A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication*

Ken Birman
Dept. of Computer Science, Cornell University

October 21, 1993

Abstract

In a paper to be presented at the 1993 ACM Symposium on Operating Systems Principles, Cheriton and Skeen offer their understanding of causal and total ordering as a communication property. In this brief rebuttal. I present some responses to their criticism, and also explain why I find their discussion of causal and total communication ordering to be narrow and incomplete.

1 Background

In a paper to be presented at the 1993 ACM Symposium on Operating Systems Principles, Dave Cheriton and Dale Skeen offer their understanding of causal and total ordering as a communication property. In this paper, I want to I respond to their criticisms from the perspective of my work on Isis [Bir93, BJ87a, BJ87b], and the overall communication model that Isis employs. I assume that the reader is familiar with the Cheriton Skeen paper, and the structure of this response roughly parallels the order of presentation that they use.[1]

Isis is not the only system to use the causal and total ordering properties that Cheriton and Skeen discuss. There is a growing community of researchers using techniques similar to those in Isis, including Ladin and Liskov's Gossip project at MIT [LLS90], Powell and Verissimo's work on Delta-4 [Pow91], Peterson and Schlicting's work on Psync at Univ. of Arizona [PBS89], Dolev's Transis project [ADKM91], developed by a group at Hebrew University, Kaashoek's work on reliable multicast in Amoeba [KTHB89], Melliar-Smith's and Moser's work on the Trans and Total communication primitives at U.C. Santa Barbara [MSMA90, AMMS+93], and others. There is also a next-generation of Isis, called Horus, under development by my group [BR93]. However, in my remarks below I'll focus on Isis, and in fact as I read the Cheriton/Skeen arguments, it seems to me that they are basically using CATOCS as a synonym for Isis in many parts of their paper.

I want to make several major points:

- Their paper isolates CATOCS ordering properties from the context in which these have been used by systems like Isis, namely a comprehensive programming environment with an execution model, reliability properties, and in the specific case of Isis, a rich collection of process group programming tools. I'll argue that the CATOCS properties are natural and valuable in this more elaborate context, regardless of their merit in systems that lack such programming models.

*This is a short form of a longer paper available as Cornell TR 93-1390. Research on Isis was supported by the Department of Defense under ARPA/ONR grant N00014-92-J-1866.

[1] Dave Cheriton showed me several intermediate drafts of the CATOCS paper, and I provided quite a bit of feedback. I am grateful to Dave for making a serious effort to avoid misrepresentation of the Isis approach. Unfortunately, I did not have a chance to react to the final version of the paper, and many of the points with which I disagree most strongly were introduced late in the revision process. On the other hand, Dave has helped me understand the points he and Dale wanted to make, and I found his explanations very useful in preparing this response.

- A basic view of the Cheriton and Skeen paper is that while many applications have some sort of ordering need, the ordering is rarely a causal or total ordering and is often inaccessible to the communication subsystem. I'll suggest that CATOCS properties are beneficial because they greatly simplify the programming model seen by the user, and that this is an important reason to favor the approach.

- Cheriton and Skeen present several arguments that are actually incorrect. For example, their quadratic overhead claims would only apply in systems with a specific, highly unlikely, pattern of communication. Overhead in systems that implement CATOCS ordering properties is actually very low; often not much different from the overhead built into point-to-point mechanisms such as TCP.

- Although claiming that they want to focus on CATOCS ordering independent of any system model – and indeed, Cheriton and Skeen do not discuss the Isis virtual synchrony execution model or any of the other models used by researchers elsewhere – they do introduce issues of durability, atomicity, and transactional consistency. This gives a biased and incomplete picture, because it omits the model we actually use in conjunction with CATOCS ordering in the papers they cite. Virtual synchrony is a middle ground between inconsistent environments and transaction, offering a fault-tolerant, dynamically reconfigurable execution environment and programming model, but without the overhead associated with durability of transactions on stable data on an external disk.

- This said, it is still reasonable to look at the issue of exploiting CATOCS ordering in a transactional setting, and my own work has done so in two papers [JB89, Bir94]. But Cheriton and Skeen do not cite this work, and do not present any of the arguments advanced in these papers in favor of combining reliable ordered multicast and process groups with a transactional model. Instead, they pose the problem but then present their own interpretation of what this means, attributing it to me through vague references to other papers of mine. Presented fairly, it may be that the transactional community would see more merit in this approach than in the version advanced by Cheriton and Skeen.

2 Styles of Distributed Computing Systems

I believe that most distributed systems seek to achieve the following properties.

