Appears in 5th WSES/IEEE WORLD MULTICONFERENCE ON Circuits, Systems, Communications & Computers (CSCC 2001)

# DIVISOR: DIstributed VIdeo Server fOr stReaming

Ekaterini M. Gialama[1], Evangelos P. Markatos[1], Julia E. Sevasslidou[2], Dimitrios N. Serpanos[3], Xeni A. Asimakopoulou[1], Evangelos N. Kotsovinos[1]

Institute of Computer Science (ICS)
Foundation for Research & Technology (FORTH)
Vasilika Vouton, Heraclio, Crete, GR-711-10 GREECE
markatos@ics.forth.gr http://www.ics.forth.gr/~markatos

*Abstract:* - This paper presents the design and implementation of a networking system architecture targeted to support high-speed video transmission to multiple clients. We have designed, implemented, and evaluated a high-speed distributed Video Server, which is divided in two different components, the video encoding unit and the network protocol-processing unit. The video-encoding unit performs the video data encoding, while the network protocol-processing unit deals with the network protocol processing. In order to provide a low-cost, scalable system, we have used commercial, off-the-self components. We have implemented our system using a small cluster of personal computers, connected via an optical fiber (raw ATM communication). Our initial experimental evaluation suggests that our Distributed Video Server for Streaming (DIVISOR) can efficiently provide predictable response to a large number of clients, guaranteeing Quality of Service and real-time delivery.

*Key-Words:* - Video Server, Quality of service (QoS), Video on Demand (VoD), network protocol processing, scalability

## 1 Introduction

The phenomenal growth of the Internet over the last decade has dramatically increased the performance requirements for web and multimedia servers. Recent results suggest that the number of Internet users, and the associated traffic they impose, continue to grow exponentially [1]. Furthermore, modern world-wide-web applications (like video streaming and multimedia content delivery) require high Quality of Service (QoS) guarantees. To provide these guarantees, multimedia servers must be predictable, scalable, and efficient.

In this paper, we describe the design and implementation of DIVISOR, a real-time video on demand (VoD) server that provides quality of service guarantees. DIVISOR consists of two major components: the video encoding unit, and the protocol-processing unit. The video-encoding unit deals with the tasks of disk management, data encoding, and data encryption, while the protocol-processing unit executes the necessary network

protocols needed to transfer the data to the end clients. Recent results suggest that the network protocol processing is a significant bottleneck in the performance of typical web servers and routers. As network speed increases at higher rates than processor speed [2], network protocol processing becomes an increasingly larger overhead for media servers. By de-coupling the video encoding from the protocol - processing function, DIVISOR will be able to meet the challenges of high-speed networks and to provide predictable response to larger numbers of users.

The rest of the paper is structured as follows: in Section 2, we give an overview of the system design. We describe the current system implementation in section 3, and in section 4, we present initial experimental

[1] Also with the University of Crete

[2] Currently with GlobEtech Solutions

[3] Also with the University of Patras

results. We place our work in context by contrasting and comparing to related work in section 5. Finally, in section 6, we summarize and conclude the paper.

# 1 System Design

## 2.1 Hardware Architecture

The DIVISOR system architecture consists of two main components, connected via a fast link: the video encoding unit, and the protocol-processing unit.

In a traditional single-processor multimedia server, data management, video encoding, network protocol processing, handling of quality of service mechanisms, as well as final data transmission to multiple clients, are all performed by server's main processor. This fact rises the unpleasant possibility of server's congestion, in the occurrence of multiple requests.

To minimize this possibility, DIVISOR employs a second component, which receives encoded data from the main processor and accomplishes the network protocol processing needed for the their final transmission to end clients. These two components are connected via a fast medium, which enables high-speed, raw data traffic between them.

DIVISOR's hardware architecture design is depicted at Fig. 1.

