Network Processors as Building Blocks in Overlay Networks

Ada Gavrilovska, Karsten Schwan, Ola Nordstrom, Hailemelekot Seifu
Center for Experimental Research in Computer Systems (CERCS)
Georgia Institute of Technology
Atlanta, Georgia, 30332
{*ada, schwan, nalo, seif*}*@cc.gatech.edu*

Abstract

This paper proposes an architecture that permits selected application- and middleware-level functionality to be 'pushed into' network processors. Such functionality is represented as stream handlers *that run on the network processors (NPs) attached to the host nodes participating in overlay networks. When using stream handlers, application- and middleware-level functionality is 'split' into multiple components that are jointly executed by the host and the attached NP (ANP). Resulting improvements in application performance are due to the network-near nature of ANPs and due to the ability to dynamically customize stream handlers to meet current application needs or match current network resources. The evaluation of our current prototype implementation indicates that the use of ANP-level handlers can reduce the delays on the application data path by more than 25%, and can sustain higher throughput for the application services provided by stream handlers. In addition, stream handlers are a suitable basis for scalable implementations of data-increasing services like destination-customized multicast.*

1. Attached Network Processors in Overlay Networks

Research in active networking has long argued the benefits of dynamically reconfiguring the network infrastructure, to better match application needs with platform conditions [11, 16, 15]. Technical advances due to such work include the development of architectures and runtime methods for programmable routers and switches, the provision of general techniques for adapting communications, and experimentation with applications like video distribution and service proxies. While building on such advances, our research focuses on the network's 'edges', where we are investigating the benefits attained for applications by dynamically customizing the functionality of 'attached network processors' (ANPs). ANPs are the programmable network devices that are attached to the end hosts used by applications. The specific ANPs used in our research are the emerging class of programmable network processors, which are becoming an attractive target for deploying new functionality 'into' the network, at lower costs and with shorter development times than custom-designed ASICs. Their utility has already been demonstrated for programming network-centric services like software routing, network monitoring, intrusion detection, load-balancing, service differentiation, and others. In contrast, we focus on application-level functionality. For example, data filtering and packet scheduling benefits applications like remote sensing, remote visualization, online collaboration, and multimedia [17, 7]. Dynamic synchronization methods or content-based multicast benefit high performance simulations or transactional applications like operational information systems [5, 10]. In all such cases, application-level performance is directly affected by the ability to 'push' suitable functionality onto ANPs, and to dynamically customize it to meet current application needs or match current network resources.

Our past work has focused on using ANPs in system area networks with high end cluster machines [10]. Our current work is broader, in that it targets any distributed application that uses overlay networks. These include the online collaboration and distributed scientific codes considered in our earlier work, transactional applications, and the middleware-based grid services that are becoming increasingly important for future distributed systems. The specific middleware considered in our research implements publish/subscribe event-based communications across cooperating peers [3, 14].

The key idea presented in this paper is to permit applications and middleware to run overlay services on the 'edge-deployed' NPs that are attached to participating hosts. This is achieved by 'splitting' the middleware- or application-level functionality executed on hosts (in their OS kernels or at application-level) into multiple components that are efficiently and jointly executed by the host and ANP. This is shown in Figure 2, where this additional functionality is placed into the region labeled *Access*, which is split across ANP and host. Splitting should be dynamic, (i.e., when overlays are first constructed), and split components should be runtime-configurable, so as to continuously match their operation to current application needs and network resources (e.g., by configuring certain execution parameters, such as rate or point of invocation, or by configuring application-level parameters, such as filtering thresholds or bounding box coordinates).

The goal is to use ANP resources to deliver to a wide range of scientific and commercial applications improvements in performance or reliability. Benefits experienced by applications are due to customized communications and the reduced computational loads on end-host, e.g., by offloading overlay computations onto ANPs. We avoid some of the safety and security issues raised in the context of active networks by engaging end host operating systems in the control of ANP-level functionality.

The experimental results presented in this paper use Intel's' IXP1200 network processors as ANPs. Measurements show that use of ANPs vs. hosts for certain application-level services can reduce the latency of the application data path by more than 25%, and that it can sustain high throughput for most of the services considered. It can also provide a 25% improvement for certain data-increasing services, such as destination-customized multicast.