- *Reliability and fault-tolerance.*
- *Consistency.*
- *Concurrency, high performance.*
- *Ability to replicate data.*
- *Reconfiguration after failures and recoveries.*
- *Support for grouping or streaming sequences of operations.*

Even when developers agree that a property is desired, they often mean very different things by these terms. One system may seek to recover data from a disk; another may seek only to reconfigure itself to exclude faulty components. One may be satisfied with a cache of possibly stale data, while another requires that sets of processes have coherent, local, copies of dynamically changing control parameters.

One premise of work on Isis is that no single technology can make all users happy – it seems that at least two types of technology are needed. The big split is between "database-style" distributed systems and "control/communication" systems. The former are worried about data files that are generally not replicated. Recovery from a failure means waiting for the faulty system to restart, cleaning up the mess created by the crash, and resuming normal processing using the restored data files. Obviously, this can involve a long wait, but the cost of replication has generally been seen as outweighing its benefits. For example, commercial OLTP systems that don't replicate data can support thousands of transactions per second; replicated database technologies tend to be a factor of ten or more slower for similar applications. To the extent that databases do use replication, they tend to replicate objects that change infrequently. And,

of course, database systems commonly offer ways to group operations into transactions, which are scheduled in accordance with a serializability criteria and guaranteed to have atomic effects on persistent data.

By "control/communication," I mean a class of systems that operate something, be it the system itself, a network, external devices, delivery of analytic data to brokers, or whatever. The military would call this command, control and communication applications; a bank or brokerage would call them data distribution systems. These systems, in contrast to database systems, are often required to maintain continuous availability by reconfiguring from a failure and resuming function. This forces them to abandon any data managed by a failed server: the system can't wait for recovery, although it may try to salvage data when the server restarts. Online progress is viewed as the main goal.

A good example of a database-style application would be a banking system. Automated teller machines need to issue a transaction against the bank's records before issuing money, so the delay between pressing "enter" and receiving money is a direct measure of the speed of the transaction subsystem. The data on the disk is critical – it has the user's current bank balance, and if a database log that might contain a withdrawal is unavailable, its a safe bet that the bank won't approve further withdrawals until the database recovers.

A good example of a control/communication application is the console display of an air-traffic control system. A controller is in no position to wait for a database to recover just because some computer has failed. Such a system either reconfigures and makes forward progress, with minimal delays, or it is useless. Internally, systems such as this are often composed of programs that monitor data streams, filtering input and producing output data streams, which are consumed by further layers of filtering programs and by application programs. Other examples include the overhead displays in a stock exchange, cell controllers in a VLSI fabrication setting, and risk management software in a financial trading system.

Database systems "group" operations into transactions. This is a fundamental principle of the approach, and works extremely well for non-distributed database applications. Distributed database systems often weaken the transactional model (for example, consider Pu's work on epsilon-serializability); the costs of a literal interpretation of transactional serializability are just too high to tolerate in distributed settings.

Control applications tend to be more concerned with relationships between sequences of asynchronous messages. This is a common pattern in such systems and reflects in a natural way the internal structure of these systems, based on the filtering of data streams. Database-style event groupings don't always arise in these systems, but when they do, a mixture of a streaming model and a transactional model is needed – not one to the exclusion of the other.

Isis emerged from our interest in control/communication applications, with a fairly broad interpretation on what these might be. Isis seeks to aid the user in building any distributed application concerned with on-line availability despite failures, consistency, and replication – with "controlling itself" in a real-world asynchronous setting. The properties and features of Isis reflect subtle issues that arise when one tries to maintain the consistency of a distributed system while also reconfiguring to make progress despite a failure.

3 A brief history of process group technologies

In 1985 Dave Cheriton and Willy Zwaenepoel suggested that control/communications distributed systems be structured using process-groups [CZ85]. They implemented the idea, showed that it could perform well (particularly over hardware broadcast, using a filtering scheme to discard unwanted messages), and discussed applications, such as the publishing paradigm for communication. V, however, lacked any ordering or reliability properties. It was a best-effort system, trying to be reliable and ordered but not slowing things down for this purpose. V was not fault-tolerant, nor did it offer a way to build fault-tolerant applications.

Isis picked up the process group idea, adding communication reliability and ordering properties to the approach, and demonstrated the ability to support fault-tolerant group-computing tools and utilities, which in turn have been used successfully in a number of demanding applications. This work started in 1986, and was published starting in 1987. As the reader may be aware, Isis offers quite a variety of tools, including ways of replicating data within a process group, monitoring membership, doing synchronization, adding members to a group and transferring state to them so as to bring them up to date, subdividing computation to increase fault-tolerance or gain performance through parallelism, n-version programming, and so forth. On top of this, Isis layers a collection of applications, including a program-to-program "news" tool, a reliable network file system, a spooling/load-balancing utility, and a reactive control system. Each of these higher

level applications works by mapping some behavior visible to the user down to a set of process groups, within which computation and communication occurs.

