server booted to a Linux login prompt in 238 seconds while a server using the RFB version of the FPGA booted in 330 seconds. This is a 38% slowdown (for the RFB version), which is acceptable for the very first, unoptimized prototype. Additionally an unmodified server booted to the Windows 2003 Server "Welcome to Windows" dialog in 186 seconds and the server with the RFB FPGA booted in 289 seconds. This is slightly worse, a 55% slowdown, but again, it is acceptable.

| Server | Native | iPRP | RFB |
|---|---|---|---|
| Linux | 238 | 343 (144%) | 330 (138%) |
| Windows | 186 | 299 (160%) | 289 (155%) |

Table 1: Linux and Windows boot time in seconds.

It should be noted that the RFB and iPRP versions performed nearly the same. While iPRP sends much more traffic over the network than RFB the bulk of the emulation in the iPRP version is performed by the 1.4 GHz Pentium M CPU of the remote station, whereas in the RFB version it is done by the severely constrained 50 MHz FPGA CPU. This leads us to conclude that there exists a trade off between performance and cost here: by throwing a stronger (more expensive) FPGA or ASIC at the emulation, the performance can be easily improved.

We measured the network utilization using the RFB version of the FPGA in the the same Linux and Windows boot scenarios described above. Table 2 shows the number of actual data bytes exchanged, the number of TCP packets and the throughput for both Linux and Windows boot sequences. The throughput is of the order of 1 Mb/s — not a significant load for a modern local area network, and acceptable even for a wide area connection.

| Booted OS | Unique Bytes | Packets | Throughput |
|---|---|---|---|
| Linux | 52324982 | 189920 | 131488 B/s |
| Windows | 47457592 | 155542 | 164384 B/s |

Table 2: Network Utilization.

An additional observation is related to an important

class of systems that may be built on the basis of IP Only Servers— the so-called "hosted clients". We ran some experiments to assess whether our low level display remoting scheme could be used to support a hosted client solution that deals with graphically intensive applications, and found, similarly to Baratto et al. [3], that remoting the higher level graphics access APIs was more effective than remoting the low level hardware graphic access. This indicates that there is even less reason to keep local graphics adapters on servers supporting hosted clients. For such servers our emulation scheme can be relegated to supporting the pre-OS environment and exception handling (possibly supporting more than one virtual console), i.e., functions that are not graphically intensive.

## 6 Future Work

Future IP Only Server work includes supporting more protocols, such as USB, serial, and parallel. USB support during pre-OS and post-OS stages is particularly desirable, as it would help provide support a diverse set of devices, including floppy and CD-ROM drives.

The IP Only Server prototype runs on x86 only at the moment. A port to other architectures such as PowerPC will validate the design.

The IP Only Server provides interesting opportunities when combined with virtualization technologies such as Xen [2] or VMWare. It can be used to give unmodified guests physical device access while providing requisite isolation and offering each guest its own view of the I/O hardware.

The IP Only Server could also provide a transparent virtualization layer on top of physical devices. It could provide several sets of device control registers associated with different virtual or physical machines. The host OS on each server would access what appears to it as a physical device, but is actually the IP Only hardware. The hardware or the remote station can direct the I/O to different real devices.

## 7 Conclusion

We present a novel approach to legacy I/O support in servers. We designed an IP Only Server, utilizing the IP network as the single I/O bus. All user interaction throughout the lifetime of the server, i.e., during boot, BIOS, OS initialization, normal operation, and post-OS (e.g., "blue screen of death") stages are done from a remote station. No changes were needed for the software running on the host — in particular, neither the BIOS nor the OS were modified. While diskless servers have been attempted before, and headless servers exist as well (e.g., IBM's JS20 blades), as far as we know this is the first attempt to create a diskless, headless server that runs industry standard firmware and software (BIOS, Windows or Linux OS) without any modifications.

We also found wider than expected dependencies on legacy devices, e.g., Windows boot touches keyboard and video hardware more than one would expect. Our emulation approach proved a good way to support Windows without requiring difficult and expensive (and not necessarily feasible) changes in the OS. One of the major reasons for the popularity of the x86 architecture is that it is affordable, and one reason for the low cost of x86 systems is ready availability of software built for large numbers of desktops. The emulation approach is instrumental in keeping this argument valid.

Our research prototype implementation of the IP Only Server provides a user experience that is comparable to that of a regular server, with reasonable latency and low network utilization.

The IP Only Server can provide significant savings in hardware and software costs, power consumption, heat dissipation and ease of management. It eliminates some legacy aspects of the PC architecture, replacing them with a single, simple, and modern counterpart.

## Acknowledgments

The authors would like to thank Kevin Lawton and other Bochs developers for their excellent emulator and Michael Rodeh and Alain Azagury for their interest and support of this project.

## References

[1] iBOOT: Boot over iSCSI. `http://www.haifa.il.ibm.com/projects/storage/iboot/`.

[2] B. DRAGOVIC, K. FRASER, S. HAND, T. HARRIS, A. HO, I. PRATT, A. WARFIELD, P. BARHAM, AND R. NEUGEBAUER. Xen and the Art of Virtualization. In *Proceedings of the 19th ASM Symposium on Operating Systems Principles (SOSP)* (2003), Bolton Landing, NY, pp. 164–177.

[3] R. BARATTO, L. KIM, AND J. NIEH. THINC: a Virtual Display Architecture for Thin-Client Computing. In *Proceedings of the 20th ACM Symposium on Operating systems* (2005), Brighton, United Kingdom, pp. 277–290.

[4] R. MINNICH, J. HENDRICKS, AND D. WEBSTER. The Linux BIOS. In *Proceedings of the 4th Annual Linux Showcase and Conference* (2000), Atlanta, GA.

[5] T. HIROFUCHI, E. KAWAI, K. FUJIKAWA, AND H. SUNAHARA. USB/IP —A Peripheral Bus Extension for Device Sharing over IP Network. In *USENIX Annual Technical Conference, FREENIX Track* (2005), Anaheim, CA.

[6] T. RICHARDSON. *The RFB Protocol*, 2004. Version 3.8.

[7] W. M. FELTER, T. W. KELLER, M. D. KISTLER, C. LEFURGY, K. RAJAMANI, R. RAJAMONY, F. L. RAWSON, B. A. SMITH, AND E. VAN HENSBERGEN. On the Performance and Use of Dense Servers. *IBM Journal of R&D 47*, 5/6 (2003), 671–688.