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

Ken Birman

October 15, 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. I find their paper highly critical of Isis, and unfairly so, for a number of reasons. In this paper I present some responses to their criticism, and also explain why I find their discussion of causal and total communication ordering to be distorted and incomplete.

1 Background

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 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

*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.

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:

- Although presented in academic terms, both Skeen and Cheriton have a commercial interest in a system called TIB(tm)[2], which competes commercially with the Isis Distributed Toolkit(tm) and the Isis Distributed News system(tm). Thus, they have a commercial product in the back of their mind, and so do I. I comment on this not as a criticism of their paper, but rather in the sense of disclosing a mutual conflict of interest here. Cheriton and Skeen stand to make money using this argument, by disparaging features of the Isis product, just as I stand to profit by arguing the usefulness of these features. Does this lead any of us to distort our arguments? Well, this is certainly the risk where a conflict of interest is present; I'll leave it to the academic community to reach its own conclusions.

- It is one thing to criticize my work based on the arguments I have made in the past, and quite another to misrepresent those arguments, and then to attack the resulting incomplete and distorted versions. I want to suggest that Cheriton and Skeen repeatedly create scenarios that overlook the virtual synchrony aspects of Isis, arguing that they are interested only in the value of CATOCS ordering. But then they point to aspects of the problem that involve a mixture of ordering and reliability issues, and conclude that CATOCS is inadequate. In particular, they introduce transactions in a way that suggests strongly that CATOCS was intended primarily to support the construction of transactional applications.

- Even if one takes the Cheriton and Skeen paper on its own ground, the arguments presented are often incomplete and hence misleading. For example, when these authors do introduce pure ordering issues, they do so through application-specific "semantic" ordering properties. One is lead to believe that the typical user has some set of ordering goals in mind, and that the purpose of the system is primarily to implement the user's programming model. This overlooks the fact that ordering issues most often arise through programming styles that involve asynchronous communication, and that the major reason for a system to worry about message ordering is to reduce user-visible design complexity and thereby increase reliability.

- Several of the arguments they make most forcefully, such as the ones about CATOCS being unable to maintain consistent "resilient" data structures, and the claims about quadratic growth in buffering space as a function of system size, are simply wrong. In this specific case, Cheriton and Skeen's quadratic growth phenomenon corresponds to

[2]TIB, or Teknekron Information Bus, is trademarked by Teknekron Software Systems Inc., which also holds a trademark on the phrase "subject based addressing". TIB is the communication subsystem used in the majority of Teknekron software installations and applications; Dale Skeen was a developer of this system, and Dave Cheriton has consulted for him, and continues to do so. The Isis Distributed Toolkit and Isis Distributed News are trademarks of Isis Distributed Systems Inc; I played a major role in developing these commercial products.

a few bytes of timestamps on the typical Isis message, averaging perhaps 12 bytes of data even for very large process groups. Overhead within the participating processes is similarly low. There is no linkage to the size of the system. This overhead analysis, and several others, is directly contradicted by experience in systems like Isis, Transis, Psync, Gossip and Amoeba.

- In the operating systems community, performance is normally considered a significant issue. Yet while Cheriton and Skeen claim that CATOCS will be costly, they say nothing about the cost of the transactional consistency model they favor. Where consistency is required, the Isis virtual synchrony model – of which they omit all mention – considerably outperforms transactions and gives strong guarantees except in regard to external data. Moreover, one can easily layer transactions over Isis, or put them next to Isis. Thus, one has three choices: a programming model that seeks to run as fast as possible, but with no semantics (and with the hope that the user won't have to build something costly to introduce reliability or consistency), a process group programming model in which one can replicate data and have other sorts of consistent, fault-tolerant programming tools, and in which the CATOCS properties are a key to extremely high performance, and a transactional model, in which performance is inevitably limited by the cost of flushing a log to stable storage. By omitting reference to virtual synchrony, Cheriton and Skeen bias their argument in favor of the conclusion they want to reach.

It is no secret that I believe the arguments in favor of providing causal and total ordering, at least in the context of process-group computing systems, to be compelling. One of the main reasons, which Cheriton and Skeen do not address, is that this approach (at least when embedded into a suitable programming environment) greatly reduces the complexity, development time, and debugging effort needed when implementing distributed systems. And I believe that even if a small performance loss is sometimes incurred, the average user is more concerned with having a simple way to build reliable software than with having total control of the properties of the communication protocols used in an application. Cheriton and Skeen overlook the benefits of simplicity, standardization, and of taking hard problems out of the hands of the developers.

Nor is it a secret that this view is disputed. Yet in challenging it, Cheriton and Skeen dredge up a contrived collection of systems that don't need what Isis offers (they claim that their examples are drawn from the literature, but most are quite different from examples appearing in papers of which I am aware). They manage to miss the entire class of systems for which Isis was developed. To the degree that they do introduce reliability issues, they do so without addressing key aspects of the Isis virtual synchrony model, leading them to a conclusion that "CATOCS ordering properties are inadequate to address these issues." Yet the Isis approach brings more than CATOCS ordering properties to bear on problems of reliability, a point that Cheriton and Skeen completely overlook. A good example of this is the issue of failures leaving gaps in the causal past due to inconsistent outcomes in atomic multicast protocols. Isis prevents these, while they would be extremely difficult to detect or overcome in the approach Cheriton and Skeen favor.

In reducing arguments in favor of Isis to arguments in favor of CATOCS, Cheriton and

Skeen simply miss the whole point, which is that when combined with an appropriate execution model, CATOCS is a valuable tool for improving performance in distributed systems and simplifying the user's programming environment. Perhaps CATOCS ordering would be of limited value in the inconsistent world of RPC programming, or in a purely transactional environment. But in between these extremes lies a rich distributed computing model in which high availability, high performance and consistency combine to offer the developer options that are not available in any other setting.

2 Real-world experience

Cheriton and Skeen position their paper primarily on academic territory, and I want to respond in kind. Yet the authors do allude to the commercial success of TIB, the Teknekron product, which, we are told, has been installed on "more than 150 trading-room floors." Several parts of the paper cite feedback from TIB users. So I want to point out that Isis is also a commercial success, and that I could cite the same sorts of feedback from my users. As a matter of fact, I think I could make a strong argument that Isis is more successful than TIB at the time of this writing.

Isis is used in market-data applications, and has been picked for several ambitious financial systems, such as the internal communication infrastructure of the New York Stock Exchange, and for the communication layer of EBS, an electronic trading system being developed by a consortium of more than 60 Swiss banks and brokerages (ATB). Isis has also had success in other settings. For example, Isis is used in telecommunications systems, factory-floor automation systems (both for process-control and electronic manufacturing), computer-aided design and engineering, scientific computing (the data acquisition system in the new CERN particle accelerator) – and this is just the surface of a varied collection of demanding distributed systems. Moreover, Isis has been selected for major military projects, such as the HiperD system (a successor to the AEGIS system), for next-generation worldwide cellular phone systems (Iridium), and for use in air-traffic control settings. These are all examples of the sorts of challenging, next-generation projects that demand innovative systems technology, and the decision of groups like these to use Isis is not taken lightly.

