Towards Causal Datacenter Networks

Ellis Michael
University of Washington

Dan R. K. Ports
Microsoft Research

ABSTRACT

Traditionally, distributed systems conservatively assume an asynchronous network. However, recent work on the co-design of networks and distributed systems has shown that stronger ordering properties are achievable in datacenter networks and yield performance improvements for the distributed systems they support. We build on that trend and ask whether it is possible for the datacenter network to order *all messages* in a protocol-agnostic way. This approach, which we call omnisequencing, would ensure *causal delivery* of all messages, making consistency a network-level guarantee.

ACM Reference Format:
Ellis Michael and Dan R. K. Ports. 2018. Towards Causal Datacenter Networks. In *Proceedings of 5th Workshop on Principles and Practice of Consistency for Distributed Data , Porto, Portugal, April 23–26, 2018 (PaPoC'18),* 4 pages.
https://doi.org/10.1145/3194261.3194269

1 INTRODUCTION

The asynchronous network has long been viewed as an obstacle to achieving data consistency in distributed systems, as it can reorder messages arbitrarily. In this work, we ask whether a network can instead *aid* in building consistent systems. That is, we ask: *can a datacenter network itself provide consistency guarantees*?

Our work is motivated by a confluence of trends in network design. Recent work has shown that it is possible to build highly efficient packet sequencing devices, leveraging new programmable network hardware. These have been used by new distributed systems to accelerate specific applications, like Paxos-based replication and distributed transactions.

We take this further, asking whether it is possible to sequence *every* network message – a concept we dub *omnisequencing*. While establishing a total global order of message delivery might be desirable, doing so at large scale appears infeasible because it places strict limits on concurrent message delivery. Weaker consistency levels, such as causal delivery, however, may well be possible, and these nevertheless offer substantially stricter semantics than traditional unordered message delivery. We discuss several deployment scenarios for an omnisequenced network, including a dedicated network, a virtualized overlay on an existing physical network, and a multi-tenant configuration. In effect, we aim to raise the bar on consistency, making causal delivery the new baseline.

A causally ordered datacenter network resembles traditional software-based primitives like causal multicast from ISIS [1], but can be implemented efficiently and at large scale. It could be used, for example, to transparently make an existing storage system causally consistent, or as a primitive to simplify the design of higher-level protocols such as transaction processing.

This is very early-stage work. While we sketch a particular system design below, the main purpose of this paper is to provoke discussion about the possibility of network-level consistency guarantees and their possible applications.

2 BACKGROUND

The standard approach to distributed systems design is to assume nothing about the network – to model it as an asynchronous, lossy medium that provides no guarantees on which packets will be delivered, at what time, or in what order. And indeed practical networks regularly demonstrate drops, delays, and reorderings. Latency varies depending on network utilization. Modern networks are designed with multiple paths between any two nodes and aggressively exploit this property for increased throughput; as a result, latency variance across different paths can cause packets to be reordered. Messages are dropped regularly: not only during (relatively rare) failures, but also as a means of signaling congestion.

Bleak as this situation may seem, there is in fact reason for optimism. Today's network devices are far from the "dumb" hubs of the past – a modern network switch runs hundreds of protocols, classifying packets and applying increasingly complex protocol-specific rules at line rate. These rules do not simply call for packets to be forwarded but can also inspect application-level content, perform computations, and modify the packet [4]. This power is exposed to system administrators through software-defined networking [3]. The

ACM ISBN 978-1-4503-5655-8/18/04…$15.00
https://doi.org/10.1145/3194261.3194269

net result is that, at least within the confines of a single datacenter, one ought to envision the network itself as a powerful computational element.

While no "killer app" has yet emerged for this level of programmability, it has been applied to improve congestion control [20], load balancing [8], and network monitoring [16]. Most relevantly, a recent line of work (including ours) has employed in-network processing to accelerate consensus operations [5, 12, 13, 18]. A key idea in this approach is to build a *sequencer* that assigns increasing sequence numbers to packets that pass through it, which can be done highly efficiently [9, 13]. Routing a subset of messages (e.g., operations for a Paxos-replicated state machine) through one sequencer at a time induces a total ordering over that set of messages, and can be used to build a high-performance consensus protocol [13].

3 OMNISEQUENCING

Can we extend the idea of network-level sequencing beyond the scale of a replica group to that of an entire application – or potentially an entire datacenter? Doing so presents significant scalability challenges, but has the potential to greatly simplify the development of applications or improve data consistency.

The obstacles to scalable total-order sequencing are both engineering-level and fundamental ones. In particular, atomic totally-ordered delivery requires coordination and limits potential concurrency, making its implementation at large-scale appear intractable. We target, instead, the weaker model of *causal delivery* of messages. In this section, we sketch a design to illustrate that causal delivery can be achieved at scale using a co-design of network-level and server-level optimizations.