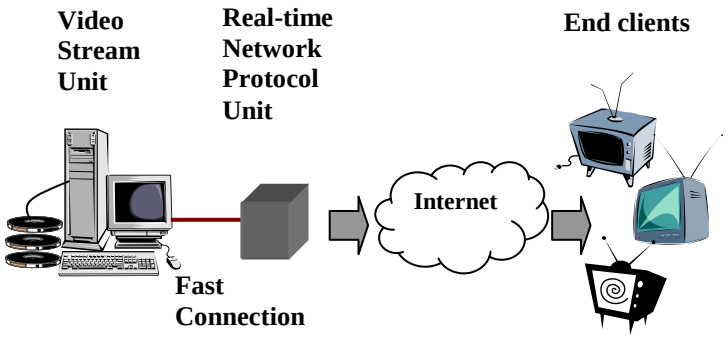



**Fig. 1: Video Server's Hardware Architecture**

## 2.2 Software Architecture

After accomplishing a wide analysis of video streaming applications, we have decided to use a software architecture, which relies on two basic sets of tasks:

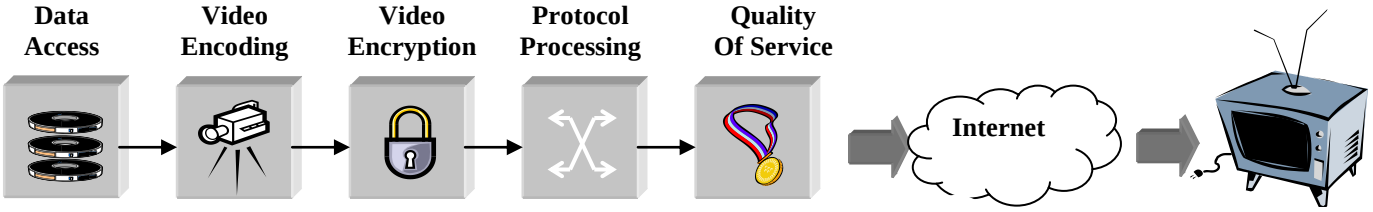



**Fig. 2: Video Server's Software Tasks**

By distributing video server's functionality to separate tasks, running on different components, we expect to outperform a traditional, single-processor video server, who is responsible for both data management and network transmission. Moreover, our decoupled video server will more easily scale to large numbers of clients. Fig. 2 depicts video server's functionality distributed in separate tasks.

### 2.2.1 Network Protocol Processing Cost

Recent studies claim that network protocol processing is a significantly time-consuming operation.

We have evaluated network transmission delay and protocol processing delay, for transmission of various sized UDP packets, using the ttcp benchmark [3]. ttcp is a benchmark that times the transmission and reception of data between two systems using the UDP or TCP protocol. ttcp creates one transmitter and one receiver, and allows data transmission between them, given certain specifications by the user, such as number of packets, packet size, network protocol used (TCP or UDP), and other. After execution, ttcp provides information about user time, system time and actual transmission time. Our test environment consisted of a Sun Enterprise 450 transmitter, and a Sun Enterprise 3500 receiver. A 100 Mbps Fast Ethernet connected the former computers. Fig. 3 depicts the protocol processing delay imposed by transmission of UDP messages vs. network transmission delay, over the 100 Mbps Fast Ethernet network. Fig. 3 shows that there is a significant time overhead generated by the UDP packetizing. In case of medium sized packets (<512 bytes), UDP processing adds time overhead greater than the actual transmission time. In case of large size packets, UDP adds a 25% - 30% delay overhead.

Recent publications suggest that today's Internet average packet size does not exceed 300 bytes [4]. So, we may conclude that in most cases, network