Presumably, Cheriton and Skeen would react by saying that this sort of list of users is no proof that these applications actually need CATOCS ordering; perhaps they need fault-tolerance or some other aspect of Isis, perhaps the Isis pricing appealed to them, or perhaps these users were actually mistaken and could have managed well with TIB. Of course, the same comment can be applied to the 150 happy TIB users to which Cheriton and Skeen point.

It seems to me that the right way to cite a real-world application is to describe that application in enough detail so that the reader can see clearly how some system property, such as CATOCS ordering, enters into solving it. For example, the New York Stock Exchange worries that overhead displays should preserve the causal order in which trades occurred; otherwise, traders might fail to recognize a related series of events. The exchange also worries that displays show the same events in the same order, to avoid giving one trader an unfair advantage over another who happens to watch a different display. And, there are loose real-time constraints in a stock exchange application. By studying these sorts of considerations, one can ask how valuable it is for the underlying message transport system to provide causal and total order.

Presumably, Cheriton and Skeen would respond that this is an unusual application.

In an air traffic control application, one is faced with an elegant demonstration of the role of consistency in a fault-tolerant system. Such a system will generally use a small cluster of workstations to provide fault-tolerance to a "position" at which a flight controller works with the aid of specialists who help prepare data for control actions the controller will need to take in the future. Consistency, replication, and performance all come together in such an application: a workstation can only take over from another, failed, workstation if the former system has an accurate copy of the flight data the latter was maintaining. Again, one can reduce such problems to mixtures of CATOCS ordering and reliability considerations – to the virtual synchrony model. But it would take a great many pages to describe this application in enough detail to show exactly where these issues arise. Even then, the reader would have no way to know whether this is the only way to solve such a problem.

No doubt, Cheriton and Skeen would respond that this is an unusual application, too. I expect that they would respond this way to any application I could present. Basically, they argue that a CATOCS ordering property is generally not the best way to solve a given problem, and just as I question their examples below, I am sure that they would question any I might put forward. Indeed, for any given application, short of comparing two approaches put forward by proponents of the respective technologies, I can't imagine how one could answer such a question convincingly. Cheriton and Skeen try and show how I would solve some contrived problems they come up with, but they never asked my input on this. And their solutions are not the ones that come to mind, based on my experience with Isis.

In applying Isis to real problems, CATOCS properties are often valuable because they simplify asynchronous communication patterns. If Cheriton and Skeen have an alternative approach to simplifying such problems, I would be very interested to see it. In their paper, however, the conclusions are purely negative and no real alternative is proposed, except of course TIB – for those readers aware that TIB is what the 150 users they cite are running.

3 Software Issues

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

- *Reliability and fault-tolerance.*
- *Consistency.*
- *Concurrency.*
- *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 write off any data managed by a failed server as a lost cause: 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 banks 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.

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 recent 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.

Now, Isis basically 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.

As I read their paper, Cheriton and Skeen confine themselves to examples of applications that need weakly consistent data distribution, transactional databases, and real-time applications to criticize ordering properties that stem from the control and consistency problems towards which Isis is biased. They argue that only transactions offer consistent failure recovery to the user. The virtual synchrony model offers an alternative to transactions that costs less, and achieves sufficiently strong guarantees to support reliable distributed control/communication systems, but Cheriton and Skeen never even mention this – although it is the "main" model used in Isis and other similar systems. Thus, they hand the reader a biased and incomplete picture of the technology, and then draw conclusions which many of us set aside years ago.

4 Process (and subject) groups

Where did this whole dispute originate? The story goes back a few years.

In 1985 Dave Cheriton and Willy Zwaenepoel suggested that 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. And V was not fault-tolerant.

Isis picked the process group idea up, 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.

To summarize, 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. In our work, the CATOCS ordering issues enter only at this lowest layer of the system: they do not stand by themselves.

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, one can understand the Cheriton and Skeen paper as a defense of a design decision made in V, and carried over into TIB – one in which guaranteed ordering and reliability are viewed as too costly to support.

5 The arguments

Let's summarize the CATOCS arguments as advanced by Cheriton and Skeen:

- *Causal ordering is either too ordered or not ordered enough, (but it definitely isn't what you usually want).*
- *... anyhow, its hard to find good examples that need causal order.*
- *Total order is much too expensive.*
- *... and in settings where Isis claims you want strong reliability properties, one would really need to group operations into serializable transactions, and CATOCS order doesn't support these.*
- *... nor do ordered communication protocols scale well or deal with real-time constraints.*

5.1 Causal delivery order

I wonder if Tommy Joseph and I shouldn't have used some other term, like "multi-sender streams" to introduce the idea of causal message delivery back in 1987. But, after all, causality wasn't our idea: Leslie Lamport used this term earlier, and he was merely picking up familiar language of quantum mechanics and relativity theory from physics! Perhaps we have Einstein to blame for this unfortunate terminology, although he would probably point to the ancient Greeks...

There is a sense in which fear of causality has gripped sections of the distributed computing community for much of the last few years. Yet, the idea isn't much different than

FIFO ordering in a stream. Basically, a stream (a TCP channel or a pipe) delivers messages in the order they were sent. This is often useful.

Let me give an example that arises in many graphics applications, such as the air-traffic control displays mentioned earlier. For simplicity, one can think of such a system as a version of the X11 display server, specialized to the air-traffic display, but performing a role similar to that of X11. Much as with X11, one can send a message to position the cursor and then a message to display a string; the string comes out at the place where the cursor was located. Notice that such sequences depend on the FIFO-ness of the stream connecting the application to the display server: if the string were displayed before the cursor was moved, errors would result. So, such a system needs a FIFO connection between the application program and the display server.

But suppose that two programs cooperate; call them P and Q. P wants to position the cursor and then it will issue a message causing Q to render an object for display on the screen (this is a common application structure – such behavior arises in a number of Isis applications of which I am aware). Even if the channels from P and Q to the display server are FIFO, they can lose the ordering of these requests because the two messages are sent on different channels. So, using a FIFO ordered communication layer, P would have to wait for acknowledgement that the cursor had been moved before sending the rendering request to Q. This delay may make the application substantially slower, because Q won't start computing the rendered object as soon as if P has not been forced to wait in this manner.

Causal delivery ordering just extends the idea of a FIFO order to cases like this one, in which a "sender" can be a single computational thread that spans more than one process. In our example, because P asked Q to do a display operation after positioning the cursor, the messages from P to the display and from Q to the display are said to be causally ordered: in a causal sense, the cursor positioning request came before the display update. Causal ordering is basically the requirement that the communication subsystem deliver messages to a destination in a way that respects the causal order in which they were transmitted. So, P can send its message asynchronously without worrying that Q's rendering will come out at the wrong place on the screen.