The key feature of Isis, which made this approach possible, is a style of synchronization in which group membership changes, communication to groups, and atomicity are linked into a programming model in which the developer of an application has a simple way to think about distributed executions, and has strong guarantees of consistency and fault-tolerance. As mentioned earlier, this model is called virtual synchrony.

As implemented, Isis has a series of layers. High level applications are layered over group communication mechanisms, which are in turn implemented using synchronization and ordering properties. The lowest layers transport the data – currently, Isis does this over TCP/IP, IP multicast, or other available mechanisms. In our work, the CATOCS ordering issues enter just above this lowest layer of the system: they do not stand by themselves, and relatively few Isis application developers are explicitly aware of the causal aspects of Isis – although totally ordered communication arises all over the system, which is why we call it virtually synchronous: it *looks* synchronous to the programmer.

In the introduction I pointed to a number of other systems that also use process groups. The approach has become widely popular, although not all process group systems use CATOCS ordering properties. Again, since I feel some pressure to keep this short, I will not discuss this other work, although the experience of other researchers is certainly germaine. The impressive performance attained in the Ameoba, Transis, and Totem work, for example, represents additional evidence that there are many ways to engineer a system so that CATOCS properties will not cost a great deal. (The performance of Isis, by the way, is completely competitive with any of these alternatives; the performance of our next system, Horus, is expected to be even better).

Finally, Dale Skeen's company invented TIB in 1989. TIB is architecturally similar to V but has some fault-tolerance features and presents process groups though a "subject addressing" interface, along the lines of network news, but for program-to-program communication. Like V, TIB has few reliability features beyond its ability to roll from a failed server to a reliable one. So, Cheriton and Skeen both work from a context of systems that have a process-group mechanism of some sort, but without CATOCS properties, atomicity mechanisms, or a distributed programming model with the strong semantics of the Isis virtual synchrony approach. I understand that a paper on TIB will appear side-by-side with the Cheriton/Skeen paper in the 1993 SOSP.

As for Isis, readers interested in a more complete discussion are referred to the December 1993 issue of CACM, in which a fairly detailed article appears. One observation that emerges from this is that in the CACM article – and Isis in general – these CATOCS ordering properties are mostly associated with one corner of the overall system. In the context of an execution model, they provide a way to allow asynchronous communication without exposing the programmer to complexity associated with race conditions and the like. So, for me, CATOCS is not the whole story – it is an important aspect of a very different story. And, I think that this is true of *every* system in which CATOCS properties have played a serious role. When Cheriton and Skeen present CATOCS out of context, as they do, one should really view their paper as advancing a proposal for a new kind of distributed system, which they conclude will not work out well – the paper simply doesn't address the distributed systems that currently use these properties, although it purports to do so.

4 The arguments

With this out of the way, let me review the the CATOCS arguments as advanced by Cheriton and Skeen:

4.1 Utility of causal order

Why are these properties useful? The long story is a bit long for a short rebuttal, and I hope that readers will consider looking up the December CACM article where I discuss this in some detail. Briefly, we use causal ordering for two things, and one of them is really more concerned with total ordering, not causal ordering.

The first use is to avoid gaps in the sequence of messages sent by a process that experiences a failure. Suppose that process p fires off a series of messages, asynchronously, and then crashes. m_0 is the first of these

and m_n is the last, and m_n gets through to some of its destinations (I tend to think in terms of multicasts, as do Cheriton and Skeen – most communication in systems that would use CATOCS ordering is multicasts).

Now, since p has crashed, various things may have happened to m_0. If we are using an atomic multicast protocol, m_0 will have reached "all of its destinations, or none of them" – e.g. the crash will trigger a flush mechanism and if anyone has a copy of m_0 buffered, they will retransmit it to complete the multicast on behalf of p, with duplicates being discarded. But, what if m_n gets to its destinations and m_0 gets to none of them? Cheriton and Skeen have a lot to say about application-level ordering mechanisms, in which the application detects that a message, such as m_n, has arrived before something else, and would delay m_n accordingly. In their approach one would expect the programmer to also realize that m_0 was somehow lost because of the crash, and to somehow fix the system state up. Well, I have always believed that problems such as this are too complex to leave unsolved. One major use of causal order in Isis is to provide a guarantee that such things won't happen: the Isis definition of atomicity is that if a message m_n is delivered to a destination that doesn't crash, than not only does m_n reach its remaining destinations, but so do any causally prior messages – like m_0 in this example.