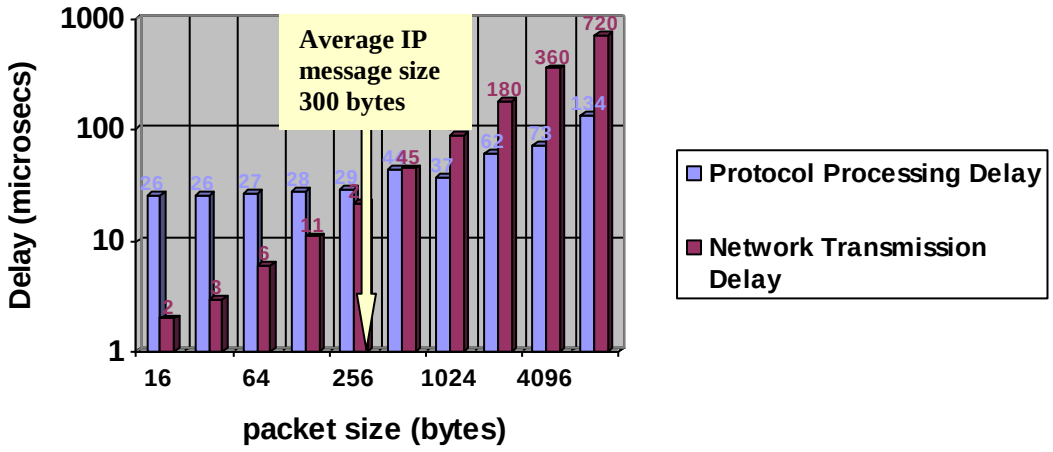



**Fig. 3: Network transmission delay vs. UDP protocol processing delay**

protocol processing imposes an undue time overhead on transmission time.

Based on the previous results, we can extrapolate the following conclusion: During the transmission of 8 Kbytes packet, CPU appeared busy at 18%. This CPU usage percentage suggests that the protocol processing bounds bandwidth up to 555 Mbps (100% / 18%*100). Therefore, in order to use today's high-speed networks, such as Gigabit Ethernet, on the current infrastructure, the use of a dedicated component for network protocols processing becomes a unique solution.

Several studies, which had been formed in order to evaluate network protocol processing overhead, present similar results. Wolman et al. [5] report that the cost of TCP/IP processing needed to transmit a 4 Kbytes message is 671 microsecs, while for the 8 Kbytes, protocol processing delay reaches 1262 microsecs. However, transmission of the previous sized buffer of 8 Kbytes, over a high-speed network, such 155 Mbps ATM, does not cost more than 0.5 msec.

Our system architecture tries to minimize this undue time overhead, generated by network protocol processing.

# 3 System Implementation

## 3.1 Hardware Architecture

To provide an easy-to-use, low-cost, and scalable system, we have decided to use commercial, off-the-shelf products. A high performance high-end PC, such as an Intel Pentium III, seems to be an ideal choice

for the video encoding unit. We have used a wide spread operating system on it, such as Linux, which is open-source and stable.

The choice for the network protocol processing unit appears more difficult. We need appropriate, specialized hardware, which employs fast network processing techniques, and which enables its fast connection with the main processor. Furthermore, in order to provide QoS and scheduling of processes depending on time priorities, the design for the second component is based on a real-time operating system (RTOS).

The most widely used hardware for specialized issues are embedded boards. An embedded board, may turn out to be a good choice for our system's protocol processing unit, as it can provide fast connection to the main processor, via PCI bus, and it can be used with a real-time operating systems [6]. However, embedded boards need to be redesigned and re manufactured every 2-3 years, to keep up with the increasing performance needs. Furthermore, since they are sold in small numbers, they usually have disproportionally high costs. On the contrary, off-the-self components are usually inexpensive (due to mass production).

Furthermore they are upgraded every few months (if not every few weeks).For example, better and faster Pentium-based PCs appear every few months in the market. Therefore, instead of investing into an expensive, inflexible, and difficult to upgrade embedded board, we decided to capitalize on a state-of-the-art, low-cost, and high-performance real-time commodity such as Intel Pentium III, connected back-to-back to the main processor, via an optical fiber, for fast data transmission (raw ATM).

The market analysis for an appropriate real-time operating system lead us to FSMLabs RTLinux [7], as it is open source, widely used, cost-free, and provides programmers with an easy-to-use API for implementation of real-time modules and periodic tasks.

Fig. 4 presents DIVISOR system architecture.

## 3.2 Software Architecture

DIVISOR consists of **two sets of processes**: the video encoding processes and the protocol execution processes.

