不完整证据下健全的分布式运行时验证

Nikolaos Kekatos Clone Systems 拉纳卡，塞浦路斯 n
kekatos@clone-systems.com

Antonio Anastasio Bruto da Costa Pure SiliconSolutions 伯明翰，英国
antoniobdacosta@gmail.com

Alexios Lekidis 色萨利大学 拉里萨，希腊
alekidis@uth.gr

Pabitra Mitra 印度理工学院克勒格布尔分校 克勒格布尔，印度
pabitra@cse.iitkgp.ac.in

Hosein Nazarpour 亚马逊 巴黎，法国
nazarpur@gmail.com

Priyanka Sinha Docyt 加尔各答，印度
priyanka.sinha.iitg@gmail.com

arXiv:2611.32868v1 [cs.DC] 24 Nov 2026

摘要

分布式运行时验证通常需要结合多个节点的证据，以检查那些无法在本地评估的性质。然而，当消息发生延迟、丢失或被故意抑制时，验证器可能无法判断违规是否发生，或相关证据是否缺失。这会带来错误报告合规性的风险。我们研究了这些条件下实现可靠监控的要求。首先，我们证明，当对手能够抑制证据时，有截止期限的监控器无法同时做到可靠、可判定且可用。因此，监控器必须能够弃权。随后，我们区分了不完整证据的两种来源。序列号和定期报告可以识别缺失事件，而检测静默流则需要时序假设。我们还考察了成员变更和协调器故障如何影响判定健全性与恢复能力。我们使用 MQTT 和 NATS JetStream，在三层监控架构中实现了这些机制。针对由大语言模型规划的机器人群开展的实验展示了各种故障模式，说明了所提出的机制如何防止在缺乏依据的情况下作出合规判定，并测量了其运行时开销和通信开销。

CCS 概念

• **计算理论** → **分布式算法**；• **软件及其工程** → 软件验证与确认。

关键词

分布式运行时验证、故障检测器、异步性、分区容错、网内聚合、机器人群

1 引言

运行时验证会利用系统在执行过程中发出的事件，检查正在运行的系统是否符合形式化规范 [33]。当系统是分布式的，决定某项要求是否满足的事件来自多个节点，而任何单个节点看到的事件都不足以独立得出判定 [5, 11, 21, 36]。必须在多个服务之间执行保留策略的数据平台就是一个例子。另一例是由多辆自动驾驶车辆组成的车队，它们必须遵守适用于整个车队的限制 [10, 18, 22]。

通常的做法是将每个节点的事件转发给中央验证器，并在那里评估全局规范。判定的可靠程度取决于到达验证器的证据。消息可能被延迟、丢失或蓄意抑制；从验证器的角度看，一个空闲节点与一个其报告从未到达的节点并无区别 [2, 4, 26]。在安全监控中，这一间隙尤为重要，因为对手想要隐藏的事件，恰恰是那些会揭示违规的事件。

在本文中，我们考虑这样一种对手：它能够干扰证据的传递，但无法伪造或篡改受监控节点产生的事件。因此，确实到达的证据可以证明违规。证明合规性则更加困难。一条未显示违规的已接收轨迹并不能说明系统合规，因为相关事件可能仍然缺失；因此，监控器在作出确定性判定之前，需要知道其证据是否完整。

已有工作涵盖去中心化监控、进程崩溃情况下的监控，以及对不完整或乱序数据流的评估 [2, 4, 7, 14, 20, 26]。故障检测器理论描述了异步系统中的进程能够了解到彼此的哪些信息 [8, 9]。我们关注的是，这些问题对证据传递机制本身提出了什么要求，即哪些故障根本能够被检测到、现有证据何时支持可靠的判定，以及节点与验证器还必须交换哪些信息。图 1 展示了证据流水线以及我们考虑的故障模式。

本文作出了三项贡献。

- **可靠监控的局限性**。对于我们所考虑的跨节点属性，一旦对手能够抑制证据，有截止期限的数据流监控器就不可能同时具备可靠性、决定性和可用性（定理 3.1）。要保持可靠，监控器就必须被允许返回不确定判决（推论 3.2）。
- **证据完整性与故障检测**。缺失事件和静默流是不同的故障，需要不同的检测器（引理 4.1 和定理 4.3）。我们给出了可问责的否定判决的报告成本（命题 4.4），并表明成员变更必须作为证据传播（定理 6.1）。
- **协议设计与恢复**。三层监控

已被第 28 届分布式计算与网络国际会议（ICDCN 2027）录用。会议将于 2027 年 1 月 4 日至 7 日在印度 Surathkal 举行。本文为作者版本，依据 CC-BY 4.0 发布。

协议实现了这些机制，我们证明其可靠性（命题 5.1）。随后我们摧毁协调器，并研究序列编号、判定标记和确认消息如何决定哪些内容得以保留（定理 6.3）。

该评估在 MQTT 和 NATS JetStream 上运行由大语言模型规划的监控群体的监控器。实验涵盖缺失证据、节点静默、成员变更、协调器销毁和报告带宽。在故障测试活动中，该协议没有产生任何静默误报平安，而尽力而为的协调器产生了三次。对在线代理和合成轨迹的测量，展示了运行时开销和可扩展性。

第 2 节确定系统模型和监控属性。第 3 节和第 4 节阐述理论要求，第 5 节介绍协议，第 6 节讨论成员变更和协调器销毁。第 7 节报告实验结果，第 8 节回顾相关工作，第 9 节作出总结。

2 系统模型与问题定义

节点、数据流与验证器。系统由一组节点 $A = \{a_1, \ldots\ldots, a_n\}$ 和一个验证器 v 组成。每个节点观察本地活动并发出事件。事件是一个元组 $e = (sq\ \tau,\ a,\ k,\ attrs)$，其中$sq$是每节点单调序列号，$\tau$是沿节点序列不递减的时间戳，$a \in A$是发出事件的节点，$k$是事件类型，$attrs$是类型化属性。跨节点顺序由发出时间戳$\tau$确定；若时间戳相同，则由 $(a,\ sq)$ 打破平局。因此，T 中的每个事件都有确定的顺序，包括那些在到达协调器之前被对手移除的事件。第 5 节所述的可信摄取顺序是验证器对该顺序的可观测代理，两者不一致时可以检测出来，而不会被悄然吸收。步骤间隔小于时钟偏差界的链不在此类范围内。a 的本地流是其发出的事件序列，而真实轨迹 T是这些事件的并集。验证器接收接收的数据流$R \subseteq T$，监控按时钟周期进行：每个时钟周期，监控器都必须针对每个受监控的性质，从下文定义的取值域中给出一个判定。

通信与对手。消息传递是异步的：消息可能被任意延迟，但除非被丢弃，否则不会永远延迟。对手可以在任何链路上丢弃、延迟和重排消息，也可以通过干扰节点或将其与验证器分区，使任何节点无限期地静默。当验证器未持有某个已发出事件时，该事件处于缺失状态；如果该事件从未送达，则视为丢失；如果验证器在某个窗口内未从节点收到任何消息，则该节点在该窗口内是静默的。在核心模型中，节点已认证且是非拜占庭式的：节点不会伪造或篡改其发出的事件，因此$R \subseteq T$始终成立：对手可以压制或延迟证据，但无法修改事件内容。完整性故障，即节点少报告自身活动的情况，需要独立佐证见证，并在第6.1节中处理。

