M.I.T. Laboratory of Computer Science October 5, 1982

Computer Systems Structures Request for Comments No. 232

PROGRESS REPORT FOR COMPUTER SYSTEMS STRUCTURES

from D.P. Reed

Attached is the 1981-1982 progress report for the Computer Systems Structures group.

COMPUTER SYSTEMS STRUCTURES

Academic Staff

D.P. Reed, Group Leader

Research Staff

M. Greenwald

Graduate Students

B. Coan	J. Stamos
D. Daniels	D. Theriault
W. Gramlich	C. Topolcic
K. Sollins	

Undergraduate Students

R. Kukura	D. Solo
M. Novick	K. Yelick
C. Rubin	

Support Staff

D. Fagin

Visitors

Ø. Hvinden	L. Svobodova

1. INTRODUCTION

During the past year, the effort of the Computer Systems Structure group has been focused on development of tools and substrates appropriate for development of distributed applications systems. We find that the most interesting research problems arise from trying to exploit two "insurmountable opportunities": a) the opportunity to connect two autonomously managed independent computer systems with a computer network in order to share data, and b) the opportunity to use high performance networks and specialized server computers to build multi-user computer systems that are modularized in a way that has not been possible before with single, large mainframe computer systems.

The first opportunity, data sharing, became apparent when the Arpanet computers first began to exchange data. The level of interconnection on the Arpanet, however, has never been particularly high. The best example of distributed applications within the Arpanet has been the mail system. Higher level applications that share databases across the network have been rare and extremely ad hoc. Our goal in this area is to develop substrates, such as the Swallow prototype described below, that can support applications at independent sites that can be later combined into larger applications while maintaining the autonomy of the original applications. The new research problems in this area result from two characteristic issues; the first issue is autonomy, that is, the fact that there is no "central administrator" who controls what is done on each computer in the distributed system, while the second issue is "growth by federation", that is, that adding a single gateway between two independent networks of autonomous processors may all of the sudden create a single system with complete interconnection. Although this <u>federation</u> process is easy at the hardware level, the software structures developed for distributed systems have been hierarchical, with naming, protection, concurrency control, failure recovery, etc., managed by what amounts to a single central authority. Trying to combine two hierarchies results in a heterarchy that no longer functions, because there is a new ambiguity -- "who's on top?".

The second opportunity modularization arises from new local network technologies, work station technologies, etc., that allow the construction of what might be called "server-oriented systems". For reasons of reliability, economy of scale, and flexibility, it is often convenient to design a distributed system consisting of a set of workstations with no secondary storage or only very small amounts of local secondary storage, with the bulk of secondary storage being provided by one or more shared, specialized data storage service machines. Similarly, specialized services such as printers, image scanners, or wire-wrap machines, may be attached to the net rather than directly to any particular work station. These new structures present problems of reliability, performance, protection, and coordination that differ significantly from the same problems as they appear in centralized time-sharing systems with many attached peripherals and file storage devices. The Swallow repository, which is a specialized data storage server computer, is a prototype of one kind of shared service.

In addition to developing substrates and servers, such as those above, we have also been working on several specialized network protocols. There protocols called non-fifo protocols, achieve extremely high performance and extreme simplicity by ignoring the conventional wisdom of protocol design. Instead of many layers of protocol implementing virtual circuits, these protocols use end to end datagram transport, and involve the application in error recovery, flow control, and coping with out of order packet arrival. Our initial experience with these protocols leads us to believe that such protocols will be necessary to exploit the potential of high bandwidth local networks, long delay high bandwidth satellite connections, and internetwork coupling.

In the following sections, we summarize the results of the past year's work on Swallow, protection and authentication in distributed systems, protocol design, naming in distributed systems, and debugging of distributed systems.

2. THE SWALLOW SYSTEM PROTOTYPE

Over the past year, work on the Swallow system has focused on the development of the Swallow system repository prototype. The Swallow repository is a specialized data storage server that provides stable storage to any number of clients on a network. A Swallow system may contain any number of repositories and each user may use any subset of the available repositories to store his data. The Swallow repository participates in concurrency control and recovery algorithms designed by Reed [1], [2], to provide multi-site atomic actions. The Swallow repository design was also conceived with the intention that it could be based on write-once storage media such as optical disk technology now being developed in a number of places.