As long as the communication subsystem is causal – for example if P, Q and the air-traffic display server are implemented using Isis – P and Q can communicate asynchronously, improving performance. The ordering enforced in the communication subsystem is strong enough to ensure that the display will be correct. Moreover, this is true without the display server needing to "know" that it will be used asynchronously – which is likely to be important, since the display server will probably be the first part of the system to be implemented.

This example illustrates several reasons that we find it useful to support causal message ordering in the Isis communication system:

- *This ordering property makes it possible for programs like P and Q to communicate asynchronously* to a display server or to a process group, improving performance without compromising correctness. Moreover, the developer of P doesn't need to know how Q and the display server were coded to be sure of correct behavior. Obviously, any approach in which one has to rewrite pieces of the display server to use it safely would be suspect. Knowledge that the communication subsystem has a certain property eliminates the need to enforce that property in the application, making it possible to communicate asynchronously with a pre-built application which cannot be modified.

- *The causal delivery property is cheap to support:* In terms of the number of messages transmitted, the cost is the same as for a FIFO delivery ordering [Sch88]. In fact, given an implementation of a pipe or stream, the same code can support a causal message delivery ordering by including additional information in the headers on the messages transmitted. No additional messages need be transmitted, and with appropriate compression algorithms, the message headers used need not be much bigger than the ones needed to ensure FIFO delivery. Our work has shown that Isis technology scales as well as any other, and we have worked with applications that include thousands of processes on hundreds of workstations in each of dozens of local area networks, linked by wide-area lines (we use a different protocol over the wide-area links, but the Isis user sees the same API).

- *Causal ordering protects against some serious failure behaviors.* For example, the usual definition of multicast atomicity is that a message will reach all its destinations, or none. (Transactional atomicity has a similar definition.) Suppose that m_1 is sent by P to the display server and then m_2 is sent by Q, and P exits or crashes after sending m_1. Would we be happy with an outcome in which m_1 is not delivered, but m_2 gets through? I think not, but without having the communication system track causal relationships, one cannot prevent this outcome from occurring. Cheriton and Skeen argue that if an application wants an ordering property, the developer should implement it. But, how would a program that receives a copy of m_2 recognize that it is not merely "early", but is actually an orphan? Systems like Isis include protocols to overcome such problems. In particular, the causally ordered communication primitive used in Isis hides m_2 from the application, delivering m_2 after m_1 is received, and discarding m_2 it m_1 is lost forever.

  If one simply hands these problems to the user, along with prebuilt display servers, I think the most likely result will be bugs – provoked by gaps in the causal past after crashes as well as by out-of-order communication, coupled with synchronous (hence, inefficient) patterns of communication. Isis users do not have to worry about this complexity, and yet can benefit from high performance.

Now, Cheriton and Skeen attack all this with a series of observations that echo comments made by Leslie Lamport in his original 1978 paper about time in distributed systems.

- *Causal ordering may be specious:* perhaps the application didn't really care about the ordering in the first place. That is, many "potentially causal" orderings will be unimportant to the application. Delaying messages to order them according to spurious orderings can only hurt performance.

  It seems to me that this can be most easily understood by again thinking about a TCP connection or a pipe. A similar argument applies: a pipe might not need to order all the messages one sends down it. Perhaps an application sends some messages without caring about the order. Performance is lost because the pipe orders these messages even when the application didn't need this property. But we tolerate this miniscule loss of performance because an unordered pipe would introduce more complexity in the application program than it would remove from the communication system. Similarly, causal ordering will rarely delay a message; it is more of a protection against race conditions. Given that the cost is essentially the same as for a FIFO ordering, why shouldn't we simplify life for the application developer?

This raises an interesting point. Cheriton and Skeen appear to be arguing in part for an interface in which the user would specify the messages on which an outgoing message depends, explicitly, for example by providing a list of prior messages. First, there is nothing in the CATOCS model that precludes such an interface. In fact, early Isis work included a labeling scheme for this purpose, and Psync offers very clean interfaces with this functionality. But our experience with real applications lead us to believe that control over fine-grained message ordering properties offered little potential performance benefit. Flow control and congestion handling are far more important in the systems we have built using Isis, and in the few instances where a performance problem was traced to undesired causal ordering, one can always disable the ordering property causing the problem.

To summarize our point, the overhead associated with causal ordering is rarely as important to performance as the delays introduced by flow control and memory/resource management mechanisms. Either a system is lightly loaded – but then causal delays are very unlikely to occur, at all, or heavily loaded, in which case causal delays are dwarfed by flow control delays. Either way, causal ordering is the least of the developer's problems.

- *Causal ordering may fail to capture ordering relationships* external to the system, or application-specific ordering properties. These two points seem minor: how many systems have external communication paths of the sort described here? Most communication in most computer systems is via internal communication paths under control of system software. So, while external communication can violate causal order, it also seems true that this is unlikely. Moreover, since causal time can be adjusted with real-time to order apparently concurrent events, in a system that does have this problem it can be fixed (up to the resolution of the clock synchronization algorithm).

  I would suggest that external ordering paths arise primarily in shared memory applications. Isis applications that use shared memory commonly associate such memory with a process group. Event-oriented communication occurs through messages to the group, while the shared memory is used for "non-event oriented" operations, such as representation of a display. The fact that Isis causal properties are only guaranteed within the event-driven side of the application is not a problem here, ecause causal ordering through the shared memory was either not needed. The type of ordering that *is* needed is normally obtained through explicit shared memory mutex mechanisms. Of course, one can build a causal shared memory (see, for example, the recent work by Neiger and Hutto [ABHN91]) – but in the applications we have considered, such measures have not been necessary.

  As for application-specific ordering, or other semantic knowledge, it is true that Isis cannot guess what the programmer had in mind. But this also seems like an unavoidable limitation of computing: one could make the same point about an optimizing compiler. Cheriton and Skeen propose a particularly uninspired fire alarm design, but one would hope that the person charged with actually building such an application would think to include location and time information on alarm messages! More broadly, it is true that there are many issues concerning proper application design methodology which causal message ordering won't address. But it may help, it isn't likely to hurt, and it has other types of positive benefits.

- *They think it is costly and doesn't scale well.* The discussion in this part of the Cheriton and Skeen paper is incorrect. They clearly don't understand the Isis protocols at all, making it unreasonably easy to respond to the claims they level against the system. The bottom line is that Isis puts relatively small headers on the average message, and this is enough to let it enforce causal ordering. Moreover, Isis rarely buffers more than a few messages in order to guarantee atomicity: there are simple ways to avoid the accumulation of substantial backlogs, and Isis employs a number of these. Sometimes Isis delays, waiting for messages to stabilize, to avoid having to put large headers on messages, but this is rare.