单调见证性质类。 如果存在一个有限见证集合 的集合 $\mathrm{W}(P)$，使得 P在轨迹 T上成立，当且仅当 $W \subseteq T$对于某个 $W \in \mathrm{W}(P)$ 成立，那么称轨迹上的性质 P 具有单调见证 性质。等价地，一组有限事件即可确立 P，且添加事件不会使已有见证失效。例如，存在性合取 （关于同一对象的几类证据）、阈值 （在一个窗口内，至少有 m 个不同节点满足某个谓词）、求和 （节点上的聚合值超过预算）以及有序链 （跨节点按规定顺序执行的步骤）。全文中，P 表示一个事件条件，因此当 P 成立时，监控器触发 。

缺失声明。覆盖义务需要额外证据。某个扇区未被观测到这一声明，对于普通事件轨迹而言并不具备单调见证属性：在相关窗口内，后续关于某次观测的报告可能使该声明失效。我们通过将第4节的序列报告视为事件来处理这一问题。所有成员针对该窗口作出的承诺，可以提供一个有限见证，证明没有发生任何观测。承诺将序列高水位标记绑定到时间前沿（第4节）。每个承诺对于其节点和窗口都必须是最终的，这意味着在已承诺的序列前缀内不得出现后续事件。在非拜占庭假设下，良构扩展不能同时包含这些承诺以及来自已承诺区间的额外观测。因此，对于遵守这些承诺的良构扩展，单调性成立，而不是对事件的任意超集成立。定理 3.1针对单调见证属性进行陈述；覆盖义务仅通过这一构造得到覆盖，而该构造还要求窗口内的成员资格是完整的（第6.1节）。

判定域。 判定是一个二元组（s，c）。安全轴为$s \in \{\text{violation}、\text{no_violation}、\text{unknown}\}$；完整性轴为$c \in \{\text{intact} \succ \text{degraded} \succ \text{incomplete} \succ \text{unavailable}\}$，按照验证器能够核算的证据量排序。组合判定的完备性是其组成流完备性的交，而降级规则将（no_violation c）且 $c \prec \text{intact}$映射为unknown：只有在支持负面发现的证据得到核算时，负面发现才会被如实报告。正面发现无需进行此类处理，因为根据 $R \subseteq T$，已经到达的见证就是已经发生的见证。我们将输出unknown称为弃权。

监控器不得做什么。 令预言机判定为在真实轨迹 T上计算得到的判定。我们关注的故障是由被抑制的证据导致的错误的合规判定。

定义*2.1*（静默误报平安）。 如果监控器在某个时钟周期输出（no_violation，intact），而该时钟周期的预言机判定为violation，则称监控器发生了一次静默误报平安（sfac）。我们也将其记作静默全清除。

sfac 比漏检更糟糕，因为对于任何使用该判定的人来说，它与真正的无故障系统无法区分：它会将攻击转化为肯定性保证。现在，我们可以为第3节表明无法同时实现的三个性质命名。

定义*2.2*（可靠、决定性、可用）。如果监控器从不提交 sfac，并且当预言机判定为无_违规时从不报告违规，则称其是可靠的；如果从不输出未知，则称其具有决定性；如果在每个时钟周期的截止期限前都作出响应，则称其可用。如果在没有漏掉任何事件时，它会在有界延迟内输出预言机判定，则称其是活跃的。

活性排除了始终输出未知的平凡监控器，而这种监控器既可靠又可用。

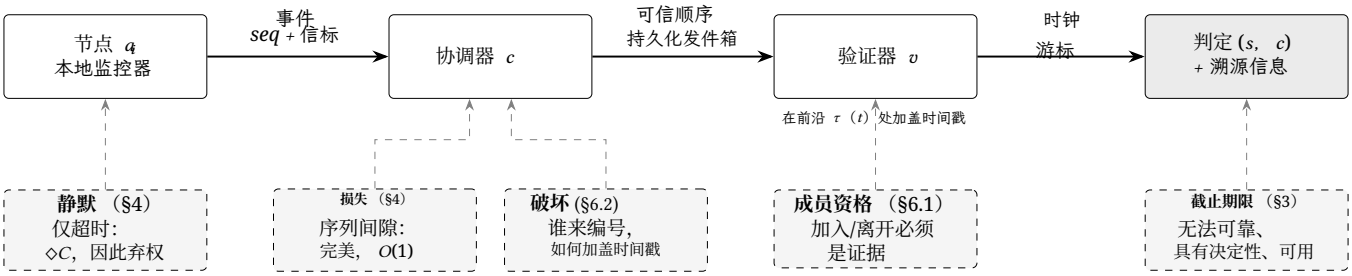


图1：证据流水线，以及每项结果适用的位置。证据从左向右流动，虚线框表示证据无法送达，或其含义与验证器所理解的不同的四种情况。右侧的截止期限是导致必须弃权的障碍。

可靠性具有双向性：它既禁止伪造一切正常，也禁止伪造警报，因此监控器不能通过在每个间隙上报警来逃避 sfac。

3 可靠分布式监控的局限

在第2节的模型下，图2展示了两种执行，它们在验证器处产生相同的观测，却要求不同的预言机判定，因此没有任何监控器能够同时做到可靠、具有决定性且可用。

定理 3.1（三难困境）。设 P 是具有单调见证性质的性质，且其见证集要求包含来自至少两个节点的事件。在第2节的模型下，不存在能够同时做到可靠、具有决定性且可用的 P 监控器。

证明。固定一个 $\subseteq$-极小见证集 $W \in \mathrm{W}(P)$（它存在，因为见证集是有限的），以及一个贡献了事件 $w \in W$ 的节点 a_i。考虑两种执行：节点发出 W 中的事件，除此之外不做任何事。在 E_1 中，每个节点都发出其在 W 中的事件，因此 P 在真实轨迹上成立，预言机判定为违规；对手丢弃携带 w 的消息，或者等价地，在 a_i 发出 w 后立即使其静默。在 E_2 中，节点的行为与 E_1 相同，只是 a_i 什么也不发出，因此真实轨迹为 $W \setminus \{w\}$。其中包含的任何见证集都将是 W 的真子集，这与极小性相矛盾，因此预言机判定为没有_违规；不会发送携带 w 的消息，并且 a_i 不会被置为静默。

验证器的本地状态在 E_1 和 E_2 中的每个有限时刻都完全相同。在两种情况下，它都恰好收到了 $W \setminus \{w\}$ 中的事件，而没有从 a_i 收到任何信息；异步性意味着，从该状态无法区分“w 尚未到达”和“w 永远不会到达”：任何能够区分二者的消息要么未被发送（E_2），要么被丢弃（E_1）。

设 d 为一个时钟周期截止期限。可用性要求在 d 之前给出判定，而决断性禁止未知，因此输出只能是违规或无_违规；由于本地状态相同，两次执行中的输出也相同。如果输出为无_违规，那么在 E_1 中，它是针对预言机判定为违规的（无_违规 c），而可靠性迫使 c =保持完好，因为否则降级规则会产生未知，这与决断性相矛盾；这就是一个 sfac。如果输出为违规，那么在 E_2 中，监控器针对预言机判定为无_违规报告了违规，这与可靠性的后半部分相矛盾。无论哪种情况，可靠性都会失效。□

推论3.2（三个值是必要的）。任何针对此类 P 的可靠且可用的监控器，其判定域至少包含三个可区分的输出，其中一个输出会同时出现在预言机判定为违规的执行和预言机判定为无违规的执行中。_

