CoMesh：边缘网状网络中感知-触发-执行例程的完全去中心化控制

Anna Karanika、Rui Yang、Xiaojuan Ma、Jiangran Wang、Shalni Sundram、Indranil Gupta {annak8、ry2、xm20、jw22、shalnis2、indy}@illinois.edu 伊利诺伊大学厄巴纳-香槟分校计算机科学系 伊利诺伊州厄巴纳，美国

arXiv:2303.00207v1 [cs.DC] 1 Mar 2023

摘要

尽管边缘环境中的网状网络（例如智能建筑、农场、战场等）已受到广泛关注，但对此类网络的控制层仍基本上是中心化的，并依赖云。本文聚焦于具有感知-触发-执行（STA）工作负载的应用——这类应用类似于智能家居中广受欢迎的例程抽象，但应用于更大规模的边缘物联网部署。我们提出 CoMesh，旨在应对构建本地、非云化且去中心化的感知-触发-执行应用控制方案这一挑战。CoMesh 的核心采用一种称为 k 组的抽象，以细粒度方式分摊 STA操作的负载。k 组内部的协调采用选择性的快速、低成本机制，而非昂贵的现成方案。k-组选择具有主动动态性，通过结合零消息交换机制（以降低负载）和局部敏感哈希（以感知设备的物理布局）来实现。我们分析并从理论上证明了 CoMesh 机制的安全性。通过仿真和树莓派实验室部署进行的评估表明，CoMesh 具有负载均衡、快速和容错的特性。

关键词

物联网、例程管理、容错、智能建筑

1 引言

配备物联网（IoT）设备的商业边缘部署正在迅速扩大规模和密度。目前已有超过 10 B 台物联网设备，预计到 2030 [26] 将超过 25 B 台。例如，智能建筑拥有 1.26 B 台设备，预计到 2027年，智能建筑物联网行业将增长至 2.5 B 台设备，并形成规模为 $90+ B 的市场 [38]。其他新兴的边缘物联网部署领域包括配备机器人和传感器的智能农场 [9, 18, 48, 52, 57]、战场部署 [15, 36, 50]、体育竞技场馆 [16, 56]、甚至是仓库 [14, 58] 和大型市场 [33, 35] 等室内空间，其市场也在不断增长 [2, 17, 46]。

在这类物理分布式环境中，如今常见的做法是在物联网和边缘设备之间构建网状网络。网状网络具有稳健性和弹性，只需添加或移除设备即可轻松扩展，而无需依赖可能过载并导致延迟增加的中央操作单元（或集线器）。有些设备可能是具备计算和存储能力（规模都较小）的“智能”设备，而另一些则可能是“简单”设备。如今，人们使用 Wi-Fi、Zigbee、蓝牙构建网状网络，以及 LoRa 等技术。 [22, 29, 34]采用 IEEE 802.15.4、WirelessHART [53], 或 IEC [25] 构建的工业网状网络正日益普及。在网络层方面已有大量研究，例如如何在网状网络中进行构建、配置和路由。 [4, 61]

位于网状网络之上的控制平面需要执行三类操作：感知-触发-执行（简称 STA）。控制平面需要感知：持续收集多个设备的测量值。应用会指定谓词，即涉及多个设备读数的任意布尔子句。当某个谓词满足时，控制平面需要触发预先编程的一系列操作，以控制多个设备。第三，由于其中一些操作可能持续很长时间（例如机械运动、重复操作或洒水器等基于时间的操作），控制平面需要启动并监控这些命令在多个设备上的执行，同时实现维持一致性这一颇具挑战的目标 [3]。

在较小规模下，智能家居中也存在 STA 这一抽象概念。智能家居中最常见、也适用于边缘网状网络的 STA 程序化抽象是例程。例程是由一系列命令构成的序列，可控制多个设备，其中每条命令只在一个设备上执行一项操作。任意时刻都可能有多个例程处于活动状态，另有许多例程处于休眠状态，等待触发。例程可以根据传感器值、在特定时间、根据两者的组合，或由人工触发。目前，Amazon Alexa [5], 、Google Home [20], 、Samsung SmartThings [49], 等平台均提供例程。例如，办公楼中的一个例程可能会在每个工作日晚上 9 点触发，开启安防摄像头和指定的室外照明，并锁上外门。另一个例程可能由传感器与时间的组合触发，例如，如果晚上 11 点至早上 6 点之间楼内温度降至 45 华氏度以下，就采取紧急措施以防止管道爆裂，并向设施工作人员发送通知。白天，多个例程可能持续运行，例如，一些例程根据温度调节阳光照入楼内的情况，从而升降建筑外侧的遮阳帘；另一些例程则根据占用情况改变不同房间和走廊的室内照明，等等。例程可以由人工按需触发，也可以自动触发。

在智能家居中，由于设备数量较少且家庭的物理范围有限，可以使用单个中央枢纽（例如 Alexa 设备或 Google Home 设备）在控制平面中执行全部三类 S-T-A 操作。然而，在商业物联网部署中，随着部署范围扩展到广大区域、涉及数百甚至更多设备，且例程（活跃+休眠）

本工作部分得到了 NSF CNS 1908888 和 Capital One 捐赠的支持。

数量从数十个到数百个不等，中心化集线器方案变得难以为继。事实上，在智能农场和战场等非常广阔的区域，甚至可能无法部署中心化集线器。因此，网状网络在这类场景中日益普及。

在可以部署的情况下，中央枢纽很快就可能被 S-T-A 流量压垮。对于我们所描述的例程设置，假设监控触发例程的设备时精度为 1 秒，那么在一栋楼里监控仅 500 台设备，就需要家庭集线器每 2 毫秒交换一次消息；即使采用 AWS IoT 宽松的（建议）30 秒 ping 频率， [8] 中央枢纽也需要每 40 毫秒交换一次消息。