The Swallow repository was built on an Alto with a special additional large disk drive since that hardware was available to us at the time. Although the rest of the Swallow system was not available to use the repository, we began to test and tune the system during the past spring semester, so that it could be incorporated easily into the Swallow system once the rest of it is constructed.

The design of the Swallow repository; particularly its use of write-once disk, required the development of a new storage organization, called append-only storage, which naturally supports the object of Swallow which have dynamically varying size and which have multiple versions over time. This concept, first introduced by Reed in his doctoral thesis [1] and developed by Reed and Svobodova, is documented in a paper recently published by Svobodova [3].

The client interface to the Swallow system is provided by a software module in each computer called the broker. The design of the broker was begun during the past year, but it was decided that the final design decisions would have to wait for the arrival of appropriate work station computer hardware. At this point it seems likely that such work stations will be implemented on VAX II/750's which will arrive during the coming summer. Our next task then will be to finalize this design and determine how to integrate it with the operating system (UNIX) of the VAX.

3. PROTECTION AND AUTHENTICATION

During the past year we completed and tested our authentication server prototype and began to see how it could be used in securing various communications that currently go on in the lab. One result of this was a bachelor's thesis by Solo, who investigated the problem of securing a file transfer protocol [4].

The approach of using authentication servers has a flaw, which we view as a very important one. This flaw is that the authentication server used to authenticate one party to another is in a sense a central authority. As distributed systems grow larger and cross organizational and governmental boundaries, there may be no real single trusted central authority. Problems of authentication and protection across such boundaries will require and different and novel solutions. During the past year Topolcic has developed a technique for creating "digital guarantees" that can be used where a client and a server need a mechanism to enforce the satisfaction of remote requests in a decentralized system without such a centralized authenticator.

Consider this scenario. In a network of autonomous computers having varied resources, a client may request a service from some other node, the server. Since autonomous nodes within different organizations may be mutually suspicious, and since there may exist no universally trusted authority, some decentralized mechanism is necessary to assure the client that its requests will be honored. Topolcic examines some failures that can interfere with fulfillment of the remote requests and methods to control them.

One source of failure is the dishonesty of the server, which might return incorrect results, or may ignore some commitment it had previously made to the client. Cryptographic "Digital Signatures", as proposed by Needham and Schroeder [5], attempt to provide a binding "guarantee" from a server to a user. Needham and Schroeder's approach requires that the encryption keys be protected for extended periods of time. Such long-term protection, regardless of the security of the encryption technique, demands administrative controls that are difficult to identify and impossible to prove correct. Topolcic proposes a system of guarantees based on a hard to duplicate yet completely public characteristic function (rather than encryption) and a distributed method of monitoring and punishment based on a "User's Group" rather than a universally accepted judge. The characteristic function of a guarantee is placed by the server into the next guarantee it issues, forming a linked list which cannot be modified without changing the latest one issued, whose uniqueness is verified with real-time authentication. The User's Group is a collection of clients which monitor the server, exchange information about it, and enforce punishment by boycotting it if a member proves the server's dishonesty. A minority of non-participating or dishonest members cannot affect the correctness of the actions of the majority.

4. NAMING WITHOUT HIERARCHY OR A CENTRAL AUTHORITY

In nearly every computer system the naming mechanism consists of a hierarchy with pieces of a single global name space assigned to each user who can then assign names within those parts to objects of his own interest. A global name space presents problems in a system that grows by adding communications points between preexisting but independent distributed systems on independent networks. Where each system had its own global name space in which all names were unique, the combined system has name conflicts. Where each system had a central authority that partitioned the name space before, there are now two or more central authorities in the combined system. Thus the notion of a hierarchy tends to break down in these federated systems (which result from interenterprise linkage, in particular). A new approach to naming is needed.