  In Isis, scaling has not been a problem: we see process groups with 20 or more senders (although smaller numbers, like 1 or 2, are much more common) and as many as hundreds of "clients". By client, we mean processes that are either passive receivers, or that multicast only to the subgroup of senders, not to the full group including other clients. IP multicast support is very helpful when multicasting to large groups, and is now used in Isis; a forthcoming paper will describe our approach, which comes close to the performance of unreliable data transport technologies and of the lower layer transport protocols. For much larger systems, or wide-area networks, Isis can also be used through a hierarchically structured software layer that scales to large numbers of local area networks – much like the TIB software, which has a similar gateway technology. Isis has been used in extremely large networks, and no fundamental limit has yet been encountered. I believe that we will see Isis applications with ten's of thousands of participating programs within a year or two. Moreover, recent versions of Isis perform quite well by comparison with any alternative, including RPC-style technologies, TCP, and even V or TIB. When asked to do the same thing, on the same platform, Isis generally performs at least as well. For communication in large groups, we are much faster than what TCP or RPC would permit for a similar communication pattern.

  All of this may sound suspicious in light of the Cheriton/Skeen claims about quadratic buffering requirements: is Isis sweeping some massive overhead under the rug? On the contrary, it isn't hard to see that that the Cheriton and Skeen proof in this section of the paper is wrong. In other fields of science, a theory needs to be confirmed by experiment. Cheriton and Skeen, in their past work, have generally been careful to root their arguments in sound experimental work. This time they fail to do so, and arrive at incorrect conclusions.

  What, specifically, is wrong with their argument? 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 in three respects. First, in any large system processes will make heavy use of multicast to one another. Secondly, asynchronous point-to-point communication introduces two issues that most systems prefer not to treat: 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), and the need to maintain a complex causal ordering structure. Consequently, most real systems – and specifically, Isis – treat point-to-point communication using synchronous RPC-style protocols, which separate the point-to-point communication burden from other communication.[3] In Isis, no multicast is ever sent when an RPC is unstable (although point-to-point messages need to carry CATOCS information about prior potentially unstable multicasts). 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. Thirdly, 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!

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.

Thus, a real system, such as Isis, deals only with CATOCS ordering primarily for asynchronous multicasts – not point-to-point messages – and the actual rate of multicasts is limited by bandwidth and flow control considerations.

[3]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.

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.

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.

- *They don't like the "news" example.* Robbert Van Renesse and others have used misordering of postings to USENET News (a common problem) to motivate having a system enforce causal delivery ordering. Cheriton and Skeen dispute the value of this type of order and (in the context of a single news group) suggest a cheap way to implement such order, if desired.

  What I find odd about this is that news-style communication is actually quite a good illustration of how causal ordering can be useful. But the way that Cheriton and Skeen present the example, they obscure the main point.

  I have always understood this example to concern the case where *programs* are communicating through a network news model – not of people using network bulletin boards. Also, the problem usually assumes that a program might subscribe to multiple news subjects.

  Take the example that Cheriton and Skeen discuss. Program P is a source of quotes for some stock, X, and program Q is a source of theoretical pricing for X, X'. In a news setting, one might have a news group for each such stock, with both the actual and theoretical values published within it. Program Q would subscribe to many news groups, since theoretical price calculations normally are based not just on stock prices but on other information, such as market volatility, bond rates, etc.

  Cheriton and Skeen point out that that if P publishes quotes X_0 and X_1 for X, the theoretical value X'_0 might be issued concurrently with X_1. From this they conclude that causal ordering buys us little: it doesn't preclude R from receiving X_0, X_1, X'_0 and mistakenly interpreting X'_0 as the theoretical price corresponding to X_1. If this could happen, of course, absurd trading patterns might result. If R operates with "current" data, this class of bugs is precluded. They suggest that this can only be achieved by annotating a message with identifiers for the messages on which it depends.

  I have several responses to this. First, I agree that it makes sense to tag the theoretical price with a timestamp identifying the quote on which it was based. With such an approach, stale theoretical prices could be filtered out upon reception, by the application. Causal ordering ensures that a theoretical price will never arrive before the corresponding quote, but not that stale theoretical prices will never be received.

  Cheriton and Skeen's solution scales poorly if many groups are used. In their approach, a message lists the prior messages on which it depends. A recipient waits until the prior messages have arrived before processing such a dependent message. What they fail to notice is that dependencies may be transitive: if X depends on Y, it also depends on anything Y depends upon. The implication is clear: lacking some way to garbage collect dependency information, messages will grow to carry unlimited numbers of identifiers for prior messages. Moreover, in a multi-group setting, a subscriber to a single group will still need to track this information for other groups to which it is not subscribing, since a causal chain could lead back into such a group "downstream", in which case such dependency information might be needed to ensure correct ordering. Recall the Cheriton and Skeen claims that the overhead for enforcing causality will grow quadratically in the size of the system. While such growth will not occur in Isis, the scheme proposed here would indeed incur unlimited overhead in even modestly complex settings!

My colleague Robert Cooper identified yet a third way to overcome the difficulty, in which a causal ordering mechanism is tricked into solving this problem for us. Suppose that process P knows that for each quote it distributes for X, Q will issue a theoretical price shortly after. P can subscribe to the output of Q, delaying subsequent quotes for X until the theoretical price for the most recent quote has been observed (P would also monitor Q, so that if Q fails it would not wait indefinitely). Such a solution is simple to implement, and each quote will now be causally ordered after not just the prior quote, but the corresponding theoretical price. In Isis, the tool interfaces available for publishing, subscribing, and monitoring make it possible to code an algorithm like this with a few procedure calls. Since wire services don't quote any given stock more than once every few seconds or minutes, it will be rare that P would actually need to delay a posting while waiting for the computation of a theoretical price.

Where does this leave us? Clearly, the example shows that causal ordering is not a panacea, but on the other hand, the problem is certainly not hard to solve. The timestamp approach is presumably the one most people would use: it addresses the problem at hand, and is easy to understand. Cooper's solution illustrates that problems such as these sometimes *are* simplified by causally ordered communication, if one starts with the right mindset; Cheriton and Skeen's solution is illustrative of the subtle complexity one encounters when trying to solve ordering problems in an ad-hoc manner.

But this isn't the whole story. Recall that when failures occur, there is a risk that R might receive a theoretical price but *never* receive the quote on which it was based. This happens because "atomic" delivery, in the absence of causal ordering, can allow a message X' to be delivered when a message X on which it depends is lost during the crash (i.e. if both P and Q crashed at the same time). A causal atomicity mechanism, such as the one in Isis, eliminates this risk.

An Isis user recently described an application in which the value of causal order is very clear. In this application, one news group contains quoted prices for commodities used in construction projects, such as steel beams. Postings to this group occur as quotes are received on batches of steel, for which pricing changes constantly. A second group contains recommended sources for subcontracted elements of the construction project; updates will be issued to this group if, for example, a changing steel bid makes it desirable to recontract the frame of a building. Now consider an application trying to price the overall cost of a job. It sees a subcontract change order in the second of these news groups and reruns the computation for overall job cost, but using the old steel price quote. An event that should have reduced the cost of the job now seems to have *increased* the cost! In a complex situation with many news groups, one can easily imagine the chaos that would result if stale data were often combined with current data in this manner.