对 STA 的分布式支持必须是本地的，无需连接云端。如今使用的中央枢纽本身并不可靠 [40], ，而且几乎总是连接到云端，而云端本身也可能发生故障 [21]。目前还没有机制能够协调多个本地集线器，以支持在大型边缘网状网络上运行例程这一特定 STA 工作负载。

本文介绍 CoMesh（控制网格），这是首个用于在边缘网状网络上监控和管理例程（例如 Amazon Alexa、Google Home、Samsung SmartThings 等平台提供的例程）的系统。CoMesh 天然是完全去中心化的，其中包含在可用智能设备之间运行的本地分布式协议（其他无法运行计算或存储数据的简单设备也可以共存）——CoMesh 只需少数设备具备智能即可运行。CoMesh 既不依赖云端，也不依赖本地中央枢纽。

CoMesh 必须应对四项重大挑战：(1) 完全去中心化的例程监控、触发和执行；(2) 在设备之间及时间维度上对工作进行负载均衡；(3) 容忍多个设备同时故障；以及 (4) 使负载感知设备的物理布局。我们提出了一种称为 k 组的构建模块，它是由智能设备动态选出的子组。这使我们能够通过为每个设备（或一组设备）的感知分配一个独立的 k 组，并为每个例程的触发和执行分配一个 k 组，将 STA 工作负载细分为精细粒度。CoMesh 的 k 组采用空间负载均衡和时间负载均衡相结合的方式：a) 每个 k 组通过一致性哈希选出；b) 定期迁移。迁移将“重”子组（例如，分配给读数变化显著的设备的 k 组，或分配给频繁触发的例程的 k 组）的负载分散到整个系统中。在每个 k 组内，我们避免使用诸如 ZooKeeper [24] 之类的基于互联网的协调系统，因为其成本可能较高；取而代之的是，我们精心设计了快速且低成本的协调技术版本，包括通过法定人数达成一致、领导者选举，以及应对子组领导者或成员的故障。具体而言，为容忍 f 故障，每个子组只需包含 $(2f + 1)$ 个成员。

为了以较低成本主动执行 k 组迁移，CoMesh 使用零消息交换协议——“零消息”意味着，沿快速路径时，任何节点都可以无需交换任何消息，计算出任意指定设备或例程当前所属的子组。在大型空间中，出于延迟和能耗方面的原因，保持通信的局部性非常重要——CoMesh 将局部敏感哈希（*LSH*） [10] 领域的思想应用于物联网和智能空间场景，使子组选择具有受监控设备的局部性，同时保留零消息交换机制的优势。

CoMesh 的底层要求包括：i）任意网状网络路由算法；以及 ii ）适用于网状网络的弱一致性成员协议（例如 Medley [60]）。

本文的贡献如下：

- 我们提出了 CoMesh，这是首个面向感知-触发-执行场景、用于边缘网状网络上的例程监控与管理的局部且完全去中心化系统。
- CoMesh 提出了一种称为 k 组的构件，用于在设备之间对感知、触发和执行工作进行负载均衡。k 组通过零消息机制和局部敏感哈希进行选择，并在 k 组内运行快速且低成本的协调与选举协议。
- 我们分析并从理论上证明了 CoMesh 机制的安全性、活性和进展。
- 我们进行了大规模仿真以及小规模的 Raspberry Pi-4 部署实验。

2 系统模型

我们假设系统采用崩溃恢复模型：物联网设备可能随时发生故障（崩溃），之后再恢复。我们不考虑拜占庭（恶意）故障，将其留待未来研究。按照容错领域文献中的惯例，我们假设系统中同时发生的崩溃故障最多为 f。 f 的值可根据部署环境计算。例如，建筑物包含多个供电域，每个供电域都连接到一个断路器。一种常见的故障模式是整个供电域断电，导致该供电域内的所有设备都发生故障。因此，一个经验法则是将 f 设置为建筑物内 最大的供电域中的智能设备数量。由于建筑物包含许多供电域， f 远小于建筑物内的设备总数 N。

我们假设一部分设备（> 3）是智能设备，配备了（少量）内存和计算能力；系统中也可能共存其他简单的（非智能）物联网设备。智能设备可以包括能够执行任意计算的各类物联网设备、树莓派、家庭中枢、手持设备等。我们假设非智能设备只能接收和处理命令，并响应状态查询，但无法运行任何计算。

由于物联网设备通常会在建筑投入使用初期就固定安装，因此我们假设智能设备 [1] 已知所有设备的位置。我们假设智能设备的时钟已同步（尽管 CoMesh 也适用于宽松同步的情况）。

智能设备通过有损无线网络进行通信。为保证一般性，我们假设智能设备之间存在无线自组织路由协议 [12, 28, 44]。我们假设任何智能设备都能向任何物联网设备发送命令。最后，CoMesh 构建于弱一致性的成员关系服务 Medley [60],之上，该服务有助于在每个智能设备上维护完整的成员列表（包含所有仍在运行的物联网设备 ID）。Medley 提供 $O(1)$ 故障和恢复事件的检测时间。表 1 总结了关键术语，图 1 展示了一个示例场景。

[1]设备移动可视为一次故障事件，随后设备会在新位置恢复。

Term	描述
命令	发送给某个物联网设备的消息，以及该设备执行的关联操作。
例程	一系列命令。由以下方式触发：时间、一组触发子句或手动触发。
物联网设备	系统中的所有物联网设备。
智能设备	具有（少量）内存和计-算能力的物联网设备。
简单设备	非智能设备的物联网设备。通常包含一个或多个传感器或执行器。

