arXiv:2603.04774v1 [cs.DC] 5 Mar 2026

语义时间之箭，第三部分：

RDMA 与完成谬误

Paul Borrill
DÆDÆLUS

02026年2月27日

为什么交付保证会摧毁意义

摘要

这是由五篇论文组成的《语义时间之箭》系列中的第三篇。第一部分（Borrill，2026a）指出，计算领域隐藏的时间之箭是语义性的，而非热力学性的，并追溯了从 Eddington 经由 Shannon 和 Lamport 延续至今的 FITO 类别错误。第二部分（Borrill，2026b）提出了建设性的替代方案： OAE链路状态机、无限逻辑时间戳，以及减速定理所要求的往返过程。

本文考察了当这些原则在工业规模上遭到违反时会发生什么。远程直接内存访问（RDMA）是当前生产环境中性能最高的数据移动技术，已部署于 Meta 的 24,000 个 GPU Llama 3 训练集群、Google 的多租户数据中心以及 Microsoft 的 Azure 基础设施中。 RDMA 的核心承诺很简单：绕过内核，绕过中央处理器，并以线速将数据直接写入远程内存。

我们认为，这一承诺包含了第一部分所指出的那种类别错误。 RDMA 的完成语义——告知发送方“操作已完成”的信号——保证了放置（数据已写入远程网卡缓冲区），却不保证提交（数据已被接收应用程序在语义上整合）。我们将其称为完成谬误：把交付确认与语义一致混为一谈。

我们通过 RDMA 写操作的七个时间阶段来阐明这一谬误，指出阶段 T_4 （返回完成信号）与阶段 T_6 （满足应用语义属性）之间的差距可能无限大。随后，我们通过四个生产规模案例研究追踪其后果：Meta 在 32,000 个图形处理器上的 RoCE网络、Google 的 1RMA重新设计、Microsoft 的 DCQCN互操作性故障，以及 SDR-RDMA 部分完成问题。

本文最后通过比较语义分析指出，CXL 3.0、NVLink 和 UALink 都解决了完成谬误的部分问题，但没有任何一种能彻底消除它。只有采用强制反映阶段的协议架构（如第一部分所述），才能弥合交付与提交之间的差距。

1 引言：破坏数据最快的方法

远程直接内存访问速度很快。它绕过操作系统内核，避免中央处理器介入数据路径，并能以超过 400 Gb/s 的速率直接写入应用程序内存。对于如今主导超大规模数据中心的人工智能训练工作负载而言，RDMA 并非可有可无——它是唯一能在一次训练步骤的同步时间预算内，在数千个图形处理器之间传输梯度张量的技术。

但速度并不等于正确性。本文的核心论点是， RDMA的完成语义在硬件层实现了 FITO 范畴错误：它们把数据的正向传递视为通信已经发生的充分证据。用第二部分的术语来说， RDMA 混淆了 放置 与承诺——它在时间序列中语义箭头尚未确立的时刻，就发出“完成”信号。

其后果并非偶尔的数据丢失，而是系统性的语义损坏：数据结构虽然能够正确解析，却包含不一致的状态；完成信号报告成功，多字段不变量却遭到破坏；集合操作则会悄无声息地出现分歧，因为在可见性得到保证之前，单个传输就已完成。

本文其余部分安排如下。第 2 节考察 RDMA 的架构前景及其背后的工程假设。第 3 节将一次 RDMA 写操作分解为七个时间阶段，明确指出完成谬误发生的具体位置。第 4 节分析 8 字节原子性边界及其对实际数据结构的影响。第 5 节介绍四个生产规模案例研究。第 6 节阐述完成谬误与 OCP 静默数据损坏倡议之间的联系。第 7 节对 CXL、NVLink 和 UALink 进行比较分析。第 8 节总结全文并预告第四部分。

2 RDMA 的架构前景

RDMA 旨在解决一个具体问题：传统网络协议栈的开销。在传统 TCP/IP 通信路径中，一次网络写操作需要经过系统调用（内核态切换）、从用户空间复制到内核缓冲区、协议处理（TCP 分段、IP 封装）、向网卡进行直接内存访问传输、数据传输，以及接收方的反向路径。每个阶段都会增加延迟并消耗 CPU 周期。