定理未涵盖的情况。 三个假设支撑着这一结论。局部性：如果每个见证集都位于同一个节点内，则无需传输证据。异步性： 在有界延迟下，超过界限后的静默具有信息量；第4节将这一例外形式化为检测器。截止期限：只需最终正确的监控器可以等待，而任务截止期限不允许这样做。

与 FLP 和 CAP 的关系。 该结果并非由二者得出。FLP 讨论的是存在一个故障进程时的异步共识 [19]；而在这里，只有一个验证器作出决定，阻碍在于该决定依赖于由对手控制的证据。CAP [23] 在分区情况下以一致性换取可用性；三难困境则以时钟周期截止期限所扮演的分区容错角色，在可靠性 与决断性之间进行权衡。

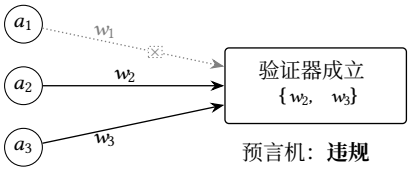
4 证据完备性与故障检测

定理 3.1 证明，可靠且可用的监控有时需要弃权。现在我们考察何时能够得出确定性判定，以及对于缺失事件和静默流，所需的检测机制有何不同。

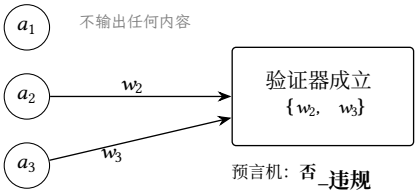
引理 4.1（间隙可证明缺失证据）。令每个节点为其事件进行连续编号，并令验证器记录每个数据流中已持有的编号。如果数据流传递了 $j+2$，而未持有 $j+1$，则 $j+1$ 已被发出但处于缺失状态。验证器不会产生误报。对于每个数据流的先进先出传递，其状态是已接收的最高编号，即 $O(1)$；在发生重排序时，其状态是乱序编号的缓冲区，即 $O(w)$，对应于重排序窗口 w。

证明。编号是连续的，因此只有在 $j+1$ 已被发出的情况下，$j+2$ 才会存在。间隙只能由已发出的事件造成，而 $R \subseteq T$ 意味着验证器从未凭空生成间隙。□

间隙表明证据缺失，而非已经丢失：该事件可能仍在传输中，在无界延迟的情况下，两者无法区分。数据流静默尾段中的损失不会留下间隙；命题 4.4 解决了这种情形。



(a) E_1**：三个见证均出现，且** w_1 **在传输过程中被干扰。**



(b) E_2**：** w_1 **永远不会发生。**

图 2：验证器无法区分的两种执行。验证器在两种情况下都持有$\{w_2,\ w_3\}$**，因此，每个时钟周期都必须给出回答的监控器会在其中一种情况下出错（定理 3.1），而可靠性要求使用第三种判定值。**

引理 4.1 给出了一个完美的不完备性检测器，其含义符合故障检测器文献 [9]：它绝不会错误地怀疑，并最终会怀疑每个间隙。命题 4.4 解决了静默尾部情形，并对两种检测器的代价进行了评估。

定义 *4.2*（证据完备性检测器）。对于节点 a 和窗口 w，检测器 C 报告 a 是否在 w 中发出了验证器未持有的事件。若 C 从不出错，则称其为完美；若仅在有限前缀内出错，则称其为最终完美，$\Diamond C$。

部署通过两种机制近似实现 C。编号和周期性序列报告可揭示缺失事件；在部分同步条件下，针对信标的超时机制可揭示不可用流。C 严格强于崩溃检测器 [9]，，后者并不足够，因为对手可能会丢弃发往仍正常运行的节点的链路上的消息。

定理 4.3（静默需要最终完美检测器）。在异步模型中，仅凭事件流无法区分静默节点和空闲但遵循协议的节点。对于单调见证属性：(i) 给定 C，存在一个可靠的监控器：只要 C 报告没有缺失事件，它就会返回确定性判定；只有在报告存在缺失事件时才弃权；(ii) 如果仅给定 $\Diamond C$，受时钟周期截止期限约束的监控器就无法同时做到可靠且具有决定性，因此必须弃权；以及(iii) 对于单流性质“该节点在当前窗口中发出了一个事件”，可靠、具有决定性且受截止期限约束的监控器，就是该节点的完美 C。

证明。*(i)*设 R 为接收流。如果 P 在 R 上成立，则报告违规；鉴于 $R \subseteq T$，该报告是可靠的。否则，如果 C 报告任何贡献节点都没有缺失事件，R 和 T 在这些流上达成一致，因此 P 在 T 上不成立，报告_违规是不正确的。如果 C 报告存在缺失事件，则它可能补全一个见证，此时监控器弃权。检测器无需指出缺失的是哪个事件。*(ii)* 在任何有限时间点，$\Diamond C$ 的怀疑都可能处于其不可靠前缀中，因此将静默归因于故障或闲置可能是错误的。无论如何作出决定都会违反可靠性的一半，而拒绝作出决定则会违反决断性。*(iii)* 在每个截止期限，对该性质作出可靠且具有决断性的判定，就是正确报告该节点是否拥有未计入的证据。反向命题涉及单流情形，并不将 C 置于检测器层级中。□

表 1 总结了这一情况。组合判定的可问责性取决于其最不具可问责性的贡献流，因此完整性构成一个格，而组合取其交。

表 1：故障类别在可检测性、完备性值和成本方面各不相同。这两种机制不能相互替代（消融实验，第 7 节）。

故障	检测器	判定	Cost
事件丢失	完美，序列间隙	不完整	$O(1)$/数据流
静默	$\Diamond C$，超时	不可用	$O(n)$ 活性状态
重排序	完美、可信摄取顺序	降级	O(缓冲区)
成员资格更改	仅当已报告时作为证据	未知开启阈值	$O(1)$/更改

对设计的启示。采用序列编号但没有存活信号的部署可以检测缺失事件，却无法识别无法解释的静默。相反，采用基于超时的检测但没有序列编号的部署，可以标记在超过超时后仍保持静默的数据流，却无法检测序列间隙。因此，可靠性保证必须明确哪些检测机制可用。第 7 节通过依次移除各个机制，评估每种机制的贡献。

信标承诺的内容。来自节点 a 的信标携带序列高水位标记 h 和时间前沿 u，即其发出的时间。由于节点已认证且为非拜占庭式，并且时间戳沿节点序列不会递减，因此每个时间戳不晚于 u 发出的事件 a，其编号均不大于 h；此后发出的事件则不会携带不晚于 u 的时间戳。当验证器持有截至 h 的所有编号后，就会将截至 u 的区间视为 a 已完整。第 5 节中的前沿 $\tau(t)$ 是各贡献流对应 u 的最小值。

这两个检测器的成本。二者都通过同一条简短消息实现。报告策略规定，对于每个节点和窗口，该节点要传输什么。

命题 4.4（可问责的否定判定的成本）。(a) 如果某节点 a 在窗口 w 内不发送任何内容，则对于某个性质而言，只要 a 能在 w 中提供一个原本未被观测到的见证，任何可靠监视器都不能返回确定的否定_违规判定。因此，可问责的否定判定每个节点每个窗口需要 $\Omega(1)$ 条消息。(b) 每个节点每个窗口发送一次该节点发出的最高序列号以及一个存活信标，无论事件速率如何，都能保持可靠性。检测延迟至多为一个窗口加上