表 1： 本文中使用的关键术语。

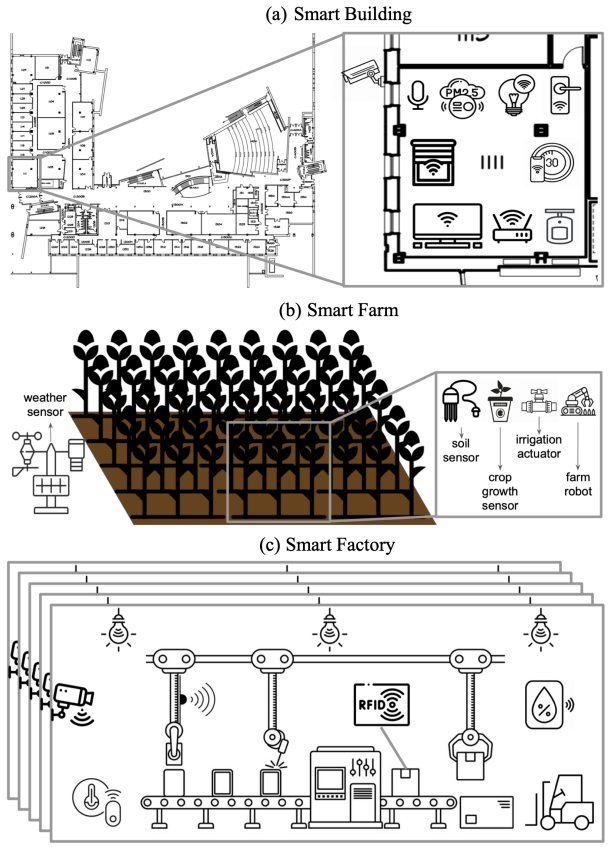


图 1： 三种感知-触发-执行部署环境：*(a)* 智能建筑，*(b)* 智能农场，*(c)* 智能工厂。

3 CoMesh 的子组：k-组机制

CoMesh 的协议依赖于子群组的构建和维护。CoMesh 使用这些子群组来执行感知-触发-执行例程和设备操作（将在后续章节中介绍）。本节将介绍子群组构建。

CoMesh 的子组称为 k 组，体现了我们的目标：将 k 组的规模维持在大约（一个较小的值） k。 k 是一个全局配置的变量。为容忍 f 全系统故障，我们设置 $k = (2f + 1)$。我们介绍 k组选择（第 3.1 节）、选举（第 3.2 节）、法定人数（第 3.3 节）、时间迁移（第 3.4 节）、故障响应（第 3.5 节）和局部性（第 3.6 节）。

3.1 k-组成员选择

CoMesh 的 k-组需要以如下方式进行选择：(i) 实现负载均衡；(ii) 使用最少的消息数量。

通过零消息交换进行负载均衡选择：具有标识符 S 的智能设备通过计算一致性哈希来发现自己是否属于目标设备 D 的 k-组： $Hash(S,\ D)$。随后，整个系统中哈希值最低的 k 个标识符将被批准为 k-组成员。该设计的灵感来自密码学抽签 [6, 39]。请注意，在给定固定哈希函数的情况下（例如 SHA-2 或 MD-5 等密码学哈希），任意智能设备都可以通过对其（完整）成员列表中的每个标识符计算该哈希函数，来计算任何 k-组的全部成员关系。只要成员列表正确，这一过程无需交换任何消息。

3.2 领导者选举

k-组的目标是以容错方式复制其受监控实体的状态。为了提高操作效率，每个 k-组选举一名领导者。具有最低哈希值的存活智能设备（$Hash(S,\ D)$）将被视为领导者。虽然选举可以通过零交换机制完成，但为了确保领导者确实存活，CoMesh 运行众所周知的 Bully 选举算法，以快速选出领导者 [19]。Bully 算法颇具吸引力，因为它在常见情况下速度很快：最低哈希值节点（领导者）只需 1 个往返时间即可通知 k-组中的其他成员。如果发生故障，Bully 算法最坏情况下需要 5 个往返时间。

3.3 仲裁一致性

k组中的快速一致性通过法定人数实现。在大小为 $k = 2f + 1$ 的 k组中，组内任意成员（通常是领导者）向所有 M 成员发送消息，并等待至少$(f + 1)$名成员的确认消息。因此，即使发生 f次故障，仍至少有一名存活的k组成员知晓 k组中的最新决策，并可将其传达给未来的领导者。

3.4 基于纪元的 k组迁移

虽然第3节了空间 负载均衡，为实现时间负载均衡，CoMesh会定期迁移每个 k组的成员关系。迁移每隔一个纪元进行一次，且整个系统的纪元长度固定。（不同的 k组可以在不同时间迁移。）

首先，我们扩展用于选择的哈希函数，将纪元编号纳入其中，使其变为 $Hash(epoch,\ S,\ D)$。每次发生纪元变更时，$epoch$ 都会递增 1。接下来，纪元到期时，按以下顺序执行：1) 使用 哈希（++纪元，S，D）组建新的 k组；2) 在新的 k组中进行选举（参见第 3.2 节）；3) 将旧 k组领导者维护的任何状态迁移到新 k组的领导者；4) 新k组的领导者将状态信息复制到新 k组成员中的法定人数处（参见第 3.3 节）；5) 原子切换到新k组：新 k组的领导者告知旧 k组的领导者，旧

k组已停用，旧领导者予以确认；然后 6) 旧领导者告知旧 k组成员，他们已被停用。

当新的 k组领导者在步骤 5 中向旧 k组领导者确认其已接收到全部状态后，旧 k组领导者可以删除其旧的 k组状态。在步骤 6 之后，旧 k组节点可以删除其针对该 k组的旧状态。在步骤 1 到 7 期间， k组中不会执行任何监控操作。这意味着，旧 k组成员和旧领导者会恰好在纪元切换边界停止所有 k组操作。在实践中，纪元变更足够迅速，因此这段监控间隔（步骤 1 到 6 之间）很短，正如我们的实验结果所示。步骤 6 之后，所有正常的 k组操作都会恢复。