If we look at the "human distributed system", we find a potential for similar problems in human language. Here, a distributed system is analogous to a human community, and the computers are analogous to individual people. As thousands of years of human history have shown, people have little trouble with the problems with integrating name spaces that so confound computer systems. We felt that by exploring this analogy, a new approach to computer naming of things could be developed, which would be flexible, natural, and free of the problems of hierarchy.

Sollins has been exploring these ideas in her Ph.D. thesis which was begun during the past year. Although the ideas are still at an early stage, they promise to be significant.

Sollins proposes contexts as the system provided tool for name management. In addition to the goal of non-hierarchical naming mentioned above, Sollins' work will provide a unified naming framework for all entities in a distributed computing environment where each node must be capable of independent operation without dependence on others. This independent or autonomous operation leads to the conclusion that names cannot be guaranteed to be unique. It is this assumption of autonomy that has led away from more traditional remote name servers. All these forces combined have led to the model of contexts proposed in this thesis. There are two sorts of functions provided by contexts. First, a context might translate a name into something else, either another name or an address. Second, a context might answer the question of whether two names name the same entity. Contexts allow names to be assigned to any entities, people, processes, data, or whatever else needs to be named.

There are several different kinds of names that are used commonly in naming during human interactions. These are modelled in the naming framework provided. One is the ability of the name user to assign nicknames. Another is the ability to name by description. In this case, an entity might be described by a collection of descriptive attributes. The namer will name the entity by indicating a logical combination of these descriptions. A third form of name is what in this work is called generic naming. This is a means of naming a class of entities by using a single name. An example of such a class is the set of implementations that provide a particular service, although the entities named by a generic name need to be the same type of entity. Each namer should be able to use whatever name he chooses for the entities he wishes to name. In fact, although superficially these three kinds of names appear to be different in nature, they are not. Any two can be described in terms of the third, or all in terms of generalized names or labels. For instance, assuming there are only generic names, a nickname is simply a generic name that names only one entity, which also has at least one other name. The entity named by a description is simply the intersection of those entities named by a collection of generic names, one for each attribute in the description.

There are a number of issues related to contexts and naming that remain to be investigated. The following is a partial list:

1) the relationship between naming as provided by contexts and protection and authentication.

2) how contexts will be used, individually and in combination with each other.

3) how and when contexts should and should not be shared and by whom or what.

4) a detailed example of the use of contexts.

This work is in progress and will continue on the assumption that contexts are a useful mechanism for name management in future systems.

5. DISTRIBUTED DEBUGGING

Gramlich has undertaken a Ph.D. thesis to investigate a debugging methodology called *checkpoint debugging*. Basically, checkpoint debugging works by taking regular checkpoints of a program. A checkpoint consists of a fixed part and incremental part. The fixed part of a checkpoint consists of a single consistent snapshot of the relevant program state. The incremental part of a checkpoint consists of a sequential recording of all program input since the time of the program snap-shot. When a program failure occurs, it is possible to use the checkpoint information to repeat deterministically the failure as many times as necessary to locate the program failure. This is done by going to a previous checkpoint, loading the fixed part, and reexecuting the program using the incremental part for program input. The major advantage of checkpoint debugging is that it converts a large class of non-deterministic failures (i.e., non-repeatable) into deterministic failures (i.e. repeatable). It turns out that it is much easier for a user to locate and correct a deterministic failure than a non-deterministic failure. A trial implementation of this debugging system will be implemented in CLU for Berkeley Unix.

6. NON-FIFO PROTOCOLS

The so-called end-to-end argument [6] is a protocol design rule that says, in effect, that the application usually knows best how to cope with such problems as loss of messages, protection, flow control, duplicate message detection, coping with messages arriving out of order, and so forth, which traditionally have been in the domain of the communications subsystem. A corollary to the argument is that the communications subsystem may be paying a very high performance price that results from implementing solutions to these problems at too low a level in the system. In fact, in order to have any layering of function at all, it is necessary to place some functions at least inside the communications system and below the application. But if the application knows best, and implements all these functions for itself, there is little or nothing left in the communications subsystem to layer.