3.1 Causal Delivery

We begin by defining the causal delivery model we target. In this model, messages are delivered to nodes in a way that respects the happens-before ($\rightarrow$) relation [2, 11, 19]. More specifically, consider two messages, m and m', which are both sent to the same node. Let s_m denote the send event of m and r_m denote the delivery of m. Causal delivery is the property:

$$s_m \rightarrow s_{m'} \implies r_m \rightarrow r_{m'}$$

Causal delivery implies FIFO delivery across single channels. Figure 1 shows an example of an execution in which causal delivery is not satisfied.

Causal delivery has traditionally been maintained using protocols based on *vector clocks*. In a broadcast network, it suffices to have each node maintain a vector clock, updating it as messages are received, and attach its vector clock to every message it receives. To ensure causal delivery, nodes delay delivery of a message until they have delivered all preceding messages, as indicated by its vector clock.

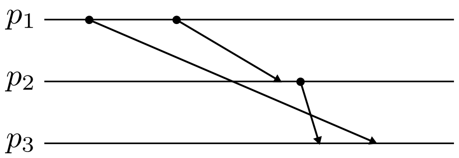

Figure 1: The canonical example of an execution which respects FIFO but not causal delivery.

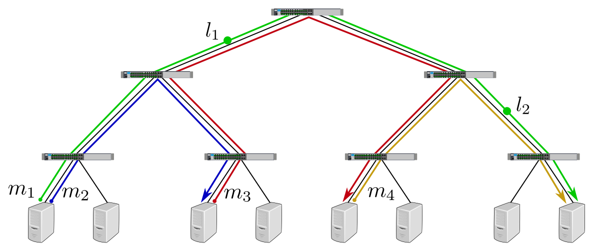

Figure 2: An illustration of how FIFO links in a tree topology guarantee causal delivery. Here, $s_{m_1} \rightarrow s_{m_2} \rightarrow s_{m_3} \rightarrow s_{m_4}$. By the time m_2 is delivered, m_1 must have been sent across l_1. Similarly, the delivery of m_3 implies m_1 was sent across l_2. Therefore, m_4 will be delivered after m_1.

When nodes can communicate directly with each other through unicast messages, they must maintain extra state. Specifically, Schiper et al.'s protocol for causal delivery requires maintaining an $On \times n$ (where n is the total number of nodes) *ordering buffer* that encodes, for each node, the latest message number that every other node sent to it [19]. This information must be attached to every message sent in the system.

Neither of these two approaches is practical for large datacenter-scale systems. Relying on a broadcast network limits the message processing capability of an entire system to that of a single node, while the second requires an unmanageably large amount of metadata to be attached to each message.

3.2 A Scalable Solution

Taking inspiration from recent work on co-designing networks and distributed systems [12, 13, 18], we propose a mechanism in which the *physical topology* of the network itself is used to guarantee causal delivery of messages. For simplicity, consider a network topology that is a simple tree.[1] Suppose that message delivery could be made reliable and that all links ensure FIFO delivery. It is not hard to see that such a network would guarantee causal delivery of all messages; Figure 2 illustrates this through an example.

Modern networking hardware is capable of inserting a per-outgoing-port sequence number on each packet and checking

[1]Tree-based network topologies are universal, though real network topologies usually incorporate multiple trees for fault tolerance and load balancing. We revisit this issue in the following section.

that each incoming packet's sequence number matches the expected one for that port. This can be used to verify that a link is indeed a FIFO channel.

The problem of lost messages. Causal delivery implies that if any message is lost, no causally dependent messages – including any subsequent message from the same sender – can be delivered. Because we envision omnisequencing being implemented in network hardware with limited capability to maintain state, the standard approach to building reliable FIFO channels (buffering the messages sent across each link until acknowledged) cannot be used. However, message loss can be *detected* as a gap in sequence numbers arriving on some link, implying that some message sent from the subtree rooted at that link was lost. Such a failure could either be exposed to the application to recover from [13], or trigger a network-level recovery protocol.

A simple recovery protocol might invoke a message recovery coordinator that temporarily pauses message delivery across the link in question, then contacts each server in the subtree to obtain the set of outstanding, unacknowledged messages. It then delivers each of these pending messages in a causally-consistent order – achievable by having each server maintain a logical clock – before resuming normal operation. Such a protocol is expensive! However, link-level flow control and other technologies (collectively referred to as "lossless Ethernet") can make packet loss extremely rare – to the point that others have advocated treating packet loss as equivalent to node failure [10]. In light of that, an expensive protocol may indeed be tolerable.

Note that the failure of a switch can be recovered from in the same way, by treating all of its links as having dropped messages. The switch can then be replaced using a standard rerouting mechanism.