3.5 故障响应： k组更新

如果智能节点 N_i 发生故障，故障检测层会将其故障通知所有其他节点。由于 CoMesh 的操作对于组大小为 $\in [f+1, 2f+1]$时仍能保持正确，我们仅在 f个节点发生故障时触发一次 k组选举（而不是每次发生故障都触发），从而降低故障响应的开销。

唯一的例外是，发生故障的节点 N_i 是某个k-组的领导者。这会导致重新运行第 3.2 节中的选举算法。该新领导者需要从 k-组中的所有节点重构状态。每一份状态都需要由法定人数（$f+1$ 个节点）确认。该状态是正确的，因为所有过去的更新至少存在于 $f+1$ 个节点上，因此新领导者能够从至少一个非故障节点接收到每一项过去的更新。如果新领导者在重构期间发生故障，则会重新进行选举，并重复这一过程。

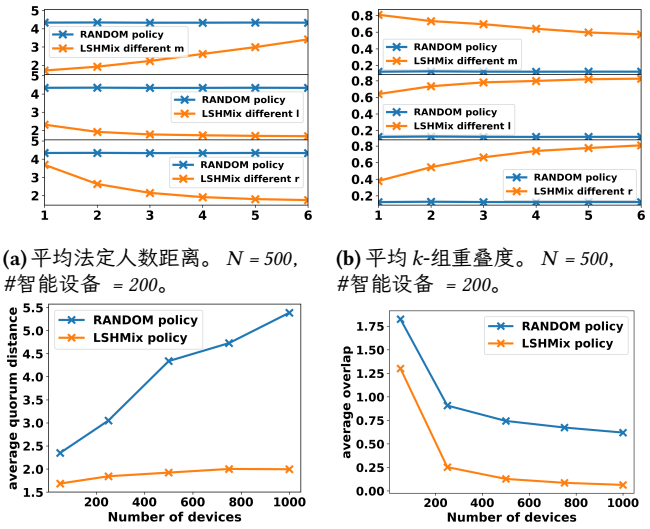
3.6 通过局部敏感哈希进行分组

由于底层路由基础设施是自组网，这会带来两个问题。首先，庞大的设备和例程数量可能导致大量的 k-组。其次，随机选取的k-组将由分散在自组网中的节点组成，从而导致链路上的消息开销很高。

限制 k-组的数量：我们使用聚类算法 [31] 将设备划分为多个簇。每个簇会被分配到一个 k-组，并从该簇中随机选择一个设备作为“中心”（即代表），以计算上文所述的局部敏感哈希函数。对于例程，我们随机选择一个与其交互的设备作为其代表，该设备所属的簇即成为该例程的簇（因而也是其 k-组）。

局部敏感哈希： 随机选择 k-组（第 3.1 节）会使 k-组成员分散在整个网络中，从而降低组内操作（选举、法定人数、领导者间通信）的速度。挑战在于如何让 k-组的选择具有局部性，同时保留哈希所带来的诸多空间负载均衡优势。为此，我们需要借鉴局部敏感哈希（LSH）的思想 [10],——这是高维数据科学中的一种方法。据我们所知，本文是首篇将 LSH 应用于边缘/物联网网络的论文。

具体而言，我们将 k-组成员选择修改为一个两阶段流程：选择受监控设备/簇附近的 $f+1$ 个节点，再从整个组中随机选择另外 f个节点。这样既能通过附近的 $f+1$ 个成员加快组内操作，同时又能确保电源域故障（见第 2 节）不会影响一个组的运行（因为一个电源域中的设备数将少于 $f+1$ 个）。



(a) 平均法定人数距离。 $N = 500$, #智能设备 $= 200$。 **(b)** 平均 k-组重叠度。 $N = 500$, #智能设备 $= 200$。

(c) 不同系统规模 (N) 下的平均法定人数距离。**(d)** 不同系统规模 (N) 下的平均 k-组重叠度。

图 2： 不同选择策略、参数和系统规模下的平均法定人数距离与 k-组重叠情况。

LSH [10] 的主要思想是，以较高概率将（空间中）相似的向量哈希到同一“桶”中。系统从受监控设备组中随机选择一个“中心”，并将智能设备哈希到各个桶中，然后从与该中心处于同一“桶”中的智能设备中选择大多数“候选项”。如果候选项多于 $f+1$ 个，我们就选择距离中心最近的 $f+1$ 个候选项。请注意，从候选项中随机选择大多数候选项通常效果不佳，因为桶中可能还包含一些距离中心较远的节点。如果候选项少于 $f+1$ 个，CoMesh 会先检查中心的邻居（距离中心较近的设备）是否有可以“借用”的候选项，然后再进行随机选择。

在连续时期中， k-组成员的重叠有助于纪元变更期间的数据传输操作。为此，我们将 $(pre_i, \ldots, pre_d)$ 附加到其自身的位置向量 $(x_1, \ldots, x_d)$ 中，其中 pre_i 是该智能设备在前一个纪元中监控的所有设备在第 i 个维度上的平均位置。然后，我们将此向量作为 LSH 的输入。为了实现负载均衡，也可以在向量条目中加入一些随机性（这是我们的默认设置）。我们将 $(x_1, \ldots, x_d)$ 附加到简单设备的位置向量中，使所有设备的维度相同。