So, the picture is mixed. Causal order doesn't eliminate all the issues in problems such as these, but it eliminates some of them: a message will never be seen prior to a message on which it depends, and crashes cannot create causal gaps. This may not be enough to solve the entire problem (at least without tricks), but it will often simplify the programmer's task. The flip side of the coin is illustrated by Cheriton and Skeen's proposed label scheme: a solution that looks quite simple may actually be hard to generalize, may scale poorly, and demands that the user settle on a convention for representing causal dependencies, for garbage collecting this information to prevent unlimited growth, and for dealing with gaps left by failures.

We are all familiar with the ordering anomalies that can arise in email and human bulletin boards. Programs are unlikely to be as equiped to handle these as are people. So perhaps Cheriton and Skeen are right, and their colleagues don't find these things very annoying. But they and their colleagues apparently have little experience writing *programs* that would run correctly over USENET news.

To summarize, causal ordering is important for the same reasons that it is useful for pipes and streams to deliver data in the order sent, and that it is useful for memory and file system caches to provide coherency guarantees. The issue isn't merely similar, it is the same issue, arising in a different setting! Despite the hand-waving theory put forth by Cheriton and Skeen, costs associated with the approach can be controlled by sound software engineering. And, turning things around, any argument advanced against causal ordering can also be used as an argument against coherency in file system caches and against ordering in TCP channels. It isn't hard to see that non-coherent mechanisms could achieve slightly better performance without loss of correctness, for some small set of contrived cases. But just as the general user knows what to expect from a pipe or a file, so the causal ordering property seeks to offer the user "expected behavior" in an asynchronous communication environment.

5.2 What about total ordering?

Somehow, total ordering diminished in importance within this paper as it evolved through a series of drafts, and they now have very little to say about this. Yet there is a sense in which total ordering is the main point.

In Isis, we use totally ordered messages for several purposes. We use these to trigger state-machine style computations, where an incoming message causes all members of a group to make a state transition. And, we use them to update replicated data without first obtaining a lock. For example a brokerage setting, totally ordered messages might allow a collection of advisory programs to provide identical advice to brokers who need to cooperate on the trading strategy they will employ. Here, the same information (or actions) is replicated at the various trader workstations.

Interestingly, most uses of causal order are in algorithms that employ this style of totally ordered action, but where the sender of a message holds a lock. In such settings, a causally ordered multicast will "implement" a totally ordered delivery, just as a FIFO multicast would be totally ordered if a system had only one process that ever initiated multicasts [Sch88]. The reader may want to think of totally ordered delivery as the most common case. As a matter of fact, recent versions of Isis don't implement total ordering directly: the totally ordered communication primitive is based on a token-passing algorithm that itself runs over cbcast (the Isis causal multicast primitive).

Readers who find the causal ordering issue confusing may not have realized that we rarely design algorithms with causal order in mind. We more often design with total order in mind and then, by judicious use of mutual exclusion, convert the communication calls to cbcast to increase asynchrony and performance. This is why we call the model virtual synchrony: it is an approach to developing software in which one thinks about synchronous executions, but arranges to run the resulting code asynchronously.

5.3 Virtual synchrony and transactions

This leads to an important point. 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 [Bir]. 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.

This all adds up to a direct refutation of a statement made by Cheriton and Skeen. These 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.

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.

Earlier, I pointed to a basic dichotomy in the systems area between database-style applications and other types of distributed services. The database applications tend to interact indirectly, through shared data objects, and tend to be coded in isolation. Both assumptions are very visible in the transactional serializability model. Moreover, the integrity of the external data is, as we have noted above, an important goal.

Distributed systems that use a purely database model have been developed – see, for example, the IBM AS-400 series of systems, which are hugely successful. Nonetheless, most conventional systems problems have some mixture of database and non-database software in them. It is hard to imagine coding a "news" server, or an "NIS" server, or a Kerberos key server, or any of a number of other such distributed services, using the transactional model. These types of applications are much more natural in a multi-threaded model; if reliability is also needed, process groups extend that model in a natural way. What stands out is that these applications require a close degree of consistency and cooperation between members of groups of processes, and that these processes interact via message passing. Transactions get in the way of this model of computing, because they explicitly legislate against communication between uncommitted transactions.

A colleague of mine, Prof. Andre Schiper of EPFL, Lausanne (Switzerland), offers the following interpretation of the situation. He observes that the transactional model distinguishes two entities: the transactions (or processes) that execute operations, and the data accessed by the transactions. The data exists independent of the transactions and the consistency of the system is defined entirely in terms of the state of this external, persistent, database. In contrast, the virtually synchronous model only talks explicitly about processes and message-based interactions, treating data as part of the state of processes, and defining consistency in terms of the joint states of the processes that belong to the system (membership is also defined, but I won't digress into this here). Schiper points out that this difference hides something fundamental.

Consider a transaction T_1 running under the transactional model, which holds a lock on a data item D_1 and may have modified it. Some other transaction T_2 blocks trying to obtain the same lock. Now, suppose that T_1 is suspected of being faulty.

To ensure the liveness of T_2, one must somehow break the lock held by T_1, e.g. by forcing

T_1 to abort, rolling both the data T_1 may have modified back to its previous value, and undoing any other actions T_1 has taken. This makes it hard to initiate external actions from a transaction, since it may be necessary to undo them. Moreover, the need to guarantee consistency relative to the state of the data items T_1 may have changed exposes the transactional concurrency control system to indefinite blocking in some cases where T_1 fails during a commit protocol.

In contrast, the virtual synchrony model considers all data to be owned by processes and concerns itself with internal consistency among the members of a "primary partition" of the network. In this model, one can obtain replicated data, lock-based synchronization, and roll-forward fault-tolerance – without needing to roll-back after failures, and far lower risk of having to stop the whole system while waiting for a failed component to recover. (There is still a situation where one may have to block, but it only arises if more than half of the machines in the network crash simultaneously, while a two-phase commit can block if as few as two machines crash).

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. Isis provides sufficient consistency for these systems, at lower cost than if transactions where used, and without 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.

6 Real-time issues

This is the one section of the Cheriton/Skeen paper where I find some statements that I can largely accept. Isis is not suitable for very demanding real-time applications – applications where deadlines are imposed that fall within a few multiples of message latencies. Our papers and programming manuals make this clear.

However, 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, Isis is certainly useful and indeed is often the technology of choice.