RDMA 消除了这条路径中的大部分环节。应用程序向网卡上的一个队列对（QP）提交工作请求；网卡通过直接内存访问（DMA）从已注册应用内存中直接读取数据；网卡负责分段、传输和重组；在接收方，网卡则将数据直接写入远程应用程序的已注册内存，无需远程 CPU 参与。软件唯一需要做的就是提交请求并轮询完成状态。

提供三类操作：

发送/接收： 需要接收方配合的双边操作。接收方提交一个接收缓冲区；发送方将数据传入其中。双方都需参与。

写操作： 单边操作。发送方指定远程地址并向该地址写入数据，无需远程 CPU 参与。远程应用程序不会收到通知。

原子操作： 对远程内存执行比较并交换（CAS）和取并加（FAA）。原子操作的长度仅限 8 字节，并由远程网卡而非远程 CPU 执行。

单边操作是 FITO在硬件层最纯粹的体现。它们像 Shannon 建模信道那样，将通信建模为：发送方发送，信道传递，接收方处于被动状态。返回路径——确认——由传输层处理（可靠连接模式），但不承载任何语义内容。它只确认网卡已接收数据，而非应用程序已理解数据。

3 RDMA 写操作的七个时间阶段

要准确阐明完成谬误，我们将一次 RDMA 写操作分解为若干时间阶段。每个阶段都代表操作生命周期中的一个不同节点；完成谬误就是假设其中一个阶段意味着后续阶段已经发生。

定义 3.1（七个阶段）。

T_0**：**提交。应用程序向一个队列对的发送队列发布一个写工作请求。此时，网卡尚未接触数据。

T_1**：**放置。网卡通过直接内存访问从已注册应用内存中读取数据，并将其暂存于网卡的内部缓冲区中。此时，数据已"放置"到网络子系统中。

T_2**：**传输。网卡将数据分割成网络数据包，添加报头，并将其发送到线路（*InfiniBand*）或以太网结构（*RoCE*）上。

T_3**：**远端放置。远程网卡接收数据包，并通过直接内存访问将其直接写入指定的远端内存地址。此时，数据已位于远程内存中，但可能仍在已发布写缓冲区内，尚无法通过缓存层次结构访问。

T_4**：**完成。 发送网卡收到传输层确认后，会向发送方的完成队列写入一个完成队列条目（*CQE*）。应用程序通过轮询或收到通知，得知写操作"成功"。

T_5**：**可见性。 数据通过缓存一致性机制对远程 *CPU* 可见。这可能需要远程 *CPU* 执行内存屏障，或读取该位置以触发缓存填充。 T_5 可能明显晚于 T_4。

T_6**：**语义一致。 远程应用程序读取数据，根据应用程序不变量对其进行验证，并将其整合到自身的语义状态中。由此建立多字段一致性、版本兼容性和因果排序。

完成谬误是指这样一种假设： $T_4 \Rightarrow T_6$，即完成信号能够保证语义一致。事实上， T_4 与 T_6 之间的差距可能任意大，而两者之间的操作——缓存一致性、内存排序、不变量验证、因果整合——完全不在 RDMA的职责范围内。

在 OAE 第二部分的框架中，各个阶段可以清晰对应：

阶段	RDMA 状态	OAE 等效项
T_0–T_1	已提交	IDLE → TENTATIVE
T_2–T_3	进行中	TENTATIVE
T_4	已完成	TENTATIVE （仍然！）
T_5	可见	TENTATIVE （仍然！）
T_6	已同意	COMMITTED
	REFLECTING	未出现在 RDMA 中

该表让这一谬误显而易见： RDMA 在 OAE 仍会将其归类为 TENTATIVE的时刻报告"已完成"。不存在反映阶段；也没有任何机制让接收方确认自己已在语义上处理了数据。完成信号其实是传输层确认，却伪装成了应用层承诺。