检测器超时。

证明。（*a*）如果在 a 内不发送任何内容，则无论 a 是否发出了见证，验证器的状态都相同。这正是定理 3.1 的不可区分性，因此，可靠性要求返回未知。（*b*）节点发出的最高序列号，即其高水位标记，限定了它在窗口内发出的内容。因而，验证器缺失的任何事件都会表现为相对于该序列号的短缺。这也涵盖静默尾段中的损失，而根据引理 4.1，这种损失不会留下可供检测的间隙。信标提供了定理 4.3 所述的超时检测器：如果超过超时期限仍未收到信标，则将数据流标记为不可用。无论事件速率如何，这两个检测器在每个节点每个窗口发送一条消息的情况下都具有活性。□

命题 4.4 区分了证据核算的成本与违规检测的成本。证据核算要求每个节点在每个窗口内承担恒定的报告开销，而检测违规则要求相应的见证事件到达验证器。减少见证流量会限制检测能力。对于无法确认的违规，验证器可以返回未知，而不必给出缺乏依据的合规判定。

5 监控协议与可靠性

定理 4.3 明确了实现可靠监控所需的检测要求。我们将这些要求落实为一种三层协议，该协议结合了基于序列的证据核算，以及基于超时的静默流检测。该协议以我们早期工作的分层监控架构为基础 [28–30], ，后者为本协议提供了工程实现基础。与早期工作不同，本文从故障检测器的要求出发分析监控机制，并阐明这些机制对可靠判定的作用。我们还提出了处理成员变更（第6.1节）以及在证据前沿为判定添加时间戳标记（第6.2节）的规则。

层级。每个节点运行一个本地监控器，判定规范中属于本节点的部分，并在自身序列中为事件连续编号。组协调器 摄取附近节点的数据流，为每个事件分配单调摄取顺序和持久标识符，在确认之前将其追加到持久化发件箱，然后向上转发。验证器 通过自身的持久游标消费该输出，并判定跨节点性质。聚合是不完整的：协调器会解析其组内见证集所涵盖的任何性质；否则，它会连同所聚合数据流的完备性一起转发该组的投影贡献，从而使完备性随证据传播 [34]。

作为检测器的机制。每个节点采用连续编号并至少一次摄取，实现了引理 4.1 中的完美间隙检测器：已摄取数据流中的编号间隙，证明已发出的证据未被保留，并标记该数据流为不完整。协调器分配验证器所使用的摄取顺序。该顺序会与第2节定义的发出顺序进行比较。一旦检测到差异，就将受影响的判定标记为降级，以避免基于顺序错误的轨迹评估性质 [32]。驻留于验证器的时钟独立于节点流量推进监控时间，实现了定理 4.3 中的超时检测器：

时钟显示本应报告却尚未报告的数据流，会被标记为不可用。

判定的盖章位置。尚未触发的检测器会留下一个窗口，在此期间，即使某个数据流处于静默状态，仍会被视为已计入。因此，协议会将每个判定盖章在证据前沿 $\tau(t)$处，即每个贡献流都由信标明确覆盖到的最新时间点，而不是挂钟时间。否则，在超时窗口之前的整个期间，协议都会报告（无_违规，完好）（定理 6.3(ii)）。

时钟必须驻留在验证器中，因为由受监控节点驱动的存活信号会与它们同命运。让某个节点静默的对手也会让它的心跳信号静默；此时，空数据流看起来就像数据流缺失，除非有独立于受监控节点群体的信号将二者区分开来。

隔离与背压。每个事件都带有全局稳定的标识符，因此在评估前会对重放进行去重，重放的见证也不会抬高计数。达到保留上限时，部署要么静默丢弃最旧的数据，要么明确拒绝新证据；只有第二种做法能保留引理 4.1 所需的序列信息。

命题 5.1（协议可靠性）。当判定结果以证据前沿为时间戳时，该协议绝不会提交错误判定：它报告为（无_违规，完整）的每项判定结果，都与预言机针对其时间戳所对应区间给出的结果一致。当没有事件缺失时，该协议也具有活性：τ（t）推进，并且在延迟 $t-\tau$（t）后，判定结果将明确确定。

证明概要。仅当每个贡献流都完整时，才会将判定报告为（无_违规，完整）。这要求不存在未解决的序列间隙、顺序违规或由时钟标记的静默。假设某个违规见证已发出，却未出现在验证器的证据中。序列间隙会检测已计入的数据流中缺失的事件，而排序机制会检测到达顺序与已提交的摄取顺序不一致的事件。如果某个节点的静默时间超过超时阈值，验证器就会标记其数据流。

另一种情况是，某个节点已停止报告，但尚未超过超时阈值。这通过在证据前沿 $\tau(t)$ 为判定添加时间戳来处理。每个贡献流都必须通过其序列报告和信标，确认其覆盖范围已达到该前沿。因此，完整的组合可证明验证器持有在 $[0,\tau(t)]$, 内发出的所有相关事件，并且验证器在此区间内的评估与预言机一致。

对于活性，假设贡献流保持活跃，其证据均已送达，且所需信标持续到达。证据前沿随后会推进，使验证器能够对已完成的区间作出确定性判定。如果某个数据流变得静默，前沿可能停止推进，从而避免作出缺乏依据的判定。最初表现为延迟不断增加，超时后则表现为未知判定。□

每个事件在各层级都会使协议增加 $O(1)$ 的工作量，而验证器会保留 $O(n)$ 个计时器，接近命题 4.4 所要求的最小值。

6 成员关系变更与协调器故障

监控基础设施中的变更和故障会影响运行时验证的可靠性。我们在第6.1节考察节点成员变更，并在第6.2节考察协调器销毁，重点关注它们对证据完整性、判定健全性和恢复的影响。

6.1 动态成员关系

到目前为止，该模型固定了节点集合。但现实中的节点集群并非如此：节点会加入、离开，也可能丢失。这会影响阈值和覆盖义务，因为它们的分母取决于成员资格。

记 M_t 为时刻 t 的真实成员关系，记 V_t 为验证器对其的视图。假设成员资格属于配置而非证据，因此 V_t 由带外提供，并且可能与 M_t 不一致。

一个 可达性检测器 报告当前哪些节点能够到达验证器。它不是 Chandra–Toueg 崩溃检测器，后者的准确性禁止怀疑正确进程，因此，对于一个陷入静默但实际正确的节点而言，可达性检测器就是验证器所拥有的可观测信息。

定理 6.1（必须报告成员资格）。如果成员变更未通过受保护路径传递，那么对于阈值和覆盖属性，每个可靠监视器都会弃权，即使是在应当给出确定性判定且所有幸存证据均完好无损的执行中也是如此。即使为验证器配备了完美可达性检测器，这一结论仍然成立。

证明。考虑一项覆盖义务：要求在一个窗口内观测集合 S 的每个扇区；节点 a_i 负责扇区 S_j。在执行 E_1 中，a_i 在窗口开始前正常离开，且没有其他节点接管 S_j；该扇区确实未被观测，预言机判定为违规。在执行 E_2 中，a_i 仍是成员，观测了 S_j，随后在其报告传递出去之前被对手静默；该扇区确实已被观测，预言机判定为不违规。_