We currently are involved in two collaborative efforts that seek to augment Isis with real-time functions. One activity is headed by Keith Marzullo (U.C. San Diego); Paulo Verissimo (INESC) leads the other. By real-time we mean a level of performance far more demanding than the 25Hz requirements cited by Cheriton and Skeen. At low levels of demand, and indeed even in more performance intensive settings like co-processors for telecommunications switches, Isis in its present form behaves as well as any other alternative, and we have active users who employ Isis in settings such as these. In particular, as discussed by Joseph in the paper cited by Cheriton and Skeen, many real-time systems have a mixture of demanding front-end requirements, which are strictly local to a single processor and weaker communications requirements, perhaps with a delivery latency deadline of a few hundred milliseconds. Depending on the desired failure reaction time, Isis may be entirely adequate for use in such systems, or it may be an inappropriate technology base. CATOCS properties have very little to do with the issue, since any delays associated with ordering will need to be far smaller than application-imposed deadlines for the technology to be useful at all.

7 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 a system lacking many of the group mechanisms supported by Isis. And in this limited context, they reach the conclusion that these ordering properties, and 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.

Acknowledgements

This paper emerged from discussions with a number of friends and colleagues who had read the Cheriton/Skeen position paper. I am grateful to all of those with whom I had an opportunity to discuss the issue. Andre Schiper and Alan Fekete made a number of specific suggestions about an early draft of this document, for which I am very grateful.

Appendix I: A detailed rebuttal

In this section we rebut, one by one in the order they appear in the Cheriton/Skeen paper, the claims made by regarding the CATOCS model and Isis itself.

1. *Atomic, durable message delivery [...] is not supported [...] and yet is a significant deficiency for using CATOCS for reliable data update.* One of the themes of the paper is a repeated insistence that reliability should include an external consistency property, called *durability* in Section 2. As the authors correctly note, Isis does not normally provide a durable atomic multicast primitive; that is, a message could be delivered to a process but then lost before the process can act on it, and in some failure situations a message could be delivered to a process that then fails, but not delivered to other processes that survive it. However, Dolev et. al. describe "durable" CATOCS primitives in their paper on the Transis system, which is similar to Isis in also using the virtual synchrony model, and Schiper and Sandoz have discussed support for such a primitive in Isis. Currently, Isis users needing this property would simply build a 3-phase commit protocol, using cbcast for each of the phases (waiting for all non-faulty recipients to reply in between each round of cbcasts). If an application wished to do so, such a protocol offers a chance to flush data to disk, etc. Skeen presented the rules for a 3-phase commit in his PhD thesis (there is a 2-phase version of the same protocol, but it is prone to blocking because of failures). Thus one can implement a durable multicast over what Isis offers using the same scheme as would be used in a transactional system to commit transactions.

   The real question, though, is that Cheriton and Skeen present durability as if this property is clearly needed in most applications. In fact, Isis permits the user to maintain reliable data with strong consistency properties, and without imposing the costs associated with this form of durability. The external consistency requirement that underlies durability only matters when a process will take an external action or write data to a disk, and when it is important for the remainder of the system to be consistent with this external information. In many settings, one just doesn't need this degree of reliability.

2. *... our concern is with the merits of implementing ordering relationships on delivery of messages from multiple sources within the communication system [and not with such issues as failure reporting, atomicity, and so forth].* But this claim is not accurate, since much of the paper and many of the criticisms the authors present depend on aspects independent of the ordering supported by the communication system. Durability is seen again and again (in the sections on transactions), and reliability issues arise in several parts of the paper. The reality is that one cannot divorce Isis from its execution model, and CATOCS is used in this paper as a synonym for a feature of Isis and related systems. By doing so, the authors create an unreasonable bias against systems supporting CATOCS properties as a part of an execution model such as virtual synchrony, since they are able to attack CATOCS on the basis of aspects that are addressed by the virtual synchrony model, but not specifically by message ordering.

3. *The ordering property provided by CATOCS is called incidental ordering.... [but] many systems use or provide what we call prescriptive ordering, where message order is effectively explicitly specified...* This statement is true, and applies as much in Isis applications as in TIB applications. But there are advantages to to simplifying the task of the programmer, by removing the need to repetitiously solve the same problems in the same manner in multiple applications. And there are advantages to allowing the programmer to use asynchronous communication with the assurance that a pre-existing system component will see them in the order they were sent, without gaps even if failures occur. So, the value of additional ordering information, as needed, does not obviate the benefits of system-provided ordering properties.

4. *It can't say for sure.* In some ways it can. It can say, for sure, that there won't be gaps in the causal past after a crash, due to inconsistent enforcement of atomicity, so that a message m is seen by all its destinations but a prior message m' is seen by none. It can say, for sure, that if you send a message asynchronously to a server someone else coded, a subsequent message will arrive after the first one. For the complex, application-specific cases Cheriton and Skeen construct, nothing could say "for sure." So, causal ordering is no panacea, but it greatly simplifies the developer's task.

5. *It can't say "whole story".* No, but neither can any other technique. Does this imply that we should just throw up our hands and offer the developer a far more complex, fragile working environment?

   As for the serializability issue: Isis and related systems can be used to build transactional support, but are generally intended for communication, not database management. By omitting discussion of the virtual synchrony model, Cheriton and Skeen imply that systems like Isis and Transis lack any execution model, and hence can only obtain consistency and data replication by supporting transactions in some awkward way. But this ignores virtual synchrony: a cheaper, communication-oriented way to obtain consistency and reliability, and even durability, without resorting to a highly restrictive serialization-based programming model, and the correspondingly high run-time costs.

   In transactional applications, Isis is mostly used for data distribution, or for linking non-distributed databases into some form of wide-area or intergrated database. The transaction issue is being used here as a red-herring, to distract the reader from the real issues – namely, the presense of a less costly and equally powerful execution model in the Isis system, which solves many of the same problems but at much lower cost!

6. *It can't say efficiently.* The claims made by Cheriton and Skeen in this section are simply wrong. They are based on unfounded, hand-waving demonstrations and ignore the significant amount of protocol engineering that goes into building systems like Isis and Transis.

   These systems are highly efficient, do not suffer the absurd overheads of which the authors speak with such authority, and in fact reach performance that equals or exceeds, for most purposes, the performance of the less reliable technologies the authors favor. For example, in an RPC environment one pays system call overhead twice on each request, and still has no guarantee that exception reporting will be consistent if a timeout occurs. With asynchronous Isis-style multicasts, many messages may be packed into each packet, amortizing costs, and yet there are much *stronger* guarantees associated with failure reporting. Semantic information, such as one is given in Isis, can greatly *reduce* costs relative to those in a weaker programming model.

Inefficiency, in fact, is more likely to arise in the approach Cheriton and Skeen favor, since they would leave the application builder with such weak guarantees that some sort of application-layer protocol (requiring extra messages) is nearly inevitable. How likely are these ad-hoc, non-standard solutions to be highly efficient and optimized? How often will they introduce bugs? How much additional development and debugging time will result from the decision to impose this burden on the developer?

And this says nothing of the cost of transactional technologies, which impose both code style restrictions and invoke complex log-forcing mechanisms, commit protocols, and require the ability to roll back actions. The cost of durable data replication is far higher than the cost of consistency within a primary partition: why do Cheriton and Skeen overlook this issue?

