Annotated Bibliography

This page gives a brief overview of the research done by the Network Systems Research Group over the last three to four years, and references the principal papers published during that time. Our work can most easily be classified as focusing on either (1) frameworks for implementing network protocols, or (2) various protocols implemented in those frameworks. This annotated bibliography is organized along these two main themes.

Clicking on a citation located in the body of this page will send you to the corresponding reference at the end of the page. Selecting this reference will then display a postscript copy of the paper. A postscript version of this annotated bibligraphy is available as file annobib.ps, and a *bibtex* version of the bibliography can be found in file xkernel.bib.

OS Support for High-Speed Networking

Protocols are typically implemented within the operating system (OS) of the hosts of which they run. Protocol implementations depend on the OS to provide several low-level mechanisms, including concurrent threads, allocatable blocks of memory, clock interrupts, and access to network devices. This aspect of our research addresses how operating systems should be designed to support more efficient protocols implementations.

Most of our work in this area has focused on a protocol implementation framework called the *x*-kernel. Our work on the *x*-kernel has passed through two major phases. We have also pursued several related projects.

x-kernel: Phase 1

The *x*-kernel began as a stand-alone operating system running on Sun3 workstations connected by an Ethernet. Our main result was to show that one could implement protocols in a modular, highly structured way, without sacrificing performance. An initial paper [Hutc91] describes the *x*-kernel's basic architecture, and demonstrates how this architecture is general enough to implement a wide variety of existing protocols, such as TCP/IP and RPC. It also reports that *x*-kernel implementations of these protocols perform as fast as (and often faster than) the same protocols implemented in their native operating system.

A second paper [O'Ma92] then demonstrates how one can take modularity even farther. It observes that typical network architectures share three important properties: their protocol graphs are simple, the nodes of these graphs (protocols) encapsulate complex functionality, and the topologies of the graphs are relatively static. The paper then goes on to describe a new way to organizing network software that differs from conventional architectures in all three of these properties. In this new approach, the protocol graph is complex, individual protocols encapsulate a single function, and the topology of the graph is dynamic. It also demonstrates that this architecture results in efficient network software. A related paper [O'Ma91] describes how this architecture can be used to extend TCP for high-speed networks.

x-kernel: Phase 2

During a second phase, we encapsulated the communications-core of the *x*-kernel, and treated it as a portable framework that could be embedded in any operating system. We also began to experiment with the *x*-kernel on RISC workstations connected by high-speed ATM and FDDI networks. As a consequence, our focus changed from one of trying to improve modularity without losing performance, to one of trying to keep up with improving network performance without giving up the modularity gains we had made earlier. The main thrust of this effort was in designing mechanisms to overcome the limitations of the workstation's memory bandwidth.

Our software platform was the Mach microkernel, with the *x*-kernel serving as the network subsystem. We *logically centralized* all the network protocols in a single *x*-kernel protocol graph, and *physically distributed* this graph throughout the system. That is, the protocol graph might span multiple application tasks, a dedicated network server, and the kernel. The decision as to what protection domain a particular protocol belongs in is delayed until configuration time (rather than OS design time) and, therefore, could be made based on how the system configurer is willing to trade performance against trust.

In our effort to optimize the performance of this architecture, the main challenge was to overcome the limitations of the workstation's memory architecture. The problem is that improvement in memory performance of workstations has not kept pace with improvements in processor performance and network bandwidth. Keep in mind that we were considering the movement of data through a microkernel-based system, where device drivers, network protocols, and application software all potentially reside in different protection domains. In other words, we were attempting to demonstrate that it is possible to achieve high application bandwidth regardless of the structure of the OS.

An overview of this effort can be found in [Drus93b]. This paper makes a case for limited memory bandwidth being the main problem in turning good network bandwidth into equally good application-to-application throughput, and outlines various techniques that can be employed to overcome this problem. Follow-on papers then look at specific issues and techniques in more detail: [Page94] investigates the effectiveness of the data cache in handling network I/O; [Bail91] and [Drus94] consider the interactions between the network adaptor and the operating system; [Drus93c] propose a new mechanism, called *fast buffers* (fbufs), for transferring network data across protection domains; and [Abbo93b] define and evaluate a technique for using *integrated layer processing* (ILP) to reduce the number of loads and stores required to process network data.

Other Related Projects

Motivated by our experiences with the *x*-kernel, we have also pursued several tangent avenues of research. These are summarized below. Note that in the case of Lipto, many of the ideas guide the implementation of the *x*-kernel in Mach described above. In the case of Scout, the project is just getting underway.