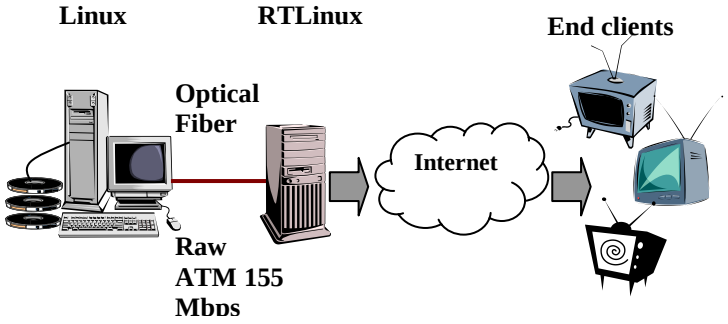
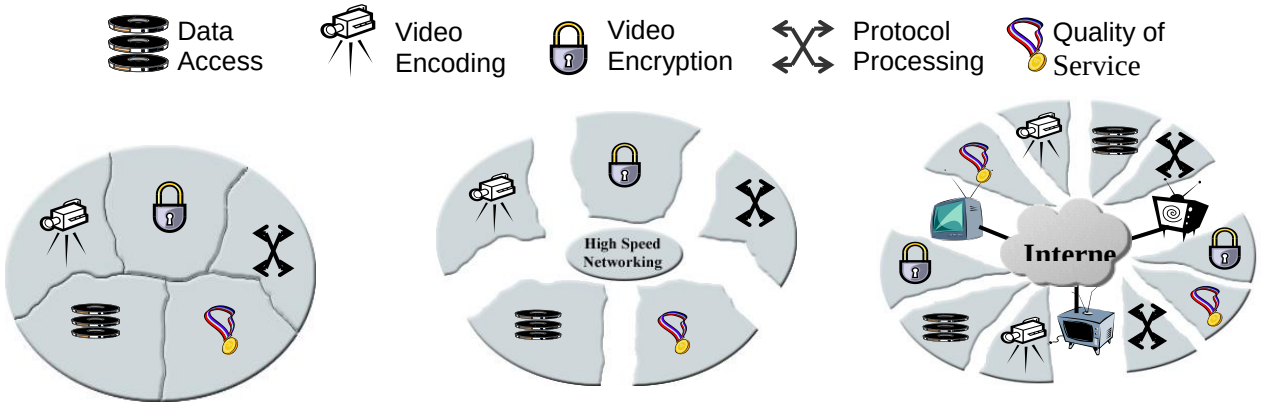



**Fig. 4: DIVISOR hardware architecture**

The video encoding processes, which run on the video stream-processing unit, deal with secondary storage management, and data encoding. Based on information retrieved from video (which is usually a mpeg file [8]), video-encoding processes calculate the appropriate bit rate. Then, video is split to packets, which are sent to the protocol processor, at video's bit rate.

Although currently our system provides its clients with data of the best possible quality (all frames, appropriate bit rate) we plan to extend it to a more realistic model, where clients will be served with different versions of data (various numbers of frames), according to their needs and financial state. Our video stream-processing unit is able to support this data filtering extra functionality, as well as data encryption, for security reasons.

The protocol execution processes, which run on the network protocol-processing unit, undertake network protocol processing of data received from the video-encoding unit. These processes are also responsible for data's final delivery to end clients, with quality of service guarantees.




**a) Monolithic approach to video server's architecture** **b) DIVISOR - our approach** **c) Our vision for the future**

**Fig. 5: traditional video server's software architecture vs. DIVISOR**

Our current implementation is based on RTLinux. We have implemented a real-time module, which manages the execution of a periodic thread with real-time constraints. This periodic thread receives data from a raw ATM socket, performs network protocol processing and transmits the requested video to clients, using Quality Of Service protocols such as RTP (real-time protocol) and RSVP (resource reservation protocol). Fig. 5 depicts the DIVISOR architecture, compared to traditional monolithic video server's architecture. A monolithic video server, (Fig. 5 (a)), forms a single point of failure, may be congested in appearance of multiple requests, and cannot scale to large numbers of clients. Our approach, (Fig. 5 (b)), enables the use of a grid of video servers, each one responsible for a separate task. Finally, our approach can be easily extended to a video server's grid, distributed all over the Internet.

## 4 Performance Evaluation

In order to provide our system with predictability and meet real-time constraints, we decided to use a real-time operating system such us RTLinux. To evaluate RTLinux's scheduling efficiency, we have measured multimedia data interarrival times shown at client's side, when the video server was running on Linux and on RTLinux. The obtained results are shown in Table 1.

| | **Linux** | **RTLinux** |
|---|---|---|
| **Variance** of Int. Time | 27642.39 | 2536.40 |
| **Average** Int. Time (msecs) | 94.49 | 18.14 |