This line of reasoning seems to be borne out by a couple of protocols we developed recently for two specialized applications. By exploiting natural properties of the applications themselves, we were able to accomplish the flow control and error control functions involved in communication in a much simpler and more efficient way. Similar simplifications and efficiencies were obtained in the protocols developed for the Swallow distributed data storage systems communications needs. [7]

The first of these specialized protocols was a protocol called BLAST. We observed that most file transfer protocols, even when implemented on very high bandwidth local networks, such as a ten megabit-per-second ring network, had disappointingly slow information transfer rates. For example, a file transfer from an Alto to another Alto on a three-megabit-per-second Ethernet rarely exceeds 75,000 bits per second, while the underlying communication medium and disks are capable of much higher rates. The "accepted wisdom" for implementing file transfer protocols is to access the file as a sequential stream of characters, transmit each character successively over a virtual circuit between the two computers, and store the file sequentially on the remote machine using a stream oriented file system interface. The stream interface to files and the virtual circuit, of course, are implemented by fairly complex mechanisms that take the raw blocks of the disk or the raw packets of the network and transform them into something that is quite different, a reliable ordered stream of bits. This transformation, or extraction, is not natural. The result is that the application has very little control of the timing of what is going on and the timing itself is critical. To cope with a lost packet in the network, for example, the network virtual circuit implementation introduces a small amount of delay in the communications. The result of such delay will be delay in accessing the next byte of the file, but delay in accessing the file can result in a significant real time delay while the disk rotates one whole revolution. This, in turn, disrupts the smooth flow of data to the receiver across the network, which can effect both the flow control to the receiver, and also the rate at which packets can be stored on disk at the receiver.

Our new BLAST protocol is based on a very simple idea. The sender transmits the blocks of the file each in a separate packet labeled with a block number of the file. Since the block number is in each packet, the receiver can place each packet directly in the file at the time he receives it. If packets are lost in the network, there is no need for the sender to retransmit those packets right away -- instead the sender can continue to transmit the rest of the packets of the file. When the sender thinks that all the packets have been transmitted to the receiver, he polls the receiver with a single packet and the receiver responds with a packet that indicates the set of file blocks that remain to be transmitted. The sender then rereads just those blocks of the file that need to be retransmitted and resends them. This process converges after a few rounds. Neither end needs buffering for error control or reordering. Duplicate packets are not a problem since they may be stored again in the same place and reordered packets just get stored in a different order into the file. The network and communication system serve as a way of getting packets from one end to the other only.

To understand why this protocol is interesting consider a satellite link. Typical satellites have channel capacities of maybe up to 50 megabits per second, but the delay due to speed of light is on the order of seconds from end to end. Satellite channels also tend to have a fairly high probability of packet loss. Traditional stream protocols do not cope well with this combination of high bandwidth and long delay.

A single packet loss discovered at the receiver may require one or two seconds before it can be filled in by a retransmitted packet. Meanwhile tens of millions of bits have been transmitted over the network and must be buffered at both the receiver and sender until the retransmitted packet arrives (thus megabyte buffers are needed). In order to maintain throughput on the order of 50 megabits per second, these millions of bits must then be written instantaneously onto the disk at the receiver. In contrast the BLAST operates quite reasonably on such a network with no buffering at the receiver or sender at all. The round trip delay only affects the final polling to determine if all packets have arrived, and for long files, this is negligible. The instantaneous dumping of the entire receive buffer to disk will no longer be necessary.

A similar protocol has been developed so that a remote single user computer can access a bitmap display such as that on the Alto across a high performance local network. In BLINK, each end maintains a copy of the bitmap for the screen. As the computer changes regions of its bitmap, the updated regions are transmitted to the remote display. Since updates to nonoverlapping regions may be applied to the display bitmap in either order, a lost packet need not delay processing of later packets arriving at the receiver. Periodically, every hundred milliseconds or so, the display sends a packet containing a version number for every region on the display. The computer then retransmits any portions of the bitmap that have not made it to the display bitmap. The performance arguments for this approach are similar to those for BLAST.

References

1. Reed, D.P., "Naming and synchronization in a decentralized computer system," TR-205, MIT Department of Electrical Engineering and Computer Science, , September, 1978.