Remainder of paper. In the remainder of paper, we first describe the application domains and platforms targeted by our work. Section 3 briefly describes the ANP-resident portion of an application-specific service, termed stream handlers, then present the system components of the composite host-ANP nodes. Section 4 presents and discusses experimental results attained with our prototype implementation. Related work, conclusions, and future work are presented last.

2. Application Domains and Execution Platforms

Applications. Our focus is on streaming applications that involve the continuous exchange of large volumes of data, exemplified by operational information systems (OIS), real-time collaboration, dynamic web applications delivering camera- or sensor-captured data to remote users, etc. Previous work has demonstrated the utility of overlay networks for such data-intensive applications, in terms of improvements in performance and resource utilization [14, 3, 19]. Improvements are attained in part by customizing the data being exchanged based on the destinations' interests, using communication paradigms that offer customization control like peer-to-peer and publish/subscribe.

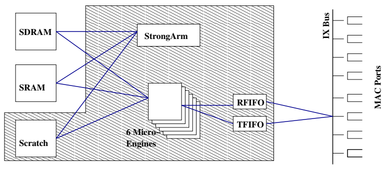

Figure 1. Intel IXP1200 Block Diagram.

Two representative applications are the operational information system used by companies like Delta Air Lines (OIS) [5] that manipulates business events and a real-time scientific collaboration, termed SmartPointer [17], which transforms and filters molecular data. In the OIS, in order to meet requirements like reliability and availability, the streaming of business events is customized with methods like selective data mirroring. In SmartPointer, data is customized via downsampling or reduction methods in order to continuously meet the real-time needs of end users despite varying network resources.

For these applications, we advocate the use of ANPs as building blocks for the dynamic and customized delivery of data across overlay networks. Host nodes perform computationally intensive data transformations or apply business rules that require substantial state, while ANPs perform application-specific stream management functions like client-specific data filtering, selective dynamic data mirroring, and transactional constructs.

Intel IXP1200 Network Processor. Our implementation uses Intel's IXP1200 [6] as an ANP. This programmable network processor is a multiprocessor on a chip containing a StrongArm (SA) core and 6 microengines with 4 thread contexts each (see Figure 1). For the Radisys boards used in our work, each microengine operates at 232MHz, and it has a 2Kword instruction store, shared among the 4 contexts. The chip integrates memory controllers for 256MB of SDRAM, 8MB of external SRAM and 4KB of internal scratch memory. Four 100T MACs connect externally through a proprietary 64-bit/66-MHz IX bus accessed through on-chip transmit and receive FIFOs, and network packets are received as a sequence of 64-byte MAC-layer packets. We expect to migrate to the next generation IXP with two gigabit Ethernet MACs in the near future.

3. System Overview

3.1 Stream Handlers

Stream handlers are lightweight, composable, parameterizable computational units that encapsulate the application- or middleware-level functionality to be applied to data streams in the network processor [4].

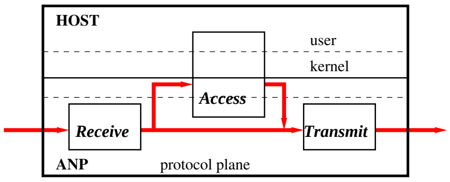

Figure 2. Data Path.

Accessing application-level data. Since stream handlers execute application-level functions, their implementation utilizes (1) protocol code that assembles application-level data units, and (2) the ability to interpret the structure of the data byte stream and access the relevant packet body contents. For the current IXP platform, we rely on an RUDP-like protocol to efficiently implement the reassembly and fragmentation of application-level data in IXP memory. This is because previous work has shown that TCP cannot be run at link speeds on this platform [13]. For next generation IXPs, we expect to use standardized protocols. Application-level data is described with compact binary data format descriptions [3], which provide to handlers information about the structure and layout of the data they manipulate. The use of formats enables us to duplicate for packet bodies the elements that make it easy for NPs to perform header-based operations: known header formats, offsets, and types and sizes of fields. IXP-resident handlers rely on such encodings to access the correct parts of the MAC-layer packet(s), or the memory buffers on which they operate.