**Table 1: scheduling efficiency comparison between Linux and RTLinux**

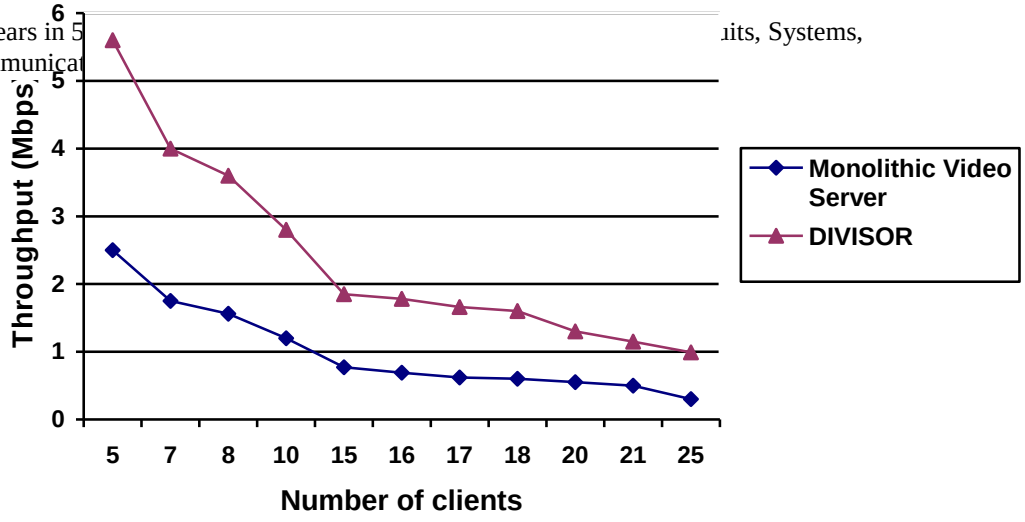



**Fig. 6: Performance evaluation of a monolithic video server and DIVISOR**

We can see that RTLinux's usage results in a significant reduction of data interarrival times. The average value of interarrival times in RTLinux is five times smaller than the one in case of Linux usage. Furthermore, the variance of interarrival times with RTLinux is an order of magnitude lower.

To evaluate DIVISOR's performance, two experiments have been undertaken. In the first experiment, we have measured a monolithic video server's throughput, responsible for data access, video encoding, on-the-fly video encryption using the DES (Data Encryption Standard) encryption algorithm [9], as well as protocol processing needed for data's final delivery to clients. In the second experiment, DIVISOR's throughput was measured. DIVISOR's video encoding unit was handling data access, video encoding and video encryption, using DES, while DIVISOR's protocol processing unit was responsible for a part of DES algorithm execution (for better load balancing) and network protocols processing, for data's final delivery to end clients.

The obtained results are shown in Fig. 6. DIVISOR achieves a speedup of more than 2 and scales better to large numbers of clients. In case of a client request for 1 Mbps bitrate video, the monolithic video server cannot serve efficiently more than 10 clients, while DIVISOR is able to handle with 25 simultaneous client requests.

The above measurements suggest that by decoupling a video server's functionality to separate tasks, running on different components, DIVISOR outperforms a monolithic server and presents great scalability. Furthermore, DIVISOR's approach may be easily extended to the distribution of its functionality over a video server's grid, which could achieve an even better performance.

# 5 Related Work

There has been significant research targeted to improve multimedia servers' performance.

The main focus of this research has been on the video storage servers and how to optimize the use of disk and memory. Some researchers have developed innovative methods for optimizing the use of stream buffers in the system memory of a video server [10]. Others have proposed increasing the block size in the file system to improve performance [11]. Novel techniques have been suggested for placement of data on disks [12][13].

To provide the best possible performance, commercial video servers tend to rely on super-computer backplanes or massively parallel memory and I/0 systems in order to provide the needed bandwidth [14][15].

However, besides performance, a video sever must be able to predictably achieve (soft) real-time constraints. To provide predictability, several researches have studied network jitter presented at video delivery across a network [16].

To provide high performance at a reasonable cost, clusters of workstations, interconnected by high-speed network, form a common architecture for video servers [17], [18], [13].

All the above approaches differ from DIVISOR, as they do not attempt to eliminate network protocol-processing overhead, measured in section 2.1.

Orfanos et al. [19] have implemented a protocol accelerator embedded system, which processes the TCP/IP headers and accomplishes checksum computations, while data is transferred from server's memory directly to the network. Contrary to embedded-based solutions, DIVISOR proposes a commodity approach that is able to ride the performance wave of the technology.