So, the first reason that we use causal ordering in Isis is that we want to support asynchronous communication, for reasons of performance, but we also want to protect the user against gaps in the past. And the causal delivery guarantee is the way that we describe the resulting system property.

The second reason for using causal ordering is based on total ordering. Some time ago, Leslie Lamport and Fred Schneider started to talk about the "state machine" approach to distributed computing, in which one sends messages to a group of processes that basically execute the same actions in the same order [Sch86]. Since all processes get the same events in the same order, they can use the same algorithm to take identical actions, or they can use other information (process identifiers, rank in process groups) to divide the work in some way. This yields various desirable properties. Isis uses a related approach heavily, to manage replicated data, trigger parallel computations, for fault-tolerance, etc.

But, it turns out that protocols for guaranteeing totally ordered message delivery are not cheap. More precisely, they are only cheap under certain conditions, namely when the sender of a message has some form of mutual exclusion: In this case, concurrent multicasts will never initiate conflicting actions. A weaker ordering property is then sufficient to still let the process group implement a state-machine style of algorithm. For example, take the extreme case where all the messages to some group originate in the same single process. This one origin of messages is the *only* sender. Clearly, if messages are transported over FIFO channels, a FIFO multicast will be ordered just as a totally ordered multicast would have been (this fudges a number of subtle issues associated with dynamicism and failures, which Isis addresses, but hopefully the main point is clear).

A causally ordered multicast works in just the same way, but it allows the "single" sender to move around, provided that permission to send is through causally ordered communication. Thus a typical Isis process group will run one or more state machine computations, using causal ordering for concurrent multicasts when we can arrange that the sender has locked out potentially conflicting multicasts. We think in terms of totally ordered algorithms (hence, "virtually synchronous" executions) but the actual message delivery ordering can interleave messages from independent senders, and the degree of asynchrony can be very high (hence, highly efficient pipelined communication can occur).

Cheriton and Skeen put forward a number of much more complex situations in which various kinds of application-level ordering are needed. In Isis, the Cheriton/Skeen examples could be much simplified by an environment with CATOCS ordering, because of the freedom from gaps after crashes and the ability to use asynchronous message passing safely. But Isis wouldn't address the application ordering issues they introduce in a direct way, and these aren't really the way that CATOCS ordering is intended. Our goal was always to create a *simple environment* for building asynchronous codes that are consistent and fault-tolerant. Programmer-oriented ordering tools are another matter entirely.

Cheriton and Skeen tend to take an argument to the extreme, showing that it then looks absurd. Well, turnabout is fair play. Their arguments appear to apply just as easily to the FIFO property of a TCP connection or stream as to the causal property against which they aim it. After all, when two processes communicate over TCP, they may do so more for reliability reasons than for ordering purposes. Perhaps one could argue that TCP is over-ordered, and that some applications would be compelled to include additional ordering information such as timestamps in the messages transmitted. One could then conclude that TCP

should be reliable but not ordered.

Obviously, most users find that the cost of ordering in TCP is very small and that the simplification in programming style afforded is substantial. Any performance benefit of an unordered TCP would be offset by the increased complexity of using such a transport.

Causal order is just a generalization of FIFO order. The same reasoning that underlies our tolerance of TCP, despite its "overly simplistic" model of application ordering requirements, justifies the belief that causal order can be useful as a tool in complex distributed systems. But, to reiterate, this is a tool in the sense that it simplifies the programming environment – not one that would normally be used directly by the application programmer to implement an order-based algorithm, such as in the Cheriton/Skeen fire alarm example.

4.2 Overhead concerns

On the cost issue, Cheriton and Skeen advance an argument that causal ordering inherently incurs an overhead quadratic in the system size. I see several serious problems with this analysis.

The first problem is the basic premise. Basically, the Cheriton/Skeen analysis studies a group of processes that *all send messages to one another*, as the system scales up. Moreover, their discussion of causality implicitly assumes that *most communication is point-to-point,* and that this point-to-point communication is also *asynchronous and needs to be represented in the causal ordering graph* for the system. This scenario is unrealistic:

1. In any large system, processes will make heavy use of multicast to one another. Multicasts are initiated by relatively few processes; other processes are mostly consumers of data, or mostly communicate point-to-point with these few producers. In systems where most processes do initiate multicasts, the pattern of communication tends to be many-to-many within small groups, not all-to-all where "all" is the entire system.