由 [10], 改进得到的局部敏感哈希函数 $h_{a,\ b} = \lfloor \frac{a \cdot v + b}{r} \rfloor$ 将一个 d 维向量 v 映射到一组整数上，其中 a 是一个随机的 d 维向量，其各项独立地从 p 稳定分布（在我们的例子中为高斯分布） [10, 62] 中选取； b 是从 $[0, r]$ 中均匀选取的实数， r 是一个参数。为了进一步降低差异很大的向量落入同一桶中的概率，可以将多个独立的哈希函数串联起来，使得 $H = (h_1, \ldots\ldots, h_m))$。此外，我们可以使用 l 个哈希（H 个）来增加落入同一桶中的相似向量数量。也就是说，我们有 $\mathrm{H} = \{H_1, \ldots\ldots, H_l\} = \{(h_{11}, \ldots\ldots, h_{1m}) \ldots\ldots (h_{l1}, \ldots\ldots, h_{lm})\}$。

N	50	250	500	750	1000
#候选项	13.0223	34.1258	50.7501	52.8035	66.9135

表 2： 不同 N 下 k 组的平均候选成员数。*40%* 的设备为智能设备。

我们对如何选择这些 LSH 参数的取值作一些直观说明。较大的 m 值会排除不相似的节点，并减少候选项数量；较大的 l 和 r 则可能增加候选项数量（图 2a 和图 2b ）。本文采用的默认设置为 $m = 2$，$l = 2$，$r = 4$。在图 2c 和图 2d 中，我们针对不同规模的网格拓扑测试了默认设置。可以看到，对于随机选择策略，要达到法定人数所需的平均跳数（法定人数距离）会随系统规模线性增长；而对于基于 LSH 的选择策略，该值始终较小。关于随机网络和簇集网络的更多实验结果见第 6 节。

尽管基于 LSH 的 k组重叠度在较大的 N 下会下降（图 2d），但仍远高于随机策略。我们推测，随着节点增多、候选项增加，同一设备在连续纪元中被选中的概率会降低。表 2 验证了这一假设。也可以调整 LSH 参数，在不影响法定人数距离的情况下降低候选项数量。例如，在 $N = 1000, m = 3,$ $l = 2,\ r = 4$ 时，候选项的平均数量为 21，平均法定人数距离为 2.4（随机策略为 5.3）。

4 例程与设备管理

现在我们将介绍，如何基于第 3 节的 k组，实现例程和设备的感知-触发-执行管理。

4.1 设备管理

在 CoMesh 中，每个设备都会被分配到唯一的 k-组。该 k-组负责对设备执行感知操作，即跟踪设备的在线/离线状态以及设备的最新状态（例如恒温器的温度读数）。设备所属 k-组的领导者（以下简称 设备领导者）会定期向设备发送 ping。如果设备在 2 个往返时间内没有响应，则视为设备发生故障，设备领导者会通知拓扑中的所有智能设备。如果设备的检测到的状态 D（在确认消息中收到）与设备领导者处记录的最新状态不同，设备领导者就会将该新状态发送给所有触发子句中包含 D 的例程（具体来说，发送给相应例程所属 k-组的领导者）。

4.2 例程管理

回想一下，例程由一个触发子句（涉及多个设备及其状态的任意布尔子句）和一组在例程被触发时执行的命令组成。前者的触发集合（读取自）与后者的命令集合（作用于）中的设备可能不同，也可能存在重叠。

在 CoMesh 中，每个例程都会分配一个唯一的 k-组，负责例程的感知（监控）、触发和执行。感知由设备 k-组协同完成，如前文所述。例程可能处于以下某种状态（由其 k-组维护）：(i) 未_触发，(ii) 正在获取_锁，(iii) 执行中，以及 (iv) 正在释放_锁。

我们将例程 k-组的领导节点称为例程领导者。例程领导者会与例程触发集合中每个设备的设备领导者建立连接。每当这些设备领导者向例程领导者发送更新后的状态时（第 4.1 节），后者就会检查该状态变化是否满足例程的触发子句。

触发例程时，例程领导者会将其状态从 NOT_TRIGGERED 更改为 ACQUIRING_LOCKS，并将此状态复制给其所有 k 组成员。它会等待 $(f + 1)$ 个响应（包括自身）。例程启动时，会先获取例程命令集合中所有设备的（虚拟）锁，然后执行其命令。最后，例程结束时会释放所有锁。事实证明，这种悲观并发控制方法速度快、能够避免死锁和活锁，并且具有良好的可扩展性 [3]。

给定设备的锁仅由该设备组维护（而非例程组维护），这使多个例程能够竞争锁。要获取设备 D 的锁，例程 R 的例程领导者会与 D 的设备领导者通信。（如果例程领导者不知道当前的设备领导者，它会联系该设备的所有 k 组成员，通过第 3.1 节的零消息交换协议获知领导者，然后再联系设备领导者。）

设备领导者维护一个等待列表，其中记录请求锁的例程，并且一次只授予一个锁（按先进先出顺序）。当例程领导者释放锁时（处于 RELEASING_LOCKS 阶段），设备领导者会从等待列表中取出下一项，向其所有设备 k-组成员发送请求，并等待（$f + 1$）个确认，然后再将锁返回给该请求例程领导者。当所有锁都已获取后，例程领导者会将该例程的状态更改为执行中，将此信息传达给所有例程 k-组成员，等待（$f + 1$）个确认，然后开始为该例程发出命令。例程的命令始终由例程领导者路由至相应的设备领导者，再由设备领导者转发给设备。这样，设备领导者（k-组）始终了解设备的最新状态。