4 原子性鸿沟：8 字节边界

RDMA提供两种原子操作：比较并交换和取并加。两者都仅限于 8 字节（64 位），并且在远程网卡上执行，而不是在远程 CPU 上执行。这就造成了我们所谓的 原子性缺口： RDMA 能够以原子方式保证的内容（8 字节）与应用程序要求以原子方式处理的内容（任意数据结构）之间的差距。

请看一个分布式哈希表条目：

字段	Size	内容
版本	8 字节	单调计数器
key	32 字节	哈希键
值	256 字节	有效载荷
状态	8 字节	{有效、已删除、已锁定}

这个 304 字节的结构至少跨越 5 条缓存行（每条 64 字节）。一次更新该条目的 RDMA 写操作并非原子操作：并发读取者可能观察到新版本搭配旧值，或新值搭配旧状态。该条目能够正确解析——每个字段单独来看都是良构的——但组合在语义上无效。这就是语义损坏的定义：语法正确，语义无意义。

标准的缓解措施是 FaRM 的版本化方法（Dragojevic 等人，2015年）：在每条缓存行的最后 8 个字节中嵌入版本号，使用 RDMA 写操作更新数据，并用最后一次 RDMA 原子操作 CAS 提交版本。读取方在读取前后检查版本；若版本不同，则重试该次读取。

这就是缓存行层面的超时重试（TAR）——恰恰是第一部分所认定的、作为分布式系统病理现象根源的 FITO模式。FaRM 并未解决原子性缺口；它只是用仅向前重试加以掩盖，这些重试假定写操作已经丢失，并寄望于最终成功。语义箭头并未确立；它只是通过概率收敛来近似。

5 生产规模证据

完成谬误并非理论上的担忧。在 RDMA 部署的各个规模中，它都会显现出来，从单机 NUMA 域到超大规模训练集群，无一例外。我们将介绍四个案例研究。

5.1 案例研究 1：Meta 在 24,000 个图形处理器上运行的 Llama 3

Gangidi 等人（Gangidi 等人，2024年）报告了 Meta 部署两个各含 24,000 个图形处理器的集群——一个采用 RoCE，另一个采用 InfiniBand——用于 Llama 3 训练。RoCE 集群暴露出若干完成谬误的病理表现：

拥塞级联。 RoCE 依靠优先级流量控制（PFC）实现无损传输。拥塞加剧时，PFC 暂停帧会沿互连网络向后传播，使发送方停顿。但 PFC 工作在链路层，而非语义层：它会暂停某个优先级类别中的所有流量，包括那些并未造成拥塞的流量。结果是发生队头阻塞，延迟已写入远程内存的操作的完成信号——扩大 T_3-to-T_4 gap，并使 T_4-to-T_6 gap变得难以预测。

低熵流量。 人工智能训练会产生集体通信模式（全归约、全收集、归约散布），从而生成少量大型、同步的流。基于等价多路径（ECMP）哈希的负载均衡假设流量具有高熵——即存在许多小型、相互独立的流。当流量熵较低时，ECMP 会退化：多个流被哈希到同一路径上，形成持续的热点。Meta 报告称，基于队列对的流扩展（为每个连接打开多个队列对，以增加哈希熵）使吞吐量提升了 40%。

这是一个范畴错误症状：网络架构假定流量在统计上相互独立（FITO在流级别），但人工智能集合操作存在因果相关性。网络架构的流量模型与流量的语义结构并不匹配。

传输活锁。 即使是低于 10^{-6} 的罕见丢包，也会持续降低性能，因为 RDMA 的可靠传输在检测到任何丢包后，都会重传整条消息。在 1 GiB 的传输中，单个数据包丢失就会触发整个 1 GiB 数据的重传。原始传输的完成信号会延迟到达，甚至永远不会到达；应用程序则会停滞，等待一个本可通过部分确认更高效地解决语义一致问题的完成信号。

5.2 案例研究2：Google的1RMA重新设计

Google的Singhvi等人（Singhvi等人，2020年）得出结论，标准 RDMA “不适用于多租户数据中心”，并从头开始设计了1RMA。关键见解是， RDMA的面向连接模型（队列对绑定到特定远程端点）会在云规模下带来隔离、公平性和安全性问题。