Lipto:

An operating system that explicitly decouples modules and protection domains [Drus92]. Starting with the premise that modular operating system design is a good idea, this paper argues that modularity and protection should be treated as orthogonal issues. That is, modules should first be designed based on sound software engineering principles, and later partitioned into protection domains according to how the users of the system are willing to trade performance for trust. This is in contrast to most existing operating systems that implicitly bind module interfaces to protection boundaries. A follow-on paper [Drus93a] argues that efficient, incremental customization of OS services can be achieved using a two-fold strategy: an object-oriented architecture that relies on composition to facilitate code reuse and customization, and an OS structure that places a minimal set of trusted functions into the kernel, with all remaining services co-located with application code in user-level protection domains.

Morpheus:

A special-purpose programming language designed to implement communication protocols [Abbo93a]. Starting with the *x*-kernel model, Morpheus refines the base abstractions to the point that one can view them as language constructs. This paper also explores the compiler optimizations made possible by casting a protocol framework as a language.

Scout:

A configurable, communication-oriented operating system, whose performance is both predictable and scales with processor performance. The main ideas in Scout include an explicit path abstraction that represents the flow of control and data through the OS, a set of software tools that can be used to generate correct and efficient Scout code, and a set of low-level optimizations. A white paper that gives an overview of Scout can be found in [Mont94]. In addition, three software tools being designed for Scout, the Universal Stub Compiler (USC), the Dynamic Code Generator (DCG), and a packet classifier (PathFinder) are described in [O'Ma94], [Engl94], and [Bail94], respectively. Finally, [Mosb94] describes Scout's approach to lock-free synchronization.

Protocol Design

The second major thrust of our research is protocol design. Quite often (but not always) using the *x*-kernel as an implementation platform, we investigate the algorithms used by various network protocols, including both standard protocols (e.g., TCP) and new protocols of our own invention (e.g., Psync). For the most part, our focus is on protocols at the transport level, and higher.

TCP Congestion Control:

Vegas is a new implementation of TCP that achieves between 40 and 70% better throughput, with one-fifth to one-half the losses, as compared to the implementation of TCP in the Reno distribution of BSD Unix [Brak94]. This paper motivates and describes the three key techniques employed by Vegas, and presents the results of a comprehensive experimental performance study, using both simulations and measurements of the actual Internet, of the Vegas and Reno implementations of TCP.

Security:

Software subsystems that implement cryptographic security features can be built from small modules using uniform interfaces. The methods demonstrated in this paper illustrate how configuration flexibility can be achieved and how complex services can be constructed, all using the same building block modules [Orma94]. These allow the configuration process to be independent of algorithm details, while the algorithms used in the subsystem are obvious. [Kim95].

Using a software subsystem for network layer security, we implemented a version of the remote login protocol, rlogin, that is protected from authentication spoofing and revealing the cleartext form of passwords.

Reliable Multicast:

We have built a suite of protocols that collectively provide a communication substrate for constructing fault-tolerant distributed programs based on replicated processing. On overview of the system, which we call Consul, is given in [Mish93b]. A second paper reports our experiences with modularity in Consul [Mish93a]. Consul includes a membership protocol, a set of message ordering protocols, a failure detection protocol, and a recovery protocol. All these protocols depend on a low-level multicast protocol, called Psync, which preserves the causal ordering among messages [Pete89].

Image Transfer:

We have studied the problem of image transfer from an end-to-end perspective. This includes a simple algorithm for encoding images into network packets in such a way that the receiver can recover from dropped packets without requiring the sender to retransmit them [Turn92a]. Italso includes an anaylsis of how different encoding algorithms effect image quality in the presence of unreliable computer networks [Turn92b].

Cluster Computing:

With improved network technology, it is now feasible to build data parallel supercomputers using traditional RISC-based workstations connected by a high-speed network. This paper presents an in-depth look at the communication behavior of a suite of application programs implemented in the C* data parallel language [Turn94]. It also compares the performance of these programs on both a cluster of 8 HP 720 workstations and a 32 node (128 Vector Unit) CM-5. The result is that under some conditions, the cluster is faster on an absolute scale, and that on a relative, per-node scale, the cluster delivers superior performance in all cases.

Mach NetIPC:

This paper describes an implementation of the Mach IPC abstraction on a network [Orma93]. Our implementation, called NetIPC, is done in the context of the *x*-kernel, which provides a networking subsystem for Mach. The paper motivates the design choices we made, describes the *x*-kernel protocol graph that implements the design, and reports on the performance of the resulting system.

References

[Abbo93a] M. B. Abbott and L. L. Peterson. A language-based approach to protocol implementation. *IEEE/ACM Transactions on Networking*, 1(1):4-19, Feb. 1993.

[Abbo93b] M. B. Abbott and L. L. Peterson. Increasing network throughput by integrating protocol layers. *IEEE/ACM Transactions on Networking*, 1(5):600-610, Oct. 1993.