锁的获取顺序会影响触发速度。有两种设计方案——顺序式和乐观式。

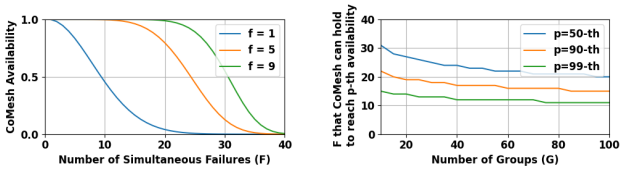
4.2.1 顺序锁获取（*SLA*）策略。例程领导者按顺序获取锁。如果按设备ID递增的顺序获取锁，就能避免多个具有冲突设备集的例程之间发生死锁（不会形成环）。当冲突很少或没有冲突时，SLA策略可能较慢。

4.2.2 乐观锁获取（*OLA*）策略。CoMesh 的乐观锁获取（OLA）策略会尝试同时获取其（所需）设备的所有锁。如果任何锁请求失败，例程领导者就会释放已获取的所有锁，然后重新尝试获取全部锁。为避免消息内爆，重试会在退避超时后进行。当例程的命令集存在重叠时，OLA 的代价较高。

我们将在后文通过实验比较这两种策略。

5 形式化分析

我们对 CoMesh 的性质进行了形式化分析。为简洁起见，我们先直观地概述研究结果，然后给出形式化定理。证明见附录A。



(a) 不同同时故障数量下的可用性。**(b)** 系统中组数不同的影响

图 3： *CoMesh* 在不同参数下正确运行的概率。（默认 S=100、G=30、f=5）

- 继承：当一个 k 组发生变更时，旧领导者和新领导者持有的状态相同。
- 安全性：任何两个涉及重叠设备集合的例程都不得同时执行。
- 活性：不会有例程发生死锁。
- 进展：每个例程都会取得进展。
- 我们还计算了系统在超过 f 台设备发生故障时的（概率）可用性。

定义 1. 继承是指，给定一个 k-组，在纪元变更前后或领导者故障后， k-组的新旧版本保持相同的状态。

现在我们陈述关键结果。

引理 1. **[继承]**当至多发生一个节点故障时，状态传输阶段结束后即可保证继承。也就是说，在至多发生一个节点故障的情况下，状态传输阶段结束后，旧 k-组领导者持有的状态与新 k-组领导者持有的状态相同。

定理 1. **[安全性]**当至多发生一个节点故障时，任何两个在设备上存在冲突的例程（即它们触及的设备集合不互斥）都不能同时执行。对于 *SLA* 和 *OLA* 锁定策略（第 *4.2.1* 节），这一结论均成立。

定理 2. **[无死锁]** 不会有两个例程陷入死锁。

定理 3. **[进展性/活性]** 考虑一组正在执行的例程 $\mathcal{R}$，其中每个例程仍在等待获取一个或多个锁，以及它们正在等待的设备 D 集合（即并集）。如果没有新的例程加入$\mathcal{R}$，并且使用 *SLA* 锁定，那么：集合 $\mathcal{R}$ 中至少有一个例程会取得进展（即获得对其所需的另一个设备的访问权限）。

定理 4. **[可用性]** 在发生 $F(>f)$ 次同时故障时，系统仍保持可用，即所有 k 组至少包含$(f+1)$个无故障节点的概率为 $P(F)$，其变化如下：

(1) 当 $F \le f$、 P（F） $= 100\%$(2)

当 $F \ge f$ 时，可用性为：

$$
\begin{aligned}
P(F) &= P(\textit{at most } f \textit{ failures across } k\textit{-groups}) \\
&= P(\textit{at most } f \textit{ failures in one } k\textit{-group})^G \\
&= \left(\sum_{i=\max(0,k+F-S)}^{f} \frac{\binom{S-F}{k-i} \times \binom{F}{i}}{\binom{S}{k}} \right)^G
\end{aligned}
$$

参数	符号表示	默认值
# 设备总数	N	250
# 完整性所需故障数	f	2
k 组大小	K	5
# 智能设备	S	100（所有设备的 40%）
#种子	–	10
纪元长度	–	200
最大例程长度	–	5
每个例程的平均设备数	–	5
# 由 k-组管理的设备	–	25
带宽上限	–	625 KBps/5Mbps
k-组选择策略	–	LSH 随机混合
设备集群策略	–	局部性
领导者选举策略	–	LSH 最小哈希
默认 LSH 参数	–	m = 2, l = 2, r = 4

表 3： 默认参数值

实际上，这种扩展方式是可行的。图 3a 显示，随着 F 上升，CoMesh 的可用性如预期般下降，但当 F 增长至 $9\times f$ 时，仍保持在 50% 以上。图 3b 显示，为了达到较高的可用性（p = 50%，90%，99%），所需的 F 值下降得非常缓慢——这表明其可扩展性与组的数量相关。

6 仿真评估

为了在 CoMesh 上对包含数百个节点的大规模实验进行测试，并测量其内部行为，我们编写了一个定制模拟器。NS3 等高保真模拟器在节点数超过几百个后存在已知的可扩展性问题。我们同时测量两类指标：i) 面向用户的指标，例如启动例程的延迟、负载均衡等；以及 ii) k-组操作的微基准测试。

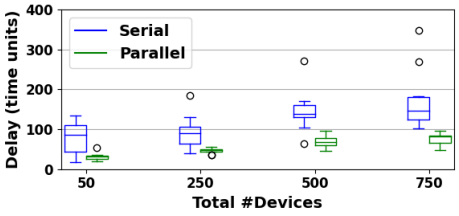
我们的模拟器通过根据密度缩减网络带宽来模拟网络争用：具体而言，从节点 A 向节点 B 发送数据的带宽被设置为默认节点带宽除以 A 的一跳邻居数量。我们根据在线论坛上用户对物联网设备 [1, 37] 的投诉中报告的带宽数据，保守地设置默认节点带宽（表 3）。由于我们假设传输半径较小，每一跳的延迟固定为 1 个时间单位。自组织路由基于 Dijkstra 最短路径。