验证器在两种情况下的状态相同：它持有除 a_i 之外每个节点的报告，而没有来自 a_i 的任何信息。完美可达性检测器也无法区分两者，因为在两次执行中，a_i 都永久不可达，而这正是它所报告的情况；对它而言，离开和静默是同一事件。预言机判定不同，因此任何确定性判定在两次执行中的某一次都会出错，而可靠性要求弃权。在 E_1 中，唯一的变化是该节点离开，覆盖问题已经确定。然而，监控器仍必须弃权，因为它无法区分 E_1 和 E_2。□

推论 6.2（将成员资格作为证据可恢复可判定性）。如果加入和离开事件与证据通过同一条持久化有序路径传递，E_1 和 E_2 就可以区分：在 E_1中，验证器会收到一条携带 a_i序列号的离开记录，因此，没有观察到 S_j 可归因于其离开，而非报告被抑制；对于 S_j的义务，离开并不能解除该义务，因此该义务会得到判定，而不是被搁置；在 E_2中则不存在此类记录，缺失的数据流会由第*4*节的检测器进行归因。随后，阈值判定和覆盖判定会在底层证据得到核算时准确确定。

表 2：成员资格在加入和离开情况下的表现，每行六次任务（在三个故障时刻各发生一次离开和一次静默）。作为配置，乐观性会在无声无息中消除实际覆盖缺口，而悲观性永远无法解决离开已定论的问题。作为证据，两种情况都不会发生。

成员资格视为	静默 全部通过	未解决 可知时	已决定 正确地
配置，乐观	24	0	0
配置，悲观	0	27	0
证据	**0**	**0**	**24**

测量。我们在部署中的扇区覆盖监视器上实例化该定理：四个扇区各由一个节点负责，因此成员资格是唯一可能导致不可靠的来源。一个节点要么离开，导致其负责的扇区无人覆盖；要么在仍然观察该扇区时被静默处理；验证器和可达性检测器无法区分这两种情况，但判定结果却不同。表 2 展示了定理 6.1 所预测的两种失败。将持续的静默视为离开，会导致 24 次静默全清除，即每个离开任务的每个评分时钟周期各一次。将静默视为成员仍在场，虽然不会错误地发出全部通过判定，却会留下 27 个未解决的时钟周期，而离开事件本已足以确定结果。在受保护路径上传递加入和离开事件后，这两种失败都会消失，而真正的静默仍会产生未知结果。每个任务中最多发生三次离开不会改变整体结构（分别为 11、8 和 5 个时钟周期）；数量减少仅仅是评分窗口缩短所致。

因此，成员资格必须像任何其他见证一样进行排序并持久化，因为丢失离开记录会重新造成该推论所消除的歧义。重新加入的节点必须重放其协调器扣留的积压记录，以便其缺席期间处于未知状态的判定结果得以确定。根据定理 3.1 的论证，完整性故障——即节点少报自身活动——只有在存在独立佐证见证时，才可归结为损失情形。

6.2 协调器销毁

到目前为止的每项结果都假设聚合层次结构能够存续。在作为动机的应用中，情况并非如此：协调器本身就是一个物理节点，而节点可能被销毁。我们考察协调器销毁对判定健全性的影响，重点关注序列号由节点还是由协调器分配。

模型扩展。协调器 $C = \{c_1, \ldots, c_m\}$ 服务于互不相交的节点组 $G_c \subseteq A$。除了先前拥有的能力外，对手还可以销毁一个协调器：使其永久停止运行，并且销毁它尚未转发的持久状态。销毁的影响严格强于容错监控中研究的崩溃恢复故障；在后者中，发件箱会保留下来，并在重启时重放，而被销毁的协调器不会重放任何内容。当 c 被销毁时，G_c 中的节点会重新归属到后继协调器 c'。

定理 6.3（协调器销毁与判定健全性）。假设某个协调器被销毁，且其尚未转发给验证器的证据也一并被销毁。

（i）协调器分配的编号。未转发证据的损失可能仍无法被观测到。在某些执行过程中，即使发生了违规，验证器仍会报告（无_违规，完整），这种情况可能持续到任务结束。

（ii）节点分配编号。当受影响节点的下一个事件或序列报告信标到达验证器时，缺失证据便会变得可观测。然而，在检测到损失之前，可能会发生 *sfac*。

（iii）判定戳记。带墙上时钟戳记的监视器无法对证据完整性尚未确定的区间可靠地发出确定性判定。在证据前沿 τ（t）处标记判定的监视器，则仅评估所有贡献流均已计入的区间，从而保持可靠性。

证据前沿 τ（t）是截至该时间、每个贡献流都已确认完整覆盖的最新时刻。在无故障运行中，假设处理及时，判定延迟 $t - \tau$（t）受限于信标周期加最大传输延迟。在持续干扰下，前沿可能停止推进，而延迟可能无限增长。此前确立的前缀判定对于其戳记区间仍然可靠，必须据此解读，因为它并未说明 τ（t）之后的执行情况，并且前沿停滞的时间越长，该判定就越过时。

证明。对于（*i*），设 G_c 的节点发出性质 P 的见证 w_1、……、w_k。协调器接收这些见证，在自己的序列中将其编号为 1、……、k，将 1、……、j连同 $j < k$ 一起转发，随后被销毁，并丢失 w_{j+1}、……、w_k；后继协调器开始自己的序列。验证器持有一个连续前缀和一个连续的后继流，而二者都无法表明被销毁的事件曾经存在，因为本应揭示该间隙的编号权限方也随这些事件一同被销毁了。在这些事件从未发出的执行中，观测结果无法区分，但预言机却不同。情况比定理 3.1 更糟：两条流看起来都完整，因此完整性轴报告完整，降级规则也不会触发：监控器不会弃权，而是自信地判定为清除。

对于*(ii)*，节点分配的编号不会随着协调器一同被破坏，因此当受影响节点下次通过后继节点报告时，验证器会在该节点自身的序列中看到一个间隙，并降低判定等级。但这种影响不会立即显现：在破坏发生到下一次报告到达之间，验证器持有的前缀在每条数据流中都是连续的，因此根据*(i)* 中的相同推理，它会报告（无_违规，完整）。这个时间段的上限由节点的报告间隔决定，而非任务持续时间，这正是它与*(i)*的不同之处；但这个时间段并非不存在。

对于*(iii)*，在 t 给出的带墙上时钟戳记的判定，是对截至 t的时间区间作出的断言；根据*(ii)*，验证器尚无法核算该区间的证据，因此定理 3.1 直接适用于此情形。相反，在 τ（t）处添加时间戳，则使该断言成为验证器能够支持的断言：信标明确覆盖了每条数据流，直至 τ（t），因此根据命题 5.1 的论证，在$[0, \tau(t)]$上的评估与预言机在同一区间上的评估一致。于是，监控器会为最新的证据完整前缀保留确定性判定，而不会发出 sfac。如果证据前沿停止推进，该判定对其带时间戳的区间仍然可靠，但会变得越来越滞后。破坏会以这种方式导致停滞 τ，继而表现为延迟不断增大，最终转为弃权，但绝不会错误地判定为无违规。□

推论 6.4（可恢复完备性与可观测排序的不同机制）。必须在部署中

表 3：协调器销毁，涵盖证据处于传输中的时间区间内的六个销毁时间。编号决定损失会隐而不见多久，时间戳标记决定没有确认时确定性判定能否保留，而确认决定损失能否恢复。

编号 / 时间戳标记	Ack	运行时使用 一个 sfac	静默窗口	判定已解析
协调器，挂钟时间	hop	6/6	22	0/6
协调器，挂钟时间	e2e	6/6	7	6/6
节点，挂钟时间	hop	6/6	7	0/6
节点，挂钟时间	e2e	6/6	7	6/6
节点，前沿	hop	**0/6**	**0**	0/6
节点，前沿	e2e	**0/6**	**0**	**6/6**