2. Asynchronous point-to-point communication introduces two issues that most systems prefer not to treat

   (a) Exposure to safety violations (if the sender of a point-to-point message crashes before is delivered and when no other process has a copy).

   (b) The need to maintain a complex causal ordering structure.

   Consequently, most real systems – and specifically, Isis – treat point-to-point communication using protocols that separate the point-to-point communication burden from other communication.[2] In Isis, no multicast is ever sent when a point-to-point message is unstable (although point-to-point messages need to carry CATOCS information about prior potentially unstable multicasts). Streams of point-to-point messages can be sent, TCP-style, but then a short delay will occur (until the last message is acknowledged) before a multicast can be initiated.

   Consequently, the CATOCS representation of causality can omit point-to-point traffic, because point-to-point messages are always stable when subsequent multicasts are initiated.

3. Even if one did chose to use a CATOCS ordering for asynchronous point-to-point communication, perhaps in a setting where fault-tolerance was not a concern, the system would still saturate at some maximum number of new messages per second. So, *even* if one focused only on point-to-point communication, given messages that become stable in roughly constant time, the number of potentially unstable messages will still be bounded by a constant!

[2]This is done out of concerns that failures might otherwise leave gaps in the causal past. The problem is that if P sends an asynchronous message to Q, point-to-point, then sends a causally dependent message to some group G, and then fails, the message to Q could be lost. If one does not delay the delivery of messages in G until it is known that Q received the point to point messages, it will never be safe for a member of G to talk to Q: a causally prior message has vanished from the system. So, point-to-point communication in systems like Isis is often by some form of RPC protocol, in which new multicasts are inhibited until the stability of the point-to-point message (or a stream of them) is established.

To summarize, Cheriton and Skeen's argument supposes a great deal of point-to-point message passing, which is questionable for several reasons. Moreover, due to concerns about gaps after failures, systems like Isis normally use a special scheme to provide CATOCS ordering for this case. The general form of CATOCS ordering arises *only* for multicast messages.

Now, even with this observation, the Cheriton/Skeen analysis would appear to claim that Isis should still be faced with a quadratic cost, because of the need to "buffer" messages for reasons unrelated to CATOCS ordering – namely, multicast atomicity. Before discussing this argument, it is useful to stress again that any communication system will ultimately be limited by flow-control and bandwidth considerations. Thus, even if presented with an unlimited number of multicasts originated at every process in the system, any real system will only deliver them at some maximum rate above which congestion occurs.

Given this setting, there are two issues: representation of the causal ordering information for multicast messages, and buffering of multicast messages for atomicity reasons (stability). Isis buffering overhead is low, because once a message becomes stable (which takes a few milliseconds) the sender piggybacks this information on some other outgoing message, or sends a special acknowledgment message that triggers garbage collection. In tests of Isis under heavy load, we rarely see more than 5 or so messages buffered per sender in a group, so a group with 3 senders would have a buffering overhead of about 15 messages, on the average, in each member. Notice that this number is linear in the size of the group, and governed by small constants. If the buffered message queue ever grows large, a process can always empty the queue by simply transmitting any unstable messages to remaining destinations. We do have a mechanism for this purpose; it gets used mostly if a process fails and a delay occurs before other processes detect that this has occurred.

Thus, Cheriton and Skeen are correct when they say that the number of messages a process may need to buffer is linear in the number of processes sending to a participant in the system, but they are incorrect when they claim that this implies a large overhead. Because a message is buffered for a limited period of time, to buffer an unlimited number of messages, a process would need to receive an unlimited number of messages during the retention time period. In practice, once a process reaches some maximum rate of incoming messages, the senders choke back. At this maximum rate of incoming messages, the average time to stabilization determines the amount of information that will be buffered. Moreover, the Isis compensation mechanisms (primarily, flushing unstable messages by transmitting them to remaining destinations) offer a simple way for a process to reduce the level of resources used for buffering.

Now, buffering is concerned with *atomicity*. The other form of overhead that Cheriton and Skeen discuss is the information needed to represent the causal relationships between messages. They claim this is a "graph" or 2-dimensional matrix, but because Isis only worries about causality for asynchronous multicasts (since one delays any multicast initiated while an unstable point-to-point message is pending), the matrix collapses into a vector timestamp, with one entry (an integer) per sender within a group. Moreover, the vector need not represent causal information for a process that is not sending within a group, or that has not done so recently. For these reasons, vector timestamp storage is not an important source of overhead in Isis: the number of senders will never be huge, and in most real systems one is discussing a vector that would have 2 or 3 entries, each integers! Stephenson has discussed additional optimizations based on detection of bursty communication that reduce this further, so that the average message may contain no overhead beyond that used to ensure point-to-point FIFO communication [Ste91].