2. Reed, D.P., "Implementing atomic actions on decentralized data," *Communications of the ACM* (1982). Accepted for publication.

3. Svobodova, L., "A reliable object-oriented repository for a distributed computer system," In ACM Eighth Symposium on Operating Systems Principles, Pacific Grove, Ca., December, 1982, 47-58.

4. Solo, "User authentication and security modifications for TFTP," S.B. Dissertation, MIT Department of Electrical Engineering and Computer Science, May, 1982.

5. Needham, R., and Schroeder, M., "Using encryption for authentication in large networks of computer," In Communications of the ACM, December, 1978.Volume 21, 12.

6. Saltzer, J.H., Reed, D.P., and Clark, D.D., "End-to-end arguments in system design," In Proceedings of the Second International Conference on Distributed Computing Systems, Paris, France, April, 1981..

7. Reed, D.P., "SWALLOW: a distributed data storage system for a local network," In Local Networks for Computer Communications, North-Holland, New York, N.Y., 1981, 335-373.A. West and P. Janson (editors).

Publications

Reed, D.P., "Implementing atomic actions on decentralized data," accepted for publication by Communications of the ACM, New York, NY, 1982.

Reed, D.P. and Svobodova, L., "SWALLOW: a distributed data storage system for a local network," in Local Networks for Computer Communications," A. West and P. Janson (Editors), North-Holland Publishing Company, New York, NY 1981, pp. 335-373.

Saltzer, J.H., Reed, D.P., and Clark, D.D., "Source routing for campus-wide internet transport," in Local Networks for Computer Communications," A. West and P. Janson (Editors), North-Holland Publishing Company, New York, NY, 1981, pp. 1-23.

Schiffenbauer, R., "Debugging in a distributed system," MIT/LCS/TR-264, MIT, Laboratory for Computer Science, Cambridge, Ma., September 1981.

Svobodova, L., "A reliable object-oriented repository for a distributed computer system," ACM Eighth Symposium on Operating Systems Principles, Pacific Grove, Ca., December 1982.

Theses Completed

Daniels, D., "Query compilation in a distributed database system," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., February, 1982 (also S.B. degree).

Lederman, A., "A Pascal structure oriented system," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., June 1981.

Schiffenbauer, R., "Debugging in a distributed system," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., June 1981.

Solo, D., "User authentication and security modifications for TFTP," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1982.

Stamos, J., "Grouping strategies for an object oriented virtual memory," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., February, 1982 (also S.B. degree).

Ulloa, M., "A window manager for microcomputers," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1982.

Weiss, S., "Managing software evolution in an object-oriented environment," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1982.

Theses in Progress

Gramlich, W., "Checkpoint debugging," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected date of completion, June 1983.

Ketelboeter, V., "Forward recovery in distributed systems," S.M. thesis, MIT

Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected date of completion, August 1982.

Mendelsohn, A., "A framework for user interfaces to distributed systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected date of completion, June 1983.

Sollins, K., "Name management in a distributed system," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected date of completion, June 1983.

Topolcic, C., "Ensuring the satisfaction of requests to remote servers in distributed computer systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected date of completion, August 1982.

Conference Participation

Reed, D.P., Program Chairperson, ACM Eighth Symposium on Operating Systems Principles, December 1981.

Svobodova, L., "A reliable object-oriented repository for a distributed computer system," ACM Eighth Symposium on Operating Systems Principles," Pacific Grove, Ca., December 1981.

Talks

Sollins, K., "Distributed computing at MIT ", University of Southern California, Los Angeles, Ca., December 1981.

Reed, D.P., "Protection issues in distributed systems," talk to JIPDEC group on security, audit and control, Cambridge, Massachusetts, April 1982.

Reed, D.P., "An overview of distributed systems research at MIT LCS," Siemens, Munich, Germany, May 1982.

Reed, D.P., "Non-FIFO protocols or streams considered harmful," IBM Zurich Research Laboratory, Zurich, Switzerland, May 1982.

Committee Membership

Reed, D.P., Program Chairperson, ACM Eighth Symposium on Operating Systems Principles, December 1981.