在协调器被销毁后恢复完备性并给出明确的有序判定结果，需要两种机制：用于完备性核算的端到端节点序列号，以及用于评估的确定性可观测顺序；在本协议中，这一顺序就是协调器分配的摄取顺序。两者不能互相替代。节点序列号可以定位缺失证据，但本身无法确定顺序；由协调器分配的顺序则可以确定顺序，但协调器一旦丢失，该顺序也无法保留。

这些结果表明，证据完整性和事件顺序需要 **अलग**的机制。节点分配的序列号支持完备性核算，而协调器分配的摄取顺序则为事件提供确定性的排序。将这些功能合并到一种编号机制中，在协调器销毁的情况下无法同时提供这两项保证。本协议所构建于其上的 fabric 会在协调器侧车 [28–30]，处分配事件标识符；根据定理 6.3(ii)，该机制在崩溃恢复下是可靠的，但在破坏情况下并不可靠；而切换为节点自身的编号，并将其端到端传递，则无需额外消息成本。

推论 6.5（确认限制了可恢复的内容）。如果节点在收到协调器的确认后丢弃缓冲证据，那么该协调器的破坏会摧毁那些已不再存在于任何位置的证据，因此判定永远无法得出结果。因此，恢复要求节点保留某个事件，直到该事件得到验证器确认，或得到其存活独立于节点协调器的层级确认。

测量。我们针对第 7 节中 ISR 部署的监控器，检验了定理 6.3 的三个部分，因此只对传输、销毁故障和标记规则进行建模。一次任务执行过程中，协调器遭到破坏，其尚未转发的发件箱也随之丢失；我们将编号权限方和标记规则与确认策略进行交叉组合（表 3）。协调器分配的编号使损失在任务剩余期间无法被察觉，形成长达 22 个时钟周期的静默窗口；节点分配编号则将其缩短至 7 个时钟周期，即受影响节点的报告间隔时间，但并未消除它，这对应于部分 *(ii)*。早期的一种表述认为节点编号已经足够；测量结果表明事实并非如此。只有前沿标记才能在每次运行中消除此问题，平均延迟为 4.3 个时钟周期，正如部分 *(iii)* 所预言的那样。确认策略决定了能否恢复：逐跳确认时，被破坏的证据已不复存在，判定始终无法得出；端到端确认时，

存活节点会重放证据，每次运行都能得出判定。

在线代理上的情况。 我们在 Mosquitto 和 NATS JetStream 上重复了这三种配置，并通过 SIGKILL 终止协调器进程，同时清除其尚未转发的发件箱。前沿标记再次确保所有运行中均不会出现静默全清除；挂钟时间上的配置结果有所不同（6 次运行中有 4 至 6 次，窗口为 3 至 6 个时钟周期），因为实时交付时序决定了损失被隐藏的时长。

始终弃权的监控器虽然可靠，却永远无法给出确定性判定；前沿标记则能在有界延迟内，对已覆盖的前缀给出确定性判定。

后继协调器可以得出什么结论。 在时间 t 接管的协调器 c' 既不知道前任协调器接收了什么，也不知道它转发了什么。它会发布一个交接间隙标记，覆盖从前任协调器最后一个经验证器确认的位置到 t的时间段；在节点通过自身编号填补该间隙之前，验证器会将 G_c 中的每个数据流都视为不完整。跨越该时间段的判定将保持未知状态，并在存活节点从其最后一个端到端确认处重放后得以确定（推论 6.5）。因此，任务判定会在破坏发生处显示一个间隙，而不是给出缺乏依据的合规判定。

7 实验评估

设置。 测量来自一项部署，该部署通过在线代理监控由大语言模型规划的情报、监视与侦察蜂群中的四个平台：节点到协调器跃点使用 MQTT (Mosquitto)，而通往验证器的持久化跃点使用 NATS Jet- Stream；节点层则通过 ROS2 和 ArduPilot SITL [28–31, 35, 38] 进行测试。监控的义务是该任务的三项跨节点属性：禁止使用集体智能软件包、遵守全蜂群排放预算，以及仅在获得授权后进入敏感区域。故障测试活动包括无故障、丢失见证者和受干扰平台三种情况，共产生九起预言机事件。监控器、驱动程序和有限模型检查器将在论文接收后发布。

报告了五项实验：两项沿用该部署（基线比较和保证消融），另有三项新实验（表 2 和表 3 中的成员资格扫描与摧毁扫描，以及下文的带宽扫描）。新实验使用部署自身的监控器代码运行，仅对待测的传输、故障和设计选择进行建模。

三难困境及其两类故障。在整个测试过程中，逐节点防护机制在 9 起预言机事件中导致了 9 次静默错误全清除，尽力而为的中央监视器导致了 3 次，而第 5 节中的协议则一次也没有。防护机制的这一数字，正是定理 3.1 中局部性逃逸的逆向情形：每个见证集都横跨多个节点，因此没有任何本地监控器能掌握它；每个节点都符合要求，但系统整体却不符合。中央监视器导致的三次，则是三难困境本身：它能接收合并后的数据流，却无法报告其中一部分缺失；而在二值域中，唯一剩下的只能是乐观的答案。

每次撤去一个检测器，可以区分这两类故障。没有序列间隙检测时，协议在丢失见证者的情况下会提交一次静默全清除，在受干扰节点的情况下则一次也不会；没有驻留于验证器的时钟时，协议在丢失见证者的情况下不会提交，在受干扰节点的情况下则会提交两次。因此，每种机制都能应对其中一类故障，但都无法同时应对两类故障。

表4：在无故障、丢失见证者和受干扰节点运行中，三者所有计分时钟刻度上的可问责的否定判决成本；每个节点每个窗口发生4 个事件。舍弃见证会牺牲检测能力；舍弃 $\Theta(1)$ 求和会导致静默错误全清除，却省不了多少成本。

报告策略	B/节点/窗口全部通过	静默	弃权检测	lag
所有见证者 & 摘要	364	**0/69**	26/69	1
每 2 个见证者中的 1 个 & 摘要	192	**0/69**	69/69	从不
每 4 个见证者中的 1 个 & 摘要	108	**0/69**	69/69	从不
仅摘要	24	**0/69**	69/69	从不
所有见证者，无摘要	340	25/72	0/72	1
暗（未发送任何内容）	0	36/72	0/72	从不

可问责判定的成本。 表4比较了不同报告策略与已部署的软件包监视器的表现；判定均按照定理 6.3(iii) 的要求，在证据前沿处加盖时间戳，因此策略是唯一的变量。每份事件报告按 96B 计费；摘要消息携带节点的高水位序列号和存活信标，按 24 B 计费。

两种成本泾渭分明。所有保留摘要的策略都不会提交静默全部通过，无论带宽多低，即使每个节点每个窗口仅有 24B、也就是只传送摘要时，仍然如此。随着见证流量被削减，策略所付出的代价是检测能力下降：判定转为未知；在见证流量充足时，69 个计分时钟刻度中有 26 个如此，而一旦抽取删减使软件包的一部分从传输流中消失，69 个刻度便全部如此。另一种节省方式是保留所有见证并丢弃摘要；这样每个节点每个窗口可从 364 B 中节省 24 B，不到百分之七，却会提交 25 次静默全部通过。这两种故障都只能靠摘要捕获：丢失的见证者是受影响节点的最后一个事件，因此后续到达的内容都无法暴露这一间隙，只有高水位序列号能够做到；受干扰节点则没有可供察觉其缺失的信标。暗策略不传输任何内容，在软件包完成后的每个时钟周期都静默通过，这具体体现了命题 4.4 的 $\Omega(1)$ 界限。