Knowledgeable readers will be aware that Isis has experimented with multiple vector-timestamp causality representations, to avoid a type of delay seen in what we call the "conservative" scheme for enforcing inter-group causality. In the conservative scheme, a sender cannot switch groups within which it is sending until prior multicasts have become stable, which may take several milliseconds. The alternative is to put multiple vector-timestamps on messages, using a garbage collection algorithm to avoid this growing to a significant overhead. At present, we use the conservative scheme, and all I wish to say about this fancier approach is that with timestamp compression and other heuristics, overhead should also be very low.

So, where Cheriton and Skeen "prove" that quadratic overhead will be necessary, experience – and sound engineering – reduces this to a constant message buffering overhead (for atomicity purposes), and an infrequent vector timestamp containing a few integers. A far cry from the performance disaster that Cheriton and Skeen envision! CATOCS is really no more expensive than, say, TCP in a point-to-point case.

In passing, I should mention that this analogy with TCP is not so far fetched. When we compare Isis with TCP, our performance is very close for all but the smallest messages, and the overhead on a typical

cbcast is not much higher (either in bytes or compute time) than for a typical TCP protocol [BSS91].

4.3 Transactional issues

Cheriton and Skeen portray a world in which there is a black and white choice between a sort of loosely coupled, reliable but unordered data publishing mechanism at one extreme, and transactions at the other. But the Isis virtual synchrony model sits in between, offering a way to balance the need for a simple distributed programming model with the costs of reliability.

This is not an appropriate setting to try and relate virtual synchrony to the transactional model – I have done this elsewhere, and interested readers may want to refer to [Bir94]. Basically, virtual synchrony is a modification of the transactional model, lacking an explicit data model or groups of operations, but including a notion of sequences of operations (causal sequences), as well as atomicity mechanisms. The model works well for the sorts of things we do in Isis, like replicating data within process groups, and triggering actions when group members fail. The model is theoretically sound, and one can show in a rigorous way that it guarantees a strong form of application-level consistency, much as for a transactional system. However, where transactions focus on independently developed and independently executing programs, and on persistent data, virtual synchrony is matched to the needs of closely cooperating programs, organized into process groups, and replicating state and control information, and focused on making online progress within the primary partition of a local area network.

Cheriton and Skeen authors assert that *"CATOCS as a communication facility is limited to ensuring communication-level semantics..."*. True, ordering properties alone are inadequate to provide higher level consistency. But the Isis computing model (within which the CATOCS ordering properties are just one aspect) provides effective support for consistency of replicated data structures, shared by groups of processes, and updated asynchronously. Cheriton and Skeen would have us believe that only transactions can permit users to define distributed data structures and objects that will remain consistent despite failures. But on the contrary, Isis is powerful precisely because it is one of the few distributed technologies to be cheaper than transactions and yet to support a consistency model that is *as powerful as serializability* except in regard to durability constraints on externally stored data. An Isis process group is fully capable of replicating any data structure that its members are able to compute, and will provide extremely strong guarantees of consistency, and of forward progress. Thus by eliminating mention of the virtual synchrony model that Isis embodies, the authors arrive at a conclusion that is not applicable to Isis, or to other systems like Transis or Psync, which use similar execution models.

When we have applied virtual synchrony to implementing transactional systems, the focus has been on transactional replicated data. And we believe the results to be very promising. Our algorithms do not experience broken read-locks and hence can avoid unilateral aborts due to partial failures. These protocols send updates asynchronously, do local reads and read-locks, and guarantee serializability. Here, the use of CATOCS ordering is *internal* to a transaction, not *between* transactions. Thus, for this work, Cheriton and Skeen's claim that only the serialization order matters is actually not correct: internal event orders matter too, for example if one transactions triggers multiple operations on the same data.

Transactions have often been viewed as the magic wand that makes all fault-tolerance problems vanish – especially by those who don't actually build fault-tolerant systems for their living. Cheriton and Skeen seem to cast them in this light, first arguing that distributed systems need transactional constructs, and then arguing that since Isis is not oriented towards transactions, Isis won't support transactional applications very well. They are wrong on both counts. For control/communication applications, roll-forward progress is vital: one cannot tolerate indefinite delays waiting for a system component to recover, and one is willing to sacrifice some degree of external consistency for a sufficient degree of internal consistency to allow safe progress. "Durability" is needed for transactions but not needed in these sorts of roll-forward availability problems. Isis provides internal consistency, within a primary partition, and this is precisely what these sorts of systems need. And not only is the cost far lower than in a transactional approach, the resulting systems are not prone to blocking in simple failure scenarios.