1RMA将每个 RDMA操作视为独立的无连接请求。软件提供拥塞控制、故障恢复和操作间排序——这些功能由 RDMA委托给硬件队列对状态机。

在本系列的框架中，Google的重新设计是对 FITO 假设的部分修正：通过使每个操作相互独立，1RMA避免了这样一种假设，即发往同一端点的一系列操作应按其提交时间排序。但1RMA仍保留了完成谬误：网卡仍在 T_4发出“完成”信号，而到 T_6 的差距仍是应用程序的问题。

5.3 案例研究3：Microsoft数据中心量化拥塞通知互操作性

Microsoft的Azure RDMA 部署（微软，2023年）暴露了完成谬误的一种更微妙形式：网卡世代之间的互操作性故障。

当第二代/第三代网卡（基于硬件的拥塞控制）与第一代网卡（基于固件）通信时，完成语义出现了分歧：第二代/第三代网卡生成拥塞通知数据包的速率压垮了第一代的固件处理，导致过度的速率降低。在相反方向上，第一代基于定时器的拥塞通知数据包合并产生了突发流量，造成第二代/第三代接收端流水线中的缓存未命中。

结果并非丢包——传输层仍持续报告操作成功完成。真正的问题是性能崩溃：完成信号虽然到达，但由于拥塞控制机制彼此冲突，有效吞吐量下降了一个数量级。操作成功完成了；但其含义（在本例中，即预期吞吐量）却遭到了破坏。

5.4 案例研究 4：SDR-rdma 与部分完成

Khalilov 等人（Khalilov 等人，2025年）指出了不可靠连接（UC）模式中一种危害尤为严重的完成谬误。当一个 4 KiB 数据包在 1 GiB

写操作期间丢失时，整个 *1 GiB* 操作都会被标记为丢失。完成信号会报告整条消息失败，尽管 99.9996% 的数据已成功到达。

SDR-RDMA 通过部分完成 API 解决了这一问题：位图会标明哪些数据块丢失，从而实现选择性重传。这是一项重大的工程改进，但也揭示了问题的严重程度： RDMA的完成语义过于粗略，以至于单个数据包丢失就会使已成功写入的数吉字节数据全部失效。完成信号采用二元（成功/失败）模型，无法表示传输的实际状态——部分数据已写入，部分数据则缺失。

在 OAE 框架中，这是 REFLECTING 阶段的失败：接收方无法将其实际收到的内容传达回发送方。完成信号是一个 1 位的响应（成功或失败），但实际需要的是语义摘要——对接收方状态的结构化描述。

6 完成谬误与静默数据损坏

开放计算项目的静默数据损坏（SDC）倡议（OCP SDC，2024年）于2022年10月启动，创始成员包括 AMD、ARM、Google、Intel、Meta、Microsoft 和 NVIDIA。该倡议旨在应对逃过错误检测机制的硬件引发的数据损坏。静默数据损坏是完成谬误的终极体现：硬件执行操作、报告成功，却产生了损坏的输出。

SDC 与完成谬误在 RDMA 系统中以一种尤其危险的方式相互作用。考虑以下时间顺序：

1. 一次 RDMA 写操作将数据写入远程内存（T_3）。 2. 硬件故障（制造缺陷、老化、宇宙射线）在放置过程中、网卡缓冲区中或 DMA 路径中破坏一个或多个比特。 3. 发送网卡收到传输层确认（T_4），并报告操作成功完成。 4. 网卡的错误检测未能发现该损坏（循环冗余校验覆盖传输帧，而不覆盖 DMA 路径；纠错码覆盖动态随机存取存储器，而不覆盖网卡缓冲区）。 5. 应用程序读取损坏的数据（T_6），并将其视为有效数据——完成信号表明写操作已成功。

完成信号不仅是不充分的——它还会积极地误导应用程序。它告诉应用程序操作已成功，而实际上数据已经损坏。应用程序无法在RDMA的语义范围内发现这种损坏；它必须依赖应用层校验和或端到端完整性验证——而 RDMA的“零拷贝、内核旁路”架构正是为了消除这些机制而设计的。