I would argue that, based on the available evidence, systems such as Isis and Transis and Amoeba offer far better performance than a transactional alternative, and a far simpler programming model. And I think that theoretical proofs carry little weight in this domain. After all, Fisher Lynch and Patterson once proved that asynchronous distributed systems cannot tolerate even one failure. Yet systems like Isis, while subject to this limitation, very rarely would encounter the case in which progress is not possible. Skeen once proved that any database system that uses a 2-phase commit is exposed to indefinite blocking and urged that database developers employ 3-phase commit protocols, which reduce the window of vulnerability. Yet few databases use 3-phase commit, because of its cost. Theory has value, but only when it can be shown to say something relevant to the question at hand.

Cheriton and Skeen point to the end-to-end argument, and cite a specific example – a file transfer protocol – to buttress their point. This argument could be paraphrased as follows: “In file transfers you need end-to-end acknowledgements, but you might still want to use a reliable transport because it gives performance gain, by reducing end-to-end retransmissions. But using CATOCS if end-to-end mechanisms are also present doesn’t improve performance or simplify the development task compared to an unordered end-to-end commit protocol.” That is, Cheriton and Skeen conclude that CATOCS ordering has costs but few benefits, and that end-to-end messages would normally be needed in addition to any messages delivered through a layer implementing CATOCS ordering, so that such benefits as it has can be obtained in the application layer at no additional cost.

Yet our response has identified a whole series of specific benefits tied to the ordering mechanisms, or to the combination of ordering and synchronization mechanisms seen in Isis. Causal and total ordering allow the user to send messages asynchronously without fear of delivery ordering anomalies or gaps, to program using powerful paradigms such as state machines and tools such as consistent replicated data, and (when combined with the virtual synchrony model), to develop applications that react to failures and recoveries in simple, consistent ways, and without the need for complex application-layer protocols. So, we would argue that this model greatly simplifies the tasks the programmer must address. Moreover, despite Cheriton and Skeen's claims to the contrary, performance of the approach is good. True, there are still cases where the programmer would be involved in application-specific ordering and reliability problems. But, after all, we are talking about systems development tools here, not magic.

7. *Netnews example.* The authors here are arguing for a style of ordering interface, in which a message indicates precisely which messages it refers to. This seems to have more to do with how CATOCS properties are implemented than with the desirability of the properties. However, since in practice one would not expect a causal delay mechanism to actually delay messages very often, the core inefficiency to which they point – namely that a message will be causally after everything the poster has read – seems to me to be a fairly minor issue; most of those messages should already have reached their destinations before this reference to them is made. After all, causal delivery delays only occur when a sort of communication-level race condition arises, such as when a packet is lost and has to be retransmitted, and this is rare. Moreover, they propose a scheme that imposes just the form of quadratic overhead of which they are elsewhere critical; a potentially complex garbage collection mechanism would be needed to avoid this.

   Actually, there is a second case where causal ordering issues arise, namely when flow control causes substantial numbers of messages from various sources to pile up. But here, the issue is not so much one of delaying messages as of letting them through in the order desired by the application: the messages are usually all at their destination and the causal delivery mechanism is more of a "scheduler". Since the costs of this type of scheduling are so low, it is difficult for us to see major advantages to omitting to order messages prior to delivery, when the messages are all present in the receiving process, and the issue is just one of sorting them prior to hand-off to the application.

   The scaling claims, as discussed earlier, are simply based on an incorrect analysis.

   The claim that users don't find non-causal ordering very troublesome overlooks the fact that users in the examples Isis has used are programs, and the question is the complexity of programming around such problems, not one of having some human being think about an apparent non-sequitor and recognize that some prior message is apparently missing. And this discussion omits the gap-freedom consideration, a property that causal delivery offers and that the scheme Cheriton and Skeen favor would lack. Their proposed solution would not address this issue; if a gap did arise because of a failure, the whole news group could freeze up from the perspective of a process that did not receive a message that was delivered to most others.

8. *Securities trading example.* As discussed earlier, the issue here revolves around the claim made in my work that CATOCS properties simplify the development of this type of application. Cheriton and Skeen argue that because these properties are not

able to enforce "all" issues that an application might wish to address, one should not attempt to address any issues. I discussed this example above, so let me just observe that where Cheriton and Skeen conclude that CATOCS properties do not simplify the application, above I argued that when combined with atomicity properties, they in fact greatly simplify such an application. True, there remains a need for some amount of application-specific logic, but the complexity of the application would be hugely reduced by the elimination of all logic concerned with atomicity and ordering of events relative to past events. And, this may eliminate many possible sources of bugs, inconsistency, and unreliability.

9. *Global predicate evaluation.* In this section of their paper, Cheriton and Skeen present objections to an algorithm for global state detection used by Robbert Van Renesse as an illustration of one benefit of causal multicast, namely that it is delivered along a consistent cut.

   These objections involve several aspects. First, the authors argue, the CATOCS methods require that these communication primitives be used universally; otherwise, messages sent external to the CATOCS subsystem could lead the protocol to fail. This is an overstatement: all messages *relevant to the predicate being evaluated* need to travel through this subsystem, but the remainder of the application can do as it likes. This is a much weaker limitation than Cheriton and Skeen make it out to be.

   The authors review a variety of specialized protocols for predicate evaluation, and I do not dispute that many of these exist, or that some other algorithm might outperform a CATOCS-based solution. The core issue here is the tradeoff between simplicity and standards on the one hand, and optimality on the other. Given an environment in which CATOCS multicast is already available, it is desirable to be able to use this mechanism to accomplish predicate evaluation in a reasonably efficient way. In a system dedicated to predicate evaluation, a specialized solution might be preferable. In particular, the protocol Van Renesse describes would actually be reasonably efficient and is exceptionally simple; it may not be optimal, but optimality isn't always the goal.

10. *Transactional applications.* As noted earlier, Isis is normally employed either in entirely non-transactional control or data distribution settings. As an adjunct to a transactional service, Isis is often used for connecting clients to a transactional subsystem, as a tool for load balancing or replicating data, or a form of glue for combining multiple databases into a single system in a local or wide-area network, or for distributing information to database clients that have registered a trigger or placed some form of monitor on the database. But in none of these cases is it necessary for Isis itself to directly support a transactional model.

    However, it bears notice that in a 1986 paper published in ACM TOCS, Tommy Joseph and I published a complete algorithm for transactions on replicated data using causally ordered multicast in several innovative ways that we considered innovative. We demonstrated that transactional systems could be built in which failures would not provoke aborts (a desirable property in many applications), and in which all updates to replicated data could be performed asynchronously (a very desirable property). In fact, Isis implements a transaction tool which operates this way, although it is not a widely used part of our system.

Recently, I wrote a paper on this issue – the choice between transactional and virtually synchronous consistency, and the options that exist for integrating the two approaches in a single system. Interested readers may wish to refer to [Bir] for a detailed discussion of the question.

11. *Replicated data.* The problem I have with this section of the Cheriton - Skeen paper is that they slip in a transactional model without pointing out that virtual synchrony offers a lightweight alternative with only slightly weaker semantics.