Basic data path components. In the IXP1200, packets are delivered from and to the network as a sequence of 64B MAC-layer packets, on any one of its ports. Application-level messages are assembled from these packets in IXP memory, and access to these buffers is enabled through a set of shared queues. Figure 2 represents the basic elements of the data path through the host-ANP. The *Receive* block executes the receive-side protocol code to detect message and flow boundaries across packets, and associates packets with the corresponding memory queues. In the IXP1200, this functionality is executed by a set of receive (Rx) threads, currently allocated on a per port basis. Received data can be forwarded directly to the transmit-side protocol code – *Transmit*, implemented by a separate Tx-thread for each port. Any additional processing is represented with the *Access* block. The stream handlers represent, in fact, such additional functions, and can be executed by one of the ANP-level processing contexts, or at the host kernel- or user-level.

Programming interface. All handlers have access to certain global information, which includes stream state, current data queues, counters, control buffers, etc. A set of IXP registers and routines can be used to move data across the DRAM data queues, R/TFIFOs, and the network. A portion of IXP memory is made available to handlers to store state and parameter values. Unlike traditional application-level handlers, which operate solely on application data on top of the underlying protocols, on the ANP, stream handlers are essentially combined with the protocol's execution. Therefore, the protocols used need to be taken into consideration, and handlers are aware of both data and header offsets, and of data delivery to and from the underlying network.

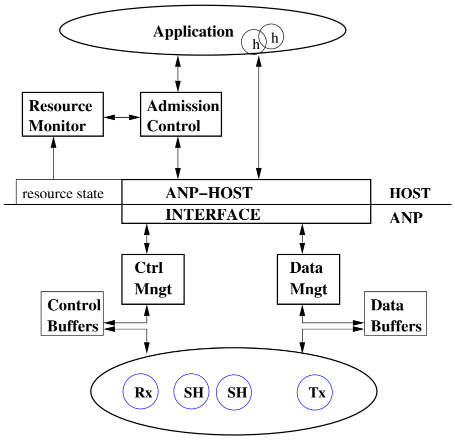

Figure 3. System Components.

3.2 System Components

Figure 3 depicts the system components of the composite host-ANP nodes. On the ANP side, multiple microengines execute the receive and transmit side protocol, and/or implement the stream handler functionality assigned to them. Controlled access to data buffers is used to coordinate the ANP's execution contexts, and to exchange data with the host-resident application component. The configuration and deployment of new handlers is managed by the admission control block, which depending on resource availability, configures the IXP runtime through accesses to its control buffers.

Stream handler deployment. A rich set of application-level services can be implemented by deploying stream handlers to all parts of the data path, at any of its execution engines. We have previously shown that on the ANP, stream handlers can be applied to data as part of the (1) Rx- or (2) Tx-side protocol processing, or (3) while data is 'in transit' in ANP memory [4]. In addition, data can be processed by kernel- or user-level handlers on the host, which is important for complex handlers or to deal with runtime resource limitations. Stream handlers can be simultaneously deployed at multiple points of the data path, and executed by any of the execution engines. In this manner, compositions of handlers are achieved, which enables us to provide complex, composite services to applications (e.g., representing application-level workflow). For instance, experimental results presented in this paper demonstrate performance gains of over 25% for message sizes of 50kB derived from implementing destination-customized multicast as a composition of an Rx-side mirroring handler and a set of Tx-side handlers that perform the customization based on the destination, compared to an implementation that uses a memory-resident handler. A more complex example is one that performs filtering or stream differentiation on the ANP, and directs the customized substream to host-resident handlers, to execute computationally expensive data transformations, or to perform value-added services like intrusion detection.

Application-ANP interaction. The deployment and configuration of stream handlers is driven and controlled by the host to which the NP is attached. Our current implementation of ANP-resident stream handling cannot dynamically map newly created handlers onto an IXP. Instead, we have implemented a more modest approach of exchanging control information between the application and the ANP in order to allow the application to choose between a set of pre-existing handlers on the IXP and/or to dynamically pass new parameter values to selected handlers. This allows us to adjust ANP actions to runtime changes in application needs and operating conditions. For instance, as a client's interests shift to different data in a remote scientific visualization, parameters identify the new coordinates of appropriate image regions to be selected for forwarding to the client from a stream of images being delivered to it. Or, as request loads increase on a server system, the IXP can downsample the incoming data streams, as permitted by the application.