我们使用三种不同的网络拓扑：三维网格、随机拓扑和簇状拓扑。我们确保图是连通的。CoMesh 配置了拓扑感知 LSHMix（第 3.6 节）。表 3 显示了默认参数。

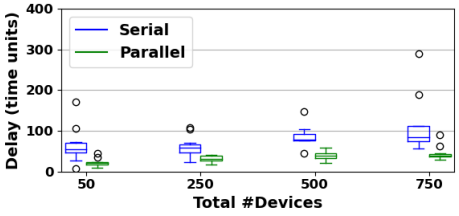
6.1 客户端延迟

当某个例程的触发条件变为真时，在没有其他例程处于活动状态的情况下，该例程需要多久才能执行第一条命令？这就是例程的客户端延迟：其 k组检测（并复制）触发条件、锁定所有涉及的设备并开始执行所需的时间。在图 4 中，我们观察到，无论拓扑或锁定策略如何（SLA = 串行或 OLA = 并行），CoMesh 的客户端延迟都会随设备数量增加而增长。所有图中的趋势都是逐渐上升的。

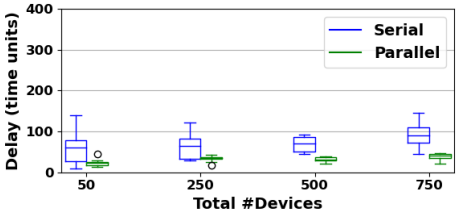
正如预期，并行锁定（OLA）策略比串行锁定（SLA）更快，因为客户端延迟不包括相互冲突的例程。



(a)网格 拓扑



(b)随机 拓扑



(c)簇状 拓扑

图4： 三种不同拓扑设置下串行与并行锁定策略的客户端延迟（秒）

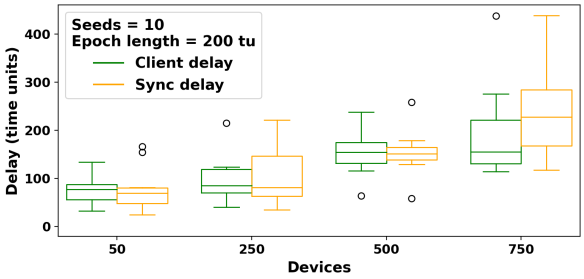
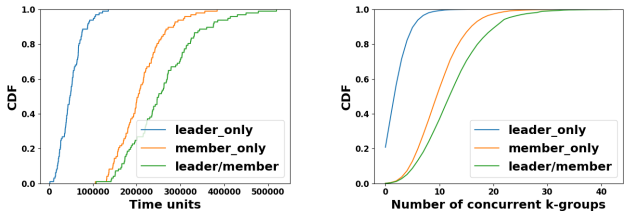


图5： 网格拓扑中 *40%* 的智能设备发生节点变动（加入或故障）时的客户端延迟和同步延迟

在这 3 种拓扑结构中，网格的客户端延迟高于随机和簇状拓扑（两者相近），因为网格拓扑中的设备分布得更分散。也就是说，尽管这三种拓扑结构的整体密度相同，但网络不同区域的密度方差在随机和簇状拓扑中更大，因此会形成节点“簇”，从而加快通信速度。

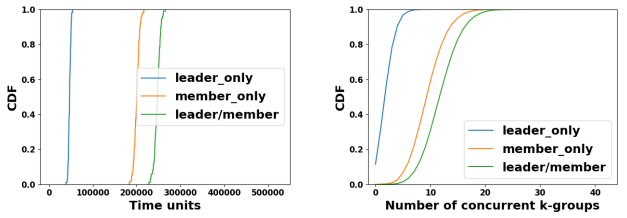
6.2 节点变动对客户端/同步延迟的影响

图 5 显示，在网格拓扑（SLA 锁定）中，当 40% 的智能设备节点发生变动时，客户端延迟仅增加 25%。这部分额外延迟源于 k 组的旧领导者故障后进行领导者选举并向新领导者转移状态。我们将例程释放其最后一个



（a）时间 负载分布 具有局部性 *k*-组

（b）空间 负载分布，具有局部性 *k*-组



（c）时间 负载分布 具有随机 *k*-组

(d)空间负载分布，采用随机 k 组

图 6： *CoMesh* 在两种成员和领导者选择算法（局部性与随机）下随时间和空间变化的负载分布

冲突锁与下一个等待执行的（被阻塞的）例程执行其第一条命令之间的时间定义为同步延迟。本实验中，我们连续触发了同一例程的两个实例（即所有设备均为冲突设备）。我们观察到，同步延迟也会随着系统规模缓慢增加——将节点数从 50 增至 750 会使同步延迟（以及客户端延迟）增加 $< 4\times$ 。

人们可能会认为客户端延迟高于同步延迟。然而，两者各自都排除了一些对方包含的操作。同步延迟不包括例程触发及相关 k 组法定人数所需的时间。客户端延迟不包括锁释放延迟及相关法定人数所需的时间。

6.3 负载分布

图 6 展示了网格拓扑中领导者节点、成员节点以及所有（领导者或成员）节点的负载分布累积分布函数。时间负载 是指在整个实验期间，智能设备担任某一特定角色的时长。空间负载 是指智能设备同时参与的 k 组数量。

上方的两幅图展示了采用 LSH 的 CoMesh，下方的两幅图展示了采用随机成员选择的 CoMesh。正如预期，随机成员选择在节点之间以及时间维度上的负载均衡效果最佳。尽管 LSH 会将职责聚类到更靠近目标设备的位置，从而导致负载不平衡（上方两幅图），但成员以及领导者/成员的时间负载中位数（左侧两幅图）在随机选择与 LSH 之间却出人意料地相近，这表明 LSH 在成员负载均衡方面几乎与随机选择同样出色。由于聚类效应，LSH 的成员负载尾部更高。同样的聚类效应会导致领导者候选者重复出现，并提高 LSH 中的领导者负载中位数（左上图）。