将每个节点每个窗口的事件速率从 1 提高到 16 后，摘要仍为 24B，而见证流量则从 100B 增长到 1312B，因此其占比从 19% 降至不足 2%。保留摘要的策略在任何速率下都不会提交静默全清除；丢弃摘要的策略在每种速率下都提交 25 次。

协议的成本。 在在线代理中，判定路径在 n=1000 个事件上的中位数为 3.7 毫秒；在丢包率为 10% 的链路上，采用持久化重传且不丢失任何已接受证据时，中位数为 42 毫秒。任务监视器评估大致呈线性增长，在 1000 个节点时仍低于 2 毫秒。前沿标记不会增加延迟，但会引入判定延迟；在摧毁扫描中，该延迟平均为 4.3 个时钟周期，接近信标周期加上传输延迟。

有效性威胁。 这三组新增的扫描对传输过程进行建模，但调用的是已部署的监控器，因此性质语义、完备性核算和降级规则均与运行中的系统一致；时间是模拟的，字节模型则是名义上的。集群包含四个平台，规模测试结果是一条合成轨迹。核心模型假设节点均为经认证的非拜占庭节点（第6.1节），且具有节点级单调时间戳；实验脚本对此作了设定，

但并未强制执行。预言机本质上是在传输前轨迹上执行相同的性质代码。因此，这项比较测试的是证据通道，而非规范。

8 相关工作

分布式运行时验证。 分布式运行时验证已在多种监控架构和通信假设下得到研究 [5, 14, 21]。有些方法将判定分布到各个节点，并容忍监控器崩溃或对执行持有不同意见 [7, 16, 20]。时空性质也已得到监控 [1]。与本文最相关的工作研究了不完整观测、网络故障和乱序传递 [2, 4, 26, 37]，并为观测不可靠时如何评估性质提供了语义和算法。我们的工作关注的是更早的一步：在给出确定性判定之前，验证器必须先确认其证据完备。我们研究当对手选择性地抑制消息时，序列报告、时序信息和成员记录如何支持这一判定。针对不完整日志的策略审计 [22] 研究了相关问题，但其设置不同，对证据可用性的处理方式也不同。

故障检测器。 Chandra 和 Toueg 按准确性和完备性对故障检测器进行分类 [9] ，并找出了共识问题所需的最弱检测器 [8]。进程故障与我们研究的问题不同。与验证器之间的链路被阻断的节点仍会继续运行并发出事件，因此，知道节点仍然存活，并不能说明其证据是否已送达。我们需要的检测器能够报告某个节点是否发出了验证器尚未持有的事件。定理 4.3 指明了这种证据完备性检测器为实现健全监控必须提供什么，但并未将其纳入 Chandra–Toueg 层级。

不可能性结果与多值判定。分布式计算中的许多不可能性结果都基于不可区分性论证，FLP 便是其中之一 [19]。我们关注的是另一种情形：单个验证器必须在截止期限前给出答复，但相关事件可能已被抑制。运行时验证中的多值判定和可监控性已有充分研究 [6, 15, 33], ，其中不确定判决表示当前已观察到的轨迹尚不足以确定该性质。我们补充了弃权的第二个理由，即当前已观察到的轨迹可能并未包含所有相关事件。Fraigniaud 等人 [20] 证明，容错去中心化监控需要三种意见，因为监控器之间可能意见不一；而本文中的三种取值则源于有截止期限的验证器中的事件抑制。

监控工具与应用。 MonPoly [3] 和基于流的引擎 [12, 17, 25] 会对事件流上的时序规范进行评估。我们关注另一个问题：送达这类引擎的证据是否完备。我们的协议采用分层转发，其方式类似于 TAG [34], 中的网内聚合，另外，每个转发的摘要都会携带其背后各数据流的完备性。机器人运行时监控 [18] 和自主群体 [10] 方面的工作构成了应用背景，而对抗性场景则借鉴了基于大语言模型的系统及其动作组合所涉及的安全问题 [13, 24, 27]。

9 结论

在本文所阐述的条件下，即使证据不完整或遭到蓄意抑制，分布式运行时验证仍可保持可靠。对于我们研究的这类跨节点属性，不存在能够同时做到可靠、具有决定性且可用的异步监视器，因此，若要保持可靠性，监视器就必须在证据不足以支持确定性判定时弃权。

处理缺失证据需要多种机制，而且这些机制不能互相替代。序列编号和周期性报告能够揭示缺失事件。只有在时序假设成立时，才能检测到静默流。成员变更必须通过受保护的证据通道报告，而协调器销毁还会对事件编号、判定戳记和确认范围提出额外要求。

实验揭示了这些保证所需付出的成本。固定大小的序列摘要使证据核算不受事件速率影响。减少见证流量可能导致违规无法被检测到，此时可靠监视器会返回不确定判决，而不是给出缺乏依据的合规性判定。在证据前沿为判定添加时间戳，可以保证其可靠性，但代价是判定会滞后于当前执行。

这一切都建立在节点已认证且非拜占庭式，以及协议的排序和时间戳假设之上。如何将证据完备性检测与 Chandra–Toueg 故障检测器层级联系起来，仍有待研究。

致谢

本研究获得欧盟数字欧洲计划（CYBERGUARD，编号 101190251）的资助。

作者使用 Claude（Anthropic）协助审阅稿件并自动化实验。作者已核查所有内容，并对此承担全部责任。

参考文献

[1] Ezio Bartocci、Luca Bortolussi、Michele Loreti 和 Laura Nenzi。2017年。移动和空间分布式信息物理系统监控（STREL）。载于*MEMOCODE*。146–155。[2] David Basin、Felix Klaedtke 和 Eugen Zălinescu。2015。面向故障的分布式系统运行时验证。载于第 *35* 届 *IARCS* 软件技术与理论计算机科学基础年会（*FSTTCS*）（*LIPIcs*，第 *45* 卷）。590–603。[3] David Basin、Felix Klaedtke 和 Eugen Zalinescu。2017。MonPoly 监控工具。载于 *RV-CuBES*。[4] David Basin、Felix Klaedtke 和 Eugen Zălinescu。2017。无序数据流上的时态性质运行时验证。载于计算机辅助验证（*CAV*）。[5] Andreas Bauer 和 Yliès Falcone。2012。去中心化 LTL 监控。载于形式化方法（*FM*）（*LNCS*，第 *7436* 卷）。85–100。[6] Andreas Bauer、Martin Leucker 和 Christian Schallhart。2011。LTL 和 TLTL 的运行时验证。*ACM* 软件工程与方法学汇刊（*TOSEM*）20，4（2011）。[7] Borzoo Bonakdarpour、Pierre Fraigniaud、Sergio Rajsbaum、David A. Rosenblueth 和 Corentin Travers。2016。去中心化异步崩溃容错运行时验证。载于*CONCUR*。[8] Tushar Deepak Chandra、Vassos Hadzilacos 和 Sam Toueg。1996。求解共识的最弱故障检测器。*J. ACM* 43，4（1996），685–722。[9] Tushar Deepak Chandra 和 Sam Toueg。1996。可靠分布式系统的不可靠故障检测器。*J. ACM* 43，2（1996），225–267。[10] Soon-Jo Chung、Aditya A. Paranjape、Philip Dames、Shaojie Shen 和 Vijay Kumar。2018。空中群体机器人综述。*IEEETransactionsonRobotics* 34，4（2018），837–855。[11] Christian Colombo 和 Yliès Falcone。2016。使用全局时钟组织分布式系统上的 LTL 监视器。*Formal Methods in System Design* 49，1 2 2016 109 158 - ( ), – 。