    This occurs in several ways. First, the authors immediately introduce a transactional interface for their replicated data, even though one can meaningfully talk about replicating data, structures, and control without accepting this "baggage" in such a blind way. We replicate all of these forms of information within Isis process groups, supporting coherent local access, transfer of the current state to a joining member, replicated update, and locking mechanisms (as well as logging and recovery mechanisms). CATOCS ordering is just one element of the solution, but it is a valuable element because it allows us to support highly asynchronous updates and locking.

    The second way the authors bias their discussion is by assuming that the replicated data is stored externally to the application, on a file, and that consistency with this stored data (what they call durability in the introduction) is needed. But many applications are concerned with online progress and not with consistency relative to external data files. And for such applications, these criticisms are not relevant. To support external consistency one pays a high performance price. Internal consistency in a primary partition can be achieved at much lower cost.

    The third bias is seen in the discussion of the Deceit file system. A reader of Siegel's thesis would find that Deceit performs as well as alternatives for most purposes, and has some benefits (such as the ability to dynamically vary the level of replication on a per-file basis, and to support other forms of flexibility through parameters such as the safety level). Siegel explicitly considers the decision to build over Isis and states that this decision was justified because Deceit was enormously simplified, indicating that although a small performance penalty may have resulted, the reduced design complexity and ability to carry out a more sophisticated design had compensating advantages.

    As an aside, Cheriton and Skeen speculate when they make performance comparisons of HARP and Deceit. I do not believe that HARP and Deceit have ever been compared on the same hardware. In particular, HARP is widely known for its use of non-volatile RAM as an accelerator; it does not need to wait for a disk to flush in order to consider data safely logged. This was one of the main research issues in HARP, and HARP performance is normally cited as an example of the potential impact of special hardware on file replication performance, not because the system uses some other ordering or fault-tolerance model. It is not at all clear how non-volatile RAM could be exploited in Deceit, but it is entirely possible that substantial speedups could be achieved for some operations using such hardware.

On the other hand, when Deceit was compared with HA-NFS, a hand-coded replicated file server built at IBM (by Anupam Bhide), performance was found to be nearly identical.

A fourth bias is reflected by the criticisms of primary-copy replication, also in this section. True, for any given entity (at the level of granularity of the concurrency control mechanism), Isis favors a primary copy scheme because this permits the use of asynchronous causally ordered multicasts for updates. But the alternative of using a totally ordered multicast is always available, and is chosen in some cases precisely to avoid a primary-site dependency. Even where a primary-site scheme is used, though, the authors neglect to point out that Isis allows multiple processes to act as primaries for different data items concurrently. Moreover, the claims about limited multi-threading are simply wrong; under a package such as Pthreads, one can multi-thread an Isis application if appropriate care is taken.

12. *Replication in the large.* Isis uses a wide-area networking layer with good results and has been scaled into world-wide networking environments by many users. Once again, the authors make unjustified claims here, which are simply refuted by the data. It is true that in wide-area settings, we use causal ordering to the exclusion of total, because of the cost implications of total order, which rise with latency. But this has not been much of a restriction in practice, because of the equivalence of causal and total order when the sender holds a token. By structuring wide-area data to have a single update source for any given data item, a technique used by Skeen himself in some of his work, and by others such as Garcia-Molina and Kogen, wide-area data replication is reduced to a completely asynchronous update problem, which causal ordering handles extremely well and at very low cost.

13. *Distributed real-time applications.* I commented extensively on this above, and will not repeat myself here. In summary, Isis is suitable for applications that mean "fast", not "guaranteed", or that mean "guaranteed, but not very tight limits", or that mean "tight limits, but only locally to a processor – not on the network." CATOCS doesn't help in such cases, but neither does it hurt. Even for very tight limits, it isn't really known whether the CATOCS properties necessarily need to interfere with real-time goals – in my view, this represents an interesting area for future research.

14. *Scalability.* As noted above, this analysis is incorrect:

    - Although the rest of their paper is multicast oriented, here they assume a point-to-point communication model.
    - They seem unaware of the techniques used to enforce CATOCS ordering for point-to-point messages, which special case these (to avoid exposure to gaps in the causal past after crashes) and hence separate this issue from the representation of CATOCS information for multicasts.

- It ignores the finite capacity of the network itself, which limits the buffering load on a process by limiting the number of messages it might receive during the period before stabilization occurs.
- It is unaware of compensating mechanisms available within Isis for actively reducing resource consumption by buffered messages, such as simply retransmitting them to any remaining destinations (which discard duplicates).
- It neglects to note that when using the conservative scheme, Isis overhead for representation of causality would never exceed a single vector timestamp per group, with one entry per process permitted to send to the group.
- It fails to note the substantial opportunities for compression of vector timestamps, which in any case only need one timestamp (integer) per sender in a group.
- It ignores a number of mechanisms that intervene to restrict the propagation of causality information in the case where multi-group vector timestamps are used (and this remains experimental in our work on Isis), such as time-stamp compression.

The authors talk with authority about work that they apparently do not understand. And they reach conclusions that have absolutely no bearing on reality.

Let me conclude this appendix with a comment on the conclusions section of the Cheriton and Skeen paper. My work never claimed that the CATOCS ordering properties, or atomic multicast delivery properties, are needed in systems other than in Isis. Rather, I have always argued that Isis is a demonstration that in the context of appropriate tools, and with an appropriate execution model, these properties make it considerably easier to develop a class of reliable distributed system that would otherwise be difficult to develop.

As I wrote this rebuttal of the Cheriton and Skeen paper, I came to recognize that although in many ways these authors attack on my work, the underlying reason for this attack is that the authors consider my work to have implications for their systems, V and TIB respectively. Yet those systems both do good jobs of solving the problems for which they were intended, and I have never argued that Isis is better at solving those same problems by virtue of its ordering properties.

I challenge Cheriton and Skeen to advance a positive version of their argument, backed with experimental data. But in contrast to the TIB paper that will appear in SOSP, such an argument would need to show a way for the application developer to accomplish the same tasks that are easily solved using Isis. Cheriton and Skeen have argued that my approach to solving problems of reliability, asynchronous communication, consistency, and dynamic reconfiguration is unreasonable, giving many reasons. But they offer no clear alternative, for solving the same problems, instead stating that the application layer should do so. Their TIB paper doesn't address these issues at all. The reader who accepts their position would be left with no way to build this class of system. Perhaps system developers would favor a better solution to their problems, but I am doubtful that they will abandon a workable approach in favor of no solution at all.

References

[ABHN91] Mustaque Ahamad, James Burns, Phillip Hutto, and Gil Neiger. Causal memory. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, GA, July 1991.

[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.

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

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

[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.

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

[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.

[Sch88] Frank Schmuck. *The Use of Efficient Broadcast Primitives in Asynchronous Distributed Systems.* PhD thesis, Cornell University, 1988.