OCP 的 2024年人工智能专用 SDC 白皮书（OCP SDC，2024年）记录了其对人工智能训练造成的后果：全归约期间梯度张量发生的静默损坏可能沿着训练循环传播，生成准确率下降、且无法与训练不足的模型区分的模型。完成谬误意味着训练框架认为每次梯度交换都已成功；这种语义损坏在模型接受评估之前都是不可见的，而评估可能要到数周后才进行。

7 比较分析：CXL、NVLink、UALink

每种主要的互连技术都能应对完成谬误的某些方面，但没有任何一种技术能彻底消除它。

7.1 CXL 3.0：有一致性，但不做承诺

CXL 3.0 的反向失效机制（CXL，2023年）在主机与 CXL 设备之间实现了硬件强制缓存一致性。当设备修改缓存行时，会向所有可能持有过期副本的主机缓存发出反向失效侦听（BISnp）。设备一致性引擎（DCOH）使用类 MESI 协议跟踪主机缓存状态。

这弥合了 T_4-到-T_5 差距：CXL 写操作完成后，缓存一致性确保任何主机后续读取时都能看到更新后的数据。但 CXL 的一致性以缓存行粒度（64–128 字节）运作。涉及多个缓存行的更新并非原子操作：读取者可能会看到新旧缓存行混杂的状态。 T_5-到-T_6 差距——从可见性到语义一致——仍完全由应用程序负责。

CXL 的无序 I/O（UIO）机制让情况更加复杂。虚拟通道 VC1– VC7 承载不受严格排序规则约束的流量，虽然这提高了吞吐量，却打破了这样一种假设：对同一地址区域执行的操作会按提交顺序完成。UIO 操作的完成信号保证数据已送达互连网络，而非在目的端可见——这正是互连网络内部的完成谬误。

7.2 NVLink：没有语义的信号

NVLink 的基于信号排序模型提供了所有生产级互连中最强的完成保证。信号操作会以原子方式更新远程计数器；信号到达时，可保证同一上下文中所有先前的写入操作在目的端均可见。

这消除了单一上下文内操作的 T_4-到-T_5 差距：信号完成时即可保证可见性。但信号本身不承载任何语义内容。它确认数据已可见，却不确认数据是否正确、一致或可解释。如果发送方传输多字段更新，而接收方对这些字段的解读不同（由于版本偏差、模式演化或字节序不匹配），这种差异将无法检测。信号保证数据送达，却不保证对方理解。

NVLink 的适用范围也限制了它的应用场景。它是一种节点内互连（借助 NVLink Switch 可扩展至 8 节点的小型集群）；它无法扩展到数据中心规模，而这正是 RDMA 的完成谬误危害最大的地方。

7.3 UALink：无需反思的内存语义

UALink 联盟的互连采用内存语义操作（加载/存储），而非 RDMA 基于动词的模型（发送/写入/读取/原子操作）。操作使用固定大小的 640 字节微片，从而消除以太网成帧开销，以及会引入可变延迟的数据包聚合。

UALink 的内存语义比 OAE 的模型更接近 RDMA 的模型：加载/存储操作意味着远程内存系统参与了该操作，而不只是远程网卡。但 UALink 面向纵向扩展互连网络（机架组内、机架内）而设计，并沿用了其成员公司现有架构的缓存一致性模型。它没有发布关于多 flit 原子操作保证的规范，也没有反射机制或语义摘要。它比 RDMA 更快、更简洁，但仍是一种 FITO 技术：数据向前传输，返回路径只携带传输层状态。

7.4 汇总表

Gap	rdma	CXL	NVLink	UALink	oae
$T_4 \to T_5$ (可见性)	open	关闭	关闭	未知	关闭
$T_5 \to T_6$ (语义)	open	open	open	open	关闭
原子性边界	8 B	64–128 B	128 B	640 B	事务
反映阶段	缺席	缺席	缺席	缺失	强制
SDC 检测	none	none	none	none	反射不匹配