# 6 Conclusion

In this paper we present the design, implementation, and evaluation of DIVISOR, a distributed, video-on-demand server, with quality of service guarantees.

As shown in section 2.1, network protocol processing imposes an undue time overhead on transmission time. To minimize this overhead, DIVISOR hardware architecture decomposes the functionality of a monolithic server into several processing units, which undertake data access, data encoding and encryption, as well as network protocol processing needed for video's final transmission to end clients. By distributing a video server's functionality to separate tasks, running on different components, DIVISOR outperforms a traditional, single-processor video server, who is responsible for both data management and network transmission.

# 7 Acknowledgements

This work was supported in part by Research and Technology Business Project (EPET II) - Researcher's Assistance Program 99 - 193 (PENED 99-193), funded by the Greek Secretariat for Research and Technology. We deeply acknowledge this financial support. Anthony G. Danalis contributed several useful ideas and drawings.

*References*

[1] Brewster Kahle, Archieving the Internet, Scientific American, March 1997.
http://www.archive.org/sciam_article.html

[2] D. Forman, Gilder's Law stretches new economy's rubber bandwidth, The Australian, November 99, pp.36.
http://www.csee.uq.edu.au/~cs270/Articles/Gilder.13.11.99.htm

[3] Test TCP (ttcp) benchmark
http://www.ccci.com/product/network_mon/tnm31/ttcp.htm

[4] M. Katevenis et al., Wormhole IP over (connectionless) ATM, ICS-FORTH, Greece, July 1998
http://archvlsi/wormholeIP/arch98/wIP_98.html

[5] A. Wolman et al., Latency Analysis of TCP on an ATM network, Proceedings of the USENIX Winter '94 Technical Conference, San Francisco, CA, January 1994, pp. 167-179.

[6] Cyclone Embedded Boards
http://www.cyclone.com

[7] FSMLabs: The RTLinux Company
http://www.fsmlabs.com

[8] International organization for standardization:
ISO/IEC JTC1/SC29/WG11: Coding of Moving Pictures and audio - MPEG1
http://www.cselt.it/mpeg/standards/mpeg-1/mpeg-1.htm

[9] DES - Data Encryption Standard
http://www.itl.nist.gov/fipspubs/fip46-2.htm

[10] A. Dan, D. Sitaram, Buffer Management Policy for an On-Demand Video Server, Research Report RC-19374, IBM Thomas J. Watson Research Center, York Town Heights, NY, 1994.

[11] R. Haskin, The Shark Continuous Media File Sever, Proceedings of IEEE COMPCON '93, San Francisco, 1993, pp. 12-15.

[12] H. M. Vin, P. V. Rangan, Designing a Multi-user HDTV Storage Server, IEEE J. Selected Areas in Commun. 11, No. 1, 1993, pp. 153-164.

[13] W. J. Bolosky, R. P. Fitzgerald, J. R. Douceur, Distributed Schedule Management in the Tiger Video Fileserver, Proceedings of 16th ACM - Symposium on Operating Systems Principles (SOSP), St.-Malo, France, Oct. 1997, pp. 212-223.

[14] M. Nelson, M. Linton, S. Owicki, A Highly Available, Scalable ITV System, Proceedings of 15th ACM - Symposium on Operating Systems Principles (SOSP), December 1995, pp. 54-67.

[15] A. Laursen, J. Olkin, M. Porter, Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data, SIGMOD record (ACM-Special Interest Group on Management of Data ), June 1994, pp. 470-477.

[16] S. L. Veira, N. R. Gomes, A. Texeira, A distributed mpeg player with jitter control, Proceedings of XVI International Conference of the Chilean Computer Science Society, Valdivia, Chile, November 1996, pp. 155-164.

[17] O. Sandsta, S. Langorgen, R. Midtstraum, Video Server on an ATM Connected Cluster of Workstations, Proceedings of XVII International Conference of the Chilean Computer Science Society, Valparaiso, Chile, November 1997, pp. 207-217.

[18] A. Bonhomme, P. Geoffray, High Performance Video Server using Myrinet, Myrinet User Group Conference, Lyon, France, September 2000, pp. 66–72.

[19] G. Orphanos et al., Compensating the Moderate Effective Throughput at the Desktop, IEEE Communications Magazine, April 2000, pp. 128 - 135.