The interaction between the ANP and the application host node involves the exchange of a set of control messages, similarly to those described in [10]. For instance, a control message can request an existing connection to be terminated, or it can contain parameters, one example being an identifier of the stream handler to be associated with a certain data stream. A portion of the IXP memory is designated for enabling such control communications. Control messages can also be mixed in with regular data traffic, and delivered to the ANP on one of its incoming ports. We are currently implementing the host-ANP control channel over the PCI-based interface described in [9]. The control interface will use OS-controlled mappings between host and IXP memory in order to configure the ANP and monitor its state.

Deploying stream handlers – admission control. For reasons of programming complexity and system safety and security, end users cannot directly map stream handlers into ANPs. Instead, end users specify suitable stream handlers, but these handlers are 'deployed' by the underlying admission control unit. Resource management in the host operating system tracks the available 'headroom' on the ANP, and it manages the different stream handlers available and executable on ANPs. Our current model is to simply compute

data size	Host–side	IXP–side
100B	36us	34us
1kB	83us	82us
1.5kB	132us	132us
10kB	896us	854us
50kB	6.8ms	5.6ms
100kB	16.4ms	11.9ms

Figure 4. Latency on source-destination path when application level routing is performed at the host vs. at the IXP side.

the total number of cycles that can be allocated to a single packet's processing (i.e., headroom) and compare this number with the number of cycles consumed by the stream handler being deployed. Such comparisons can be easily implemented with the existing IXP programming tools [6, 9]. In ANP-level resources are not available, handlers are executed on the host, with a corresponding performance cost.

Dynamic reconfiguration. In order to best utilize resources and match current application needs and platform resources, stream handlers need to be dynamically deployed and configured. Handler selection and parameterization both rely on runtime resource monitoring, admission control, and resource management mechanisms.

4. Experimental Evaluation

Evaluation is performed on the aforementioned IXP1200 Radisys boards, interconnected via 100Mbps Ethernet links, using a cluster of Dell 530s with dual 1.7GHz Xeon processors running Linux 2.4.18. We also use the IXP1200 simulation package SDK2.0, claimed to be cycle-accurate to within 2%. Stream data consists of (1) sequences of images files, and (2) replays of flight data originally captured at Delta Airlines. All data passes through an intermediate node (host or ANP), where a handler is applied to it. The application components executed on the cluster nodes use our version of the RUDP protocol built on top of raw sockets. The same protocol is used on the IXP microengines.

The first experiment compares the latency on the source-destination data path when performing application-level routing at an intermediate host vs. an IXP-based ANP. The results are gathered by timestamping the data at the source and destination, and the average is reported over multiple runs. From Figure 4 we observe that while for smaller data sizes, application-level routing can be performed equally efficiently on the IXP as on the host, as message sizes increase, IXP-side data routing can reduce delays on the data

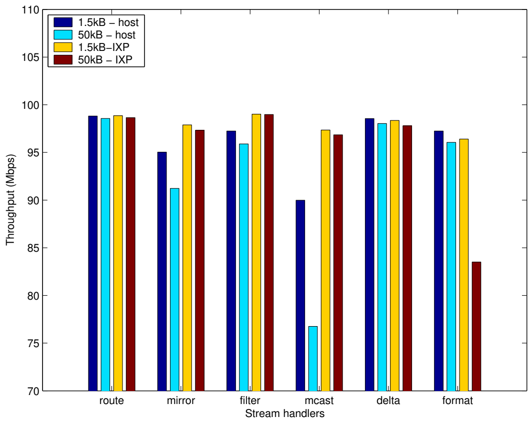

Figure 5. Data throughput at the intermediate node - the host vs. the IXP for different types of stream handlers.

path by more than 25%. This reduction is due to the IXP's built-in message handling concurrency, and due to the reduced costs of protocol processing, buffer copying and host kernel invocation. We expect that further improvements in the IXP runtime implementation will result in additional reductions in the data delay through the NP.