This insight is missed by Cheriton and Skeen. There are important settings, like air-traffic control systems and factory floor automation, where *keeping the system running* is much more important than external consistency. Sure, a traditional database application is best treated in a transactional model. But

it is clear, after 20 years of database research, that this model is not the right one for high availability online control applications, or for the sorts of distributed services that one uses in a general purpose operating system, or for conventional O/S programming. And, there is an issue of programming model: how many people in the SOSP community are ready to start programming in SQL?

Despite the claims Cheriton and Skeen make to the contrary, Isis is a perfectly reasonable technology to use side-by-side with transactional systems, even if it is not a complete solution to such problems. Isis has often been used by developers who are implementing replicated or wide-area transactional systems, or who need a data distribution technology as an adjunct to a database technology from some other source. Transactions are slower and more synchronous than virtually synchronous communication in Isis, but if the application needs what transactions offer, the need for what Isis offers may be, if anything, enlarged.

4.4 Real-time issues

Real-time has very little to do with how fast a system can be made to run: most existing real-time systems are more concerned with how long to delay the delivery of a message so that it can be processed simultaneously by all members of a group. That is, real-time systems generally work by slowing things down to accommodate the slowest, balkiest operational process.

On the other hand, many users who characterize their application as having a real-time requirement actually mean that it has a requirement for sustained communication rates or limits on average latency and guarantees of average throughput. Isis works well for this class of application, as long as the desired limits are far enough from the performance envelope in which Isis operates. So, if real-time means fast, as opposed to deadline-driven, systems like Isis that support CATOCS order can certainly be useful, and indeed may even be the technology of choice.

Is CATOCS ordering at odds with real-time? When real-time means constraints that are small multiples of message latency times, for example millisecond reaction times in a fly-by-wire aircraft, perhaps so. Such systems are heavily scheduled and the integration of schedulers with CATOCS communication properties is an untested concept. But once constraints push up to the 100ms range, such as one sees inside a telecommunications switch, CATOCS ordering delays are small compared to scheduling objectives. Isis can and has been used in settings with goals in this range. And I would hazard a guess that most real-time systems have constraints even weaker than this.

5 Summary and conclusions

To summarize, Cheriton and Skeen have shared their thoughts on the usefulness of causal and total communication ordering. They view the issues from a narrow perspective and from experience with systems lacking the group programming model, fault-tolerance goals, and programming tools supported by Isis. And in this limited context, they reach the conclusion that these ordering properties – indeed about almost any type of "property" that a system might support – are undesirable. Perhaps they are correct insofar as their own technology base (V and TIB) is concerned, but the paper is positioned as a critique of my work on Isis, and yet offers arguments that are largely unfounded or trivial in regard to the actual Isis system.

The reality of the situation is that Isis enables a style of distributed computing that can't easily be undertaken using Cheriton's V system or Skeen's TIB product, or with the transactional technologies they apparently favor. This style is important in a widely varied, challenging, significant range of applications. And for these applications, the ordering properties of Isis are important. They bring fault-tolerance, consistency, a simplified programming model, and performance benefits. Perhaps the introduction of causal and total order to TIB would not in and of itself, confer these benefits on TIB. But within the context in which we work on Isis and Horus, it would be hard to do without them.

Cheriton and Skeen tell us how not to build reliable distributed systems, namely the way that I have proposed. But they seem not to have a positive alternative to offer – at least not for the settings in which Isis is used. So, let me end with a challenge: now that you've told us how not to do it, why not advance a concrete, positive, proposal, backed with some experimental data?

Acknowlegements

A great many of my colleagues have made suggestions and comments on which I drew in writing this rebuttal. Rather than enumerate them by name, let me just thank all of these individuals for their help, which I have found extremely valuable.

References

[ADKM91] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Transis: A communication subsystem for high availability. Technical Report TR 91-13, Computer Science Department, The Hebrew University of Jerusalem, November 1991.

[AMMS+93] Yair Amir, Louise E. Moser, P. M. Melliar-Smith, Deb A. Agarwal, and Paul Ciarfella. Fast message ordering and membership using a logical token-passing ring. In *Proceedings of the 13th International IEEE Conference on Distributed Computing Systems*, pages 551–560, Pittsburgh, PA, May 1993.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12), December 1993. To appear.

[Bir94] Kenneth P. Birman. Maintaining consistency in distributed systems. *Journal of Parallel and Distributed Computing*, 1994. Accepted for publication.