[Bail91] M. L. Bailey, M. A. Pagels, and L. L. Peterson. The *x*-chip: An experiment in hardware demultiplexing. In *Proceedings of the IEEE Workshop on High Performance Communications Subsystems*, Feb. 1991.

[Bail94] M. L. Bailey, B. Gopal, M. A. Pagels, L. L. Peterson, and P. Sarkar. PathFinder: A Pattern-Based Packet Classifier. In *Proceedings of the 1st Symposium on Operating System Design and Implementation*, Nov. 1994.

[Brak94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the SIGCOMM '94 Symposium*, Aug. 1994.

[Drus92] P. Druschel, L. L. Peterson, and N. C. Hutchinson. Beyond micro-kernel design: Decoupling modularity and protection in Lipto. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems*, pages 512-520, Yokohama, Japan., June 1992.

[Drus93a] P. Druschel. Efficient support for incremental customization of OS services. In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 186-190, Asheville, NC, Dec. 1993.

[Drus93b] P. Druschel, M. B. Abbott, M. Pagels, and L. L. Peterson. Network subsystem design. *IEEE Network (Special Issue on End-System Support for High Speed Networks)*, 7(4):8-17, July 1993.

[Drus93c] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 189-202, Dec. 1993.

[Drus94] P. Druschel, L. L. Peterson, and B. S. Davie. Experience with a high-speed network adaptor: A software perspective. In *Proceedings of the SIGCOMM '94 Symposium*, Aug. 1994.

[Engl94] D. R. Engler and T. A. Proebsting. DCG: An efficient, retargetable dynamic code generation system. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1994.

[Hutc91] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64#76, Jan. 1991.

[Kim95] G. Kim, H. Orman and S. O'Malley. Implementing a Secure rlogin Environment: A Case Study of Using a Secure Network Layer Protocol. In *Proceedings of the 1995 Usenix Security Symposium*, Salt Lake City, June. 1995.

[Mish93a] S. Mishra, L. L. Peterson, and R. D. Schlichting. Consul: A communication substrate for fault-tolerant distributed programs. *Distributed Systems Engineering Journal*, 1(2):87-103, Dec. 1993.

[Mish93b] S. Mishra, L. L. Peterson, and R. D. Schlichting. Experience with modularity in Consul. *Software---Practice and Experience*, 23(10):1050-1075, Oct. 1993.

[Mont94] A. B. Montz, D. Mosberger, S. W. O'Malley, L. L. Peterson, T. A. Proebsting, J. H. Hartman. Scout: A communications-oriented operating system. Technical Report 94-20, Department of Computer Science, University of Arizona, June 1994.

[Mosb94] D. Mosberger, P. Druschel, and L. L. Peterson. A fast and general software solution to mutual exclusion on uniprocessors. Technical Report 94-07, Department of Computer Science, University of Arizona, Mar. 1994.

[O'Ma91] S. W. O'Malley and L. L. Peterson. TCP extensions considered harmful. Request for Comments 1263, University of Arizona, Oct. 1991.

[O'Ma92] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110#143, May 1992.

[O'Ma94] S. W. O'Malley, T. A. Proebsting, and A. B. Montz. Universal stub compiler. In *Proceedings of the SIGCOMM '94 Symposium*, Aug. 1994.

[Orma93] H. Orman, E. M. III, S. O'Malley, and L. Peterson. A fast and general implementation of Mach IPC in a network. In *Proceedings of the 3rd Usenix Mach Conference*, pages 75-88, Apr. 1993.

[Orma94] H. Orman, S. O'Malley, R. Schroeppel, and D. Schwartz. Paving the road to network security, or the value of small cobblestones. In *Proceedings of the 1994 Internet Society Symposium on Network and Distributed System Security*, Feb. 1994.

[Page94] M. A. Pagels, P. Druschel, and L. L. Peterson. Cache and TLB effectiveness in processing network I/O. Technical Report 94-08, Department of Computer Science, University of Arizona, Mar. 1994.

[Pete89] L. L. Peterson, N. Buchholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217-246, Aug. 1989.

[Turn92a] C. J. Turner and L. L. Peterson. Image transfer: An end-to-end design. In *Proceedings of the SIGCOMM '92 Symposium*, pages 258-268, Baltimore, Maryland, Aug. 1992.

[Turn92b] C. J. Turner and L. L. Peterson. The effects of transfer encoding on image quality. In *Proceedings of the 2nd IEEE International Conference on Image Processing*, pages 63-67, Singapore, Sept. 1992.

[Turn94] C. J. Turner, D. Mosberger, and L. L. Peterson. Cluster-C*: Understanding the performance limits. In *Proceedings of the Scalable High Performance Computing Conference*, May 1994.

Larry Peterson / llp@cs.arizona.edu