The next set of experiments compares the host's vs. the IXP's ability to sustain throughput, while performing a variety of additional application-level functions on the data path. Results are gathered by timing the transmission duration for $3,000$-$10,000$ application-level messages, depending on their sizes. The stream handlers implementing content-based routing, mirroring and filtering are executed on the Rx-side in the IXP. The destination-specific multicast is implemented in the IXP by composing an Rx-side mirroring handler with a Tx-side filtering handler. The 'delta' handler in Figure 5 modifies 20% of the application-level message, and the 'format' handler operates on the entire payload. These handlers, as well as all the host-side handlers are executed once the application-level message has been assembled in memory.

Figure 5 represents the throughput with which the intermediate node, host or IXP, can operate on the data stream, and deliver the corresponding service to the application. For the most part, the IXP performs at least slightly better than the host-side implementation of the same service. Some of the results support our previous simulation-based conclusions, e.g., the importance of filtering application-level data as early as possible ('filter'). Particularly interesting is that the ANP implementation of data-increasing handlers, such as mirroring, have a much improved ability to sustain higher throughput. This is because it is more efficient to implement the multiple packet transmissions at the network level (e.g., to perform multiple port accesses per each packet), than to execute the host-side protocol stack for multiple connections. Furthermore, while IXP-side mirroring is just slightly affected by the message size, a 7% drop in the sustained throughput is observed when the host mirrors 50kB data to two destinations. The results becomes more dramatic if the mirrored data is to be customized based on the destination: for 50kB messages and two clients the IXP outperforms the host by 25%. This difference is even more profound as the data sizes and number of clients increase.

For computationally intensive handlers (e.g. format translations, security checks, etc), host-side implementations are more efficient. As we increase both the message size and the computational requirements, we eventually exhaust the IXP's resources. At the same time, the host CPU is orders of magnitude faster than current generation IXP microengines. This supports our desire to 'split' handlers across host and ANP, depending on resource availability and application requirements.

Additional results not included for brevity demonstrate the scalability of the ANP handlers, and their effects on host CPU utilization.

5. Related Work

Network-level services realized on NPs include software routing, firewalling, low-level intrusion detection, packet classification and scheduling, network monitoring, software multicast, and service differentiation across different traffic classes [13, 2, 20, 15]. Application-level services demonstrated on NPs include load-balancing for cluster servers, the proxy forwarding functionality servers need, and payload caching [1, 18]. With our approach, these and many other services can be implemented on ANPs or jointly by ANPs and hosts, independent of the specific protocol- or application-level headers being used.

As with research in active networking, our goal is to extend and customize the behavior of the network infrastructure, in order to meet application needs [16, 11, 15]. We differ by focusing on the 'edges' of the network, and by operating in the controlled environment consisting of host and attached NP. The customized delivery of streaming data, which results from the execution of these handlers, has already been widely used in industry and academia, in a variety of peer-to-peer or publish-subscribe systems [14, 3, 19].

Handler compositions are similar to those performed in modular frameworks for dynamically generating customized higher-level services from simpler components [12, 8]. Unique to our approach is the deployment of these components (i.e., of stream handlers), across the host - NP boundary, dependent on service requirements and on available system and network resources.

6. Conclusions and Future Work

The paper presents a software architecture and its prototype implementation that enables the movement of application- and middleware-level functionality ‘into’ the network, ‘onto’ network processors attached to nodes (ANPs) that run certain overlay components of distributed, data-intensive applications. The ‘stream handlers’ executed by the ANP or host can be dynamically deployed and configured to match current application needs or operating conditions, and also taking into account current platform resources. Performance gains are due to the network-near execution of stream handlers and the flexibility with which stream handling can be mapped across the ANP-host boundary.

Future work will focus on the composition of complex services from stream handlers executed at different points in the data path. The goal is to allow applications to embed critical functionality into the fast path and create new kinds of services for distributed commercial and scientific codes.

Acknowledgments. Ivan Ganev and Kenneth Mackenzie provided the initial implementation of the Georgia Tech IXP1200 driver for the Radisys ENP2505 placed in a PC running Linux, and the PCI-based host-IXP interface, on top of which we are implementing the stream handlers architecture and the composite host-ANP nodes. The authors would also like to thank Austen McDonald for his helpful suggestions during the implementation phase.