[BJ87a] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 123–138, Austin, Texas, November 1987. ACM SIGOPS.

[BJ87b] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[BR93] Kenneth P. Birman and Robbert Van Renesse. *Reliable Distributed Computing Using The Isis Toolkit.* IEEE-Press, 1993.

[BSS91] Kenneth Birman, Andre Schiper, and Patrick Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3), August 1991.

[CZ85] David Cheriton and Willy Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[JB89] Thomas Joseph and Kenneth Birman. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Transactions on Computer Systems*, 4(1):54–70, February 1989.

[KTHB89] M. Frans Kaashoek, Andrew S. Tanenbaum, Susan Flynn Hummel, and Henri E. Bal. An efficient reliable broadcast protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[LLS90] Rivka Ladin, Barbara Liskov, and Liuba Shrira. Lazy replication: Exploting the semantics of distributed services. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 43–58, Qeubec City, Quebec, August 1990. ACM SIGOPS-SIGACT.

[MSMA90] P. M. Melliar-Smith, Louise E. Moser, and Vivek Agrawala. Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):17–25, January 1990.

[PBS89] Larry L. Peterson, Nick C. Bucholz, and Richard Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[Pow91] D. Powell. *Delta-4: A generic architecture for dependable distributed computing.* Springer-Verlag ESPRIT Research Reports, 1991.

[Sch86] Fred B. Schneider. The state machine approach: a tutorial. Technical Report TR 86-800, Department of Computer Science, Cornell University, December 1986. Revised June 1987.

[Ste91] Pat Stephenson. *Fast Causal Multicast.* PhD thesis, Cornell University, February 1991.

Statement of Ownership, Management and Circulation
(Required by 39 U.S.C. 3685)

1A. Title of Publication	1B. PUBLICATION NO.	2. Date of Filing
Operating Systems Review	0 1 0 - 0 7 6	10/1/93
3. Frequency of Issue: Quarterly	3A. No. of Issues Published Annually: 4	3B. Annual Subscription Price: $15 Members, $41 Nonmembers

4. Complete Mailing Address of Known Office of Publication *(Street, City, County, State and ZIP+4 Code) (Not printers)*
Association for Computing Machinery, Inc. 1515 Broadway, NY NY 10036

5. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher *(Not printer)*
Association for Computing Machinery, Inc. 1515 Broadway, NY, NY 10036

6. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor *(This item MUST NOT be blank)*

Publisher *(Name and Complete Mailing Address)*
Association for Computing Machinery, Inc. 1515 Broadway, NY NY 10036

Editor *(Name and Complete Mailing Address)*
William Waite, U of Colorado, Dept of Elect. & Comp. Eng. 425 Boulder COLO 80309-0425

Managing Editor *(Name and Complete Mailing Address)*
Janet Benton, ACM, 1515 Broadway, NY NY 10036

7. Owner *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Item must be completed.)*

Full Name	Complete Mailing Address
Association for Computing Machinery, Inc (a non profit organization)	1515 Broadway NY NY 10036

8. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities *(If there are none, so state)*

Full Name	Complete Mailing Address
none	

9. For Completion by Nonprofit Organizations Authorized To Mail at Special Rates *(DMM Section 424.12 only)*
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes *(Check one)*

(1) [X] Has Not Changed During Preceding 12 Months
(2) [ ] Has Changed During Preceding 12 Months *(If changed, publisher must submit explanation of change with this statement.)*

10. Extent and Nature of Circulation *(See instructions on reverse side)*	Average No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
A. Total No. Copies *(Net Press Run)*	6838	6647
B. Paid and/or Requested Circulation 1. Sales through dealers and carriers, street vendors and counter sales	-----	-----
2. Mail Subscription *(Paid and/or requested)*	6761	6567
C. Total Paid and/or Requested Circulation *(Sum of 10B1 and 10B2)*	6761	6567
D. Free Distribution by Mail, Carrier or Other Means Samples, Complimentary, and Other Free Copies	10	10
E. Total Distribution *(Sum of C and D)*	6761	6567
F. Copies Not Distributed (1. Office use, left over, unaccounted, spoiled after printing)	67	70
2. Return from News Agents	-----	------
G. TOTAL *(Sum of E, F1 and 2—should equal net press run shown in A)*	6838	6647

11. I certify that the statements made by me above are correct and complete

Signature and Title of Editor, Publisher, Business Manager, or Owner
Janet D. Benton Deputy/Director Publisher

PS Form 3526, January 1991 *(See instructions on reverse)*