表 1：互连技术的时间间隙分析。只有 OAE 能弥合 T_5 到 T_6 的差距通过其强制性的反映阶段。

8 总结与第四部分预览

本文论证了， RDMA 的完成语义在硬件层实现了 FITO 类别错误。完成信号——发送方的完成队列中发布的 CQE——在时间阶段 T_4 报告成功，此时语义箭头尚未确立。 T_4 （完成）与 T_6 （语义一致）之间的差距，正是意义遭到破坏的空间：多字段不变量被违反，静默数据损坏扩散而未被检测到，完成信号还会主动误导应用程序，使其对事务状态产生错误认知。

这四项案例研究表明，这并非理论上的担忧，而是最大规模部署中的运营现实。Meta 的 24,000 GPU集群、Google 的多租户数据中心、Microsoft 的异构网卡部署，以及苏黎世联邦理工学院的 SDR-RDMA 工作，都记录了同一模式： RDMA 完成；意义丢失。

CXL、NVLink 和 UALink 各自解决了问题的一部分——缓存一致性、基于信号的排序、内存语义——但无一提供能够弥合 T_5到T_6 差距的反映阶段。完成谬误之所以贯穿整个互连领域，是因为它嵌入在每项技术所共有的 FITO 假设之中：信息向前流动，而返回路径只是辅助。

第四部分（Borrill，2026c）将完成谬误的后果追溯到数据中心之外，进入人们日常使用的系统：悄然删除内容的文件同步、重排因果关系的电子邮件，以及重构而非回放的人类记忆——每一个都是语义箭头经常遭到违背的系统，其后果并非由网卡和图形处理器承担，而是由依赖它们的人来承担。

参考文献

Borrill, Paul。《时间的语义箭头，第一部分：从爱丁顿到以太网》。DÆDÆLUS，2026。

Borrill, Paul。《时间的语义箭头，第二部分：开放原子操作以太网链路的语义》。DÆDÆLUS，2026。

Gangidi 等人。“《用于Meta规模分布式AI训练的以太网上远程直接内存访问》。”*ACM SIGCOMM*会议论文集，2024年。

Singhvi, Arjun 等人。“1RMA：重新构想多租户数据中心的远程内存访问。”*ACM SIGCOMM*会议论文集，2020年。微软研究院。“Azure规模的远程直接内存访问：经验与教训。”技术报告，2023年。Khalilov, Ilyas, Shen 等人。“SDR-RDMA：面向行星级远程直接内存访问通信的软件定义可靠性架构。”*SC25*会议论文集，2025年。 ARXIV:2505.05366。Ma, Yandong 等人。“深入探讨远程直接内存访问内存语义。”*IEEE*国际集群计算会议，2021年。Ziegler, Tobias 等人。“使用单边远程直接内存访问实现正确、高效且可扩展同步的设计准则。”*ACM*数据管理（SIGMOD），2024年。Dragojevic, Aleksandar 等人。“FaRM：快速远程内存。”第*25*届研讨会论文集

m

操作系统原理（SOSP），2015年。Compute Express Link联盟。“CXL 3.0规范。”2023年。开放计算项目。“人工智能中的静默数据损坏。”OCP白皮书，2024年8月。开放计算项目。“服务器组件弹性规范，v1.0。”2024年9月。Hoefler，Torsten 和 Roweth，Duncan 等。“数据中心以太网与RDMA：超大规模环境中的问题。” ARXIV:2302.03337，2023年。Shannon，Claude E. “通信的数学理论。”贝尔系统技术期刊，27(3):379–423，1948年。Herlihy，Maurice。“无等待同步。”*ACM*编程语言

s

与系统，13(1):124–149，1991年。Kalia，Anuj、Kaminsky，Michael 和 Andersen，David G。“高性能远程直接内存访问系统设计指南。”*USENIX ATC*论文集，2016年。“远程直接内存访问的语义：TSO架构上的RDMA操作模型与声明式模型。”*ACM*编程语言论文集，2024年。Borrill，Paul。“语义时间之箭，第四部分：事务为何失败。”DÆDÆLUS，2026。