References

[1] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha. Design, Implementation and Performance of a Content-Based Switch. In *Proc. of IEEE INFOCOM 2000*, volume 3, pages 1117–1126, Tel Aviv, Israel, Mar. 2000.

[2] A. T. Campbell, S. Chou, M. E. Kounavis, V. D. Stachtos, and J. B. Vicente. NetBind: A Binding Tool for Constructing Data Paths in Network Processor-based Routers. In *Proc. of IEEE OPENARCH’ 02*, New York City, NY, June 2002.

[3] G. Eisenhauer, F. Bustamante, and K. Schwan. Event Services for High Performance Computing. In *Proc. of HPDC-9*, Pittsburg, PA, Aug. 2000.

[4] A. Gavrilovska, K. Mackenzie, K. Schwan, and A. McDonald. Stream Handlers: Application-specific Message Services on Attached Network Processors. In *Proc. of Hot Interconnects 10*, Stanford, CA, Aug. 2002.

[5] A. Gavrilovska, K. Schwan, and V. Oleson. Practical Approach for Zero Downtime in an Operational Information System. In *Proc. of ICDCS’02*, Vienna, Austria, July 2002.

[6] Intel IXP1200 NP Family. http://developer.intel.com /design/network/producs/npfamily/ixp1200.htm.

[7] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West. Architecture and Hardware for Scheduling Gigabit Packet Streams. In *Proc. of Hot Interconnects 10*, Stanford, CA, Aug. 2002.

[8] X. liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable. Building reliable, high-performance communication systems from components. In *Proc. of 17th ACM Symposium on Operating Systems Principles (SOSP’99)*, Kiawah Island, SC, Dec. 1999.

[9] K. Mackenzie, W. Shi, A. McDonald, and I. Ganev. An Intel IXP1200-based Network Interface. In *Proceedings of the Workshop on Novel Uses of System Area Networks at HPCA (SAN-2 2003)*, Anaheim, CA, Feb. 2003.

[10] M.-C. Rosu, K. Schwan, and R. Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The Virtual Communication Machine-based Architecture. *Cluster Computing,*, May 1998.

[11] M. Sanders, M. Keaton, S. Bhattacharjee, K. Calvert, S. Zabele, and E. Zegura. Active Reliable Multicast on CANEs: A Case Study. In *Proc. of IEEE OpenArch 2001*, Anchorage, Alaska, Apr. 2001.

[12] J. Smith, I. Hadzic, and W. Marcus. ACTIVE Interconnects: Let’s Have Some Guts. In *Proc. of Hot Interconnects 6*, Palo Alto, CA, Aug. 1998.

[13] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a Robust Software-Based Router Using Network Processors. In *Proc. of 18th SOSP’01*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001.

[15] D. E. Taylor, J. W. Lockwood, T. S. Sproull, J. S. Turner, and D. B. Parlour. Scalable IP Lookup for Programmable Routers. In *Proc. of IEEE Infocom 2002*, New York, NY, June 2002.

[16] D. J. Wetherall. Active Network Vision and Reality: Lessons from a Capsule-based System. In *Proc. of he 17th ACM SOSP’99*, Kiawah Island, SC, Dec. 1999.

[17] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smart Pointers: Personalized Scientific Data Portals in Your Hand. In *Proc. of Supercomputing 2002*, Nov. 2002.

[18] K. Yocum and J. Chase. Payload Caching: High-Speed Data Forwarding for Network Intermediaries. In *Proc. of USENIX Technical Conference (USENIX’01)*, Boston, Massachusetts, June 2001.

[19] Y. Zhao and R. Storm. Exploiting Event Stream Interpretation in Publish-Subscribe Systems. In *Proc. of ACM Symposium on Principles of Distributed Computing*, Newport, RI, Aug. 2001.

[20] X. Zhuang, W. Shi, I. Paul, and K. Schwan. Efficient Implementation of the DWCS Algorithm on High-Speed Programmable Network Processors. In *Proc. of Multimedia Networks and Systems (MMNS)*, Oct. 2002.