3.3 Virtual Omnisequencing

Above, we have assumed that the datacenter network is a simple tree. While this assumption is not wholly without merit – the spanning tree protocol [17], which deactivates links until the network forms a tree, has been a mainstay of network design for three decades – today's networks are designed to exploit redundant links for higher bandwidth [6, 15].

Our design above, by requiring a simple tree topology, limits the bisection bandwidth of an omnisequenced network to the bandwidth of a single link. We propose two strategies for mitigating this problem by overlaying an omnisequenced network onto a more well-connected physical network.

First, note that only a tree topology is only required for messages of a single causal domain. Separate trees could be implemented for independent traffic, either on separate physical networks or virtualized onto one network. The virtualization approach is ideally suited to the multi-tenant datacenter setting. Cloud providers are increasingly offering configurable deployment options to their customers. A causally-ordered network could be another such offering. The datacenter operator could then deploy many virtual omnisequencing networks, each providing causal message delivery to a different customer.

Second, note that in the tree topology is only necessary in establishing a causal *order* for message deliveries. Following the classic networking principle of separating the control plane from the data plane, nodes could send bulk data across the traditional network as normal, and send *delivery notifications* for those messages as omnisequenced packets. A node receiving a message would wait for both the message and the delivery notification (containing a unique identifier for the message).

Separating the ordering of messages from the transfer of data has the potential to not only increase maximum bandwidth but also further decrease the probability of dropped packets on the omnisequencing network. The small, uniform size of delivery notifications could enable aggressive optimization and prioritization [7, 18].

4 DISCUSSION

Previous work has shown that vector timestamps for causal consistency can be compressed by exploiting *temporal locality* [14]. Omnisequencing can be seen as compressing ordering information by exploiting *spatial locality* in the network, while still providing the gap detection property that is critical for causal message delivery. All messages sent across the same link are conflated to the same sequence number, no matter their origin. Unlike schemes which involve nodes and messages having On or On^2 separate timestamps, our approach is space-efficient and implementable in network hardware.

Omnisequencing is an extension of previously proposed sequencing approaches [12, 13] while still providing the same guarantees. The one missing feature from the proposal above is support for multicast and similar primitives; however, this is readily implementable. Whereas previous sequencing solutions were application specific, omnisequencing is a general way to provide a strong ordering property in the network, enforcing consistency across application boundaries.

We have focused on causal delivery as the network consistency model here, as it is relatively well understood. An open question is whether there are other weaker consistency models (perhaps including those yet to be discovered!) that support both an efficient implementation and useful semantics in the network context. Regardless, we believe that recent advances in network hardware provide an exciting new capability to implement consistency primitives at the network level, offering the potential to provide performance gains for and simplify the design of both strong and weakly consistent distributed systems.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under award CNS-1615102 and a Graduate Research Fellowship, and by gifts from Google and VMware.

REFERENCES

[1] K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP '87)*, Austin, TX, USA, Oct. 1987.

[2] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, Jan. 1987.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *Proceedings of ACM SIGCOMM 2013*, pages 99–110. ACM, 2013.

[5] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 5:1–5:7, New York, NY, USA, 2015. ACM.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *Proceedings of ACM SIGCOMM 2009*, Barcelona, Spain, Aug. 2009.

[7] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can JUMP them! In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, Oakland, CA, USA, May 2015. USENIX.

[8] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. NetCache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP '17)*, Beijing, China, Oct. 2017. ACM.

[9] A. Kalia, M. Kaminsky, and D. G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, Denver, CO, June 2016. USENIX Association.

[10] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 185–201, Savannah, GA, 2016. USENIX Association.

[11] L. Lamport. Time, clocks, and ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[12] J. Li, E. Michael, and D. R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP '17)*, Beijing, China, Oct. 2017. ACM.

[13] J. Li, E. Michael, A. Szekeres, N. K. Sharma, and D. R. K. Ports. Just say NO to Paxos overhead: Replacing consensus with network ordering. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Savannah, GA, USA, Nov. 2016. USENIX.

[14] S. A. Mehdi, C. Littley, N. Crooks, L. Alvisi, N. Bronson, and W. Lloyd. I can't believe it's not causal! scalable causal consistency with no slowdown cascades. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, pages 453–468, Boston, MA, 2017. USENIX Association.

[15] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of ACM SIGCOMM 2009*, Barcelona, Spain, Aug. 2009.

[16] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of ACM SIGCOMM 2017*, Los Angeles, CA, USA, Aug. 2017. ACM.

[17] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended lan. In *Proceedings of ACM SIGCOMM 1985*, Whistler, BC, Canada, Sept. 1985. ACM.

[18] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy. Designing distributed systems using approximate synchrony in datacenter networks. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, Oakland, CA, USA, May 2015. USENIX.

[19] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, pages 219–232, London, UK, 1989. Springer-Verlag.

[20] N. K. Sharma, A. Kaufmann, T. Anderson, C. Kim, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, Boston, MA, USA, Mar. 2017. USENIX.