[12] Lukas Convent、Sebastian Hungerecker、Martin Leucker、Torben Scheffel、Malte Schmitz 和 Daniel Thoma。2018。TeSSLa：基于时间流的规约语言。载于*SBMF*。[13] Edoardo Debenedetti 等。2024年。AgentDojo：用于评估大语言模型智能体提示注入攻击与防御的动态环境。载于*NeurIPS 数据集与基准*。[14] Antoine El-Hokayem 和 Yliès Falcone。2017年。去中心化规约监控。载于第 *26* 届 *ACM SIGSOFT* 软件测试与分析国际研讨会（*ISSTA*）。125–135。[15] Yliès Falcone、Jean-Claude Fernandez 和 Laurent Mounier。2012。运行时可以验证和强制执行什么？*STTT* 14，3（2012），349–382。[16] Yliès Falcone、Hosein Nazarpour、Saddek Bensalem 和 Marius Bozga。2021。分布式组件系统监控。载于组件软件的形式化方面（*FACS*）（*LNCS*）。Springer，153–173。doi:10.1007/978-3-030-90636-8_9[17] Peter Faymonville、Bernd Finkbeiner、Malte Schledjewski、Maximilian Schwenger、Marvin Stenger、Leander Tentrup 和 Hazem Torfah。2019年。StreamLAB：信息物理系统的基于数据流的监控。载于计算机辅助验证（*CAV*）。[18] Angelo Ferrando、Rafael C. Cardoso、Michael Fisher、Davide Ancona、Luca Franceschini 和 Viviana Mascardi。2020。ROSMonitoring：ROS 运行时验证框架。载于迈向自主机器人系统（*TAROS*）（*LNCS*，第 *12228* 卷）。Springer，387–399。[19] Michael J. Fischer、Nancy A. Lynch 和 Michael S. Paterson。1985。存在一个故障进程时分布式共识的不可能性。*J. ACM* 32，2（1985），374–382。[20] Pierre Fraigniaud、Sergio Rajsbaum 和 Corentin Travers。2014。分布式系统中容错运行时监控所需的意见数量。载于运行时验证（*RV*）。[21] Adrian Francalanza、Jorge A. Pérez 和 César Sánchez。2018。去中心化和分布式系统的运行时验证。载于运行时验证讲义。LNCS，第 10457 卷。176–210。[22] Deepak Garg、Limin Jia 和 Anupam Datta。2011。对不完整日志进行策略审计：理论、实现与应用。载于第 *18* 届 *ACM* 计算机与通信安全会议（*CCS*）。151–162。[23] Seth Gilbert 和 Nancy Lynch。2002。Brewer 猜想与一致、可用、分区容错的 Web 服务的可行性。*ACM SIGACT News* 33，2（2002），51–59。[24] Kai Greshake、Sahar Abdelnabi、Shailesh Mishra、Christoph Endres、Thorsten Holz 和 Mario Fritz。2023。并非你所期待的：通过间接提示注入攻陷现实世界的大语言模型集成应用。载于*ACM* 人工智能与安全研讨会（*AISec*）。[25] Klaus Havelund、Doron Peled 和 Dogan Ulus。2017年。基于二叉决策图的一阶时态逻辑监控。载于计算机辅助设计中的形式化方法（*FMCAD*）。[26] Yogi Joshi、Guy Martin Tchamgoue 和 Sebastian Fischmeister。2017年。基于有损轨迹的线性时序逻辑运行时验证。载于应用计算研讨会论文集（*SAC '17*）。ACM，1379–1386。doi:10.1145/3019612.3019827[27] Nikolaos Kekatos、Stylianos Basagiannis、Marinelio Chintri、Alexios Lekidis、Tom Nianios、Ioannis Seitoglou、Anastasios Temperekidis 和 Panagiotis Katsaros。2026年。工具使用型大语言模型智能体的形式化运行时验证：基于 AgentDojo 和 STAC 的离线同基准研究。载于*CPS*与物联网安全和隐私联合研讨会（*CPSIoTSec*）。arXiv:2610.09793。[28] Nikolaos Kekatos、Marinelio Chintri、Panagiotis Katsaros、Alexios Lekidis、Tom Nianios、Ioannis Seitoglou 和 Stylianos Basagiannis。2026年。边缘物联网的分层安全监控：一种形式化方法。载于*IEEE*网络安全与韧性国际会议（*CSR*）。arXiv:2610.09817。[29] Nikolaos Kekatos、Marinelio Chintri、Panagiotis Katsaros、Alexios Lekidis、Tom Nianios、Ioannis Seitoglou、Anastasios Temperekidis 和 Stylianos Basagiannis。2026年。边缘物联网安全的混合分层运行时验证：结合 MonPoly 与 RTLola。载于可用性、可靠性与安全性。*ARES 2026*欧盟项目研讨会工作坊（*LNCS*）。Springer，222–240。arXiv:2610.09825。doi:10.1007/978-3-032-37218-5_14[30] Nikolaos Kekatos、Marinelio Chintri、Panagiotis Katsaros、Alexios Lekidis、Tom Nianios、Ioannis Seitoglou、Anastasios Temperekidis 和 Stylianos Basagiannis。2026年。用于关键边缘物联网基础设施安全监控的弹性运行时验证基础设施。载于关键信息基础设施安全（*CRITIS 2026*）（*LNCS*）。Springer。arXiv:2610.07282。[31] Nikolaos Kekatos、Panagiotis Katsaros、Alexios Lekidis、Theodoros Nestoridis 和 Tom Nianios。2026年。面向大语言模型辅助情报、监视与侦察集群的任务级运行时保障：基于验证感知基础设施。arXiv:2607.23532。[32] Leslie Lamport。1978年。分布式系统中的时间、时钟与事件排序。*Commun. ACM* 21，7（1978年），558–565。[33] Martin Leucker 和 Christian Schallhart。2009年。运行时验证简述。*Journal of Logicand Algebraic Programming* 78，5（2009年），293–303。[34] Samuel Madden、Michael J. Franklin、Joseph M. Hellerstein 和 Wei Hong。2002年。TAG：面向自组织传感器网络的微型聚合服务。载于*USENIX OSDI*。[35] OASIS。2019年。MQTT 5.0版。*OASIS Standard*（2019年）。[36] Koushik Sen、Abhay Vardhan、Gul Agha 和 Grigore Roşu。2004年。分布式系统中安全性的高效去中心化监控。载于第*26*届软件工程国际会议（*ICSE*）。418–427。[37] Scott D. Stoller、Ezio Bartocci、Justin Seyster、Radu Grosu、Klaus Havelund、Scott A. Smolka 和 Erez Zadok。2011年。结合状态估计的运行时验证。载于运行时验证（*RV*）（*LNCS*，第*7186*卷）。193–207。[38] Synadia / CNCF。2024年。NATS 与 JetStream。https://nats.io。