6.4 k 组操作

现在，我们测量 k-组内的性能。我们使用表 3 中的默认参数。图 7 显示，所有 k- 组

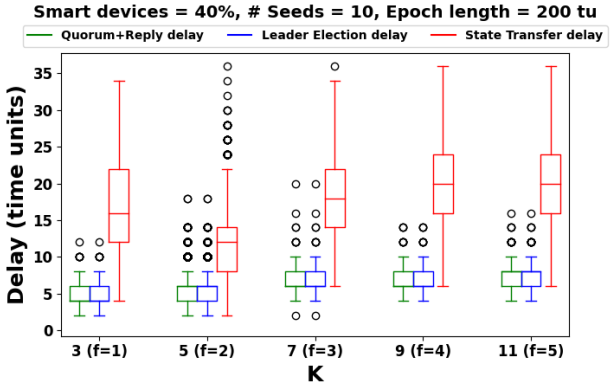


图 7： 不同 k取值下， k组内操作的延迟

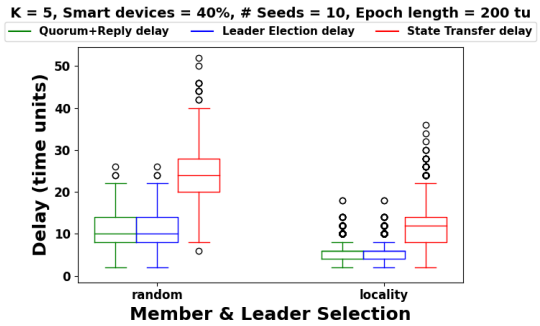


图 8： 实体聚类、成员选择和领导者选举采用不同策略时， k组内操作的延迟

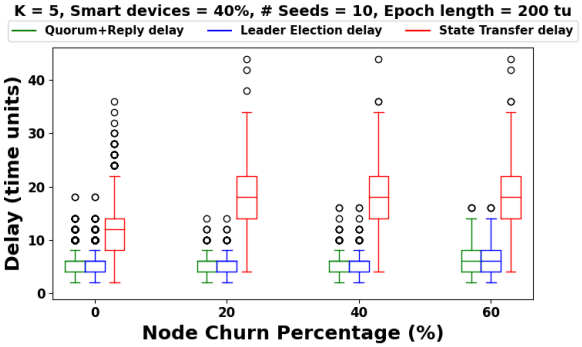


图 9： 节点 *churn* 比例不同时， k组内操作的延迟

操作——选举、法定人数和状态转移——随着 k-组规模从 3（$f = 1$）增至 11（$f = 5$）而具有良好的可扩展性。法定人数具有可扩展性，因为请求和回复是并行发送的。在常见情况下，领导者选举延迟具有可扩展性，这是因为 Bully 算法在最佳情况下的表现。状态转移延迟取决于状态量，但不取决于 k。

图 8 证实了我们的假设：用于 k-组实体选择、成员选择和领导者选举的位置感知机制



(a) 网格拓扑

(b) 线形拓扑

图 10： 树莓派拓扑：实验室部署。

可以降低 k-组的运行延迟。状态转移操作的延迟下降幅度不如法定人数或领导者选举的延迟那么大，因为除法定人数外，向旧领导者发送的单播不受 LSH 的影响。

图 9 显示，拓扑中的智能设备发生节点变动，不会影响 k 组的运行。只有当节点变动影响约 60% 的智能设备时， k 组的运行才会变慢。这是因为 k 组的运行只需要其多数成员参与。当 f 名成员在一个 k 组中发生故障时，领导者会确保替换故障成员，因此未来最多有 f 台智能设备发生故障也不会影响该组。

7 部署评估

我们在树莓派（RPi）4 环境中实现了 CoMesh。我们的实现包含 12.5K 行 Java 代码。部署使用的是 RPi 4 Model B，配备 2GB LPDDR4 内存和一颗 1.5GHz Cortex-A72 核心的 Broadcom BCM2711 四核 CPU。第 6 节展示了大规模仿真结果，而我们使用树莓派的目标是对 CoMesh 核心机制（即 k 组）在少量真实设备上的运行情况进行基准测试。

我们在实验室部署了 11 台 RPi4 设备，构建了不同的网状拓扑。为减弱 RPi 的强信号并构建网状拓扑，我们始终用铝箔包裹每台设备，将传输功率降至最低设置 15dBm [3]。虽然 CoMesh 适用于任何自组织路由协议，但为具体说明，我们使用 OLSR 路由 [43]，因为它已在 Pi 4 上标准化。

我们将 CoMesh 配置为使用 Medley [60] 成员协议的开源实现。CoMesh 配置为使用 k-组和 $k = 3$，并每秒监控一次设备。为避免冷启动偏差，我们在 CoMesh 初始化 10 秒后开始测量。每个数据点至少包含 5 次使用不同种子进行的试验运行。纪元每 60 秒变化一次。不同例程可能同时触发。同一例程可能多次触发，但两次触发之间的间隔不会短于其上一次调用后的 10 秒。我们使用 SLA 锁定（第 4.2.1 节）。

我们实现了 图中所示的两种 2D 网格的拓扑，以及（ii）一种 L 形线形 （用于模拟我们办公楼某一层的轮廓）。每种拓扑覆盖的面积均为 4 米 × 4 米。线形拓扑的直径大于网格拓扑。

带宽使用量——去中心化与中心化： 图 11 展示了去中心化 CoMesh 的每节点带宽，包括端到端（E2E）消息以及临时路由的（逐跳/H2H）消息。为进行比较，我们还实例化了具有 $k = 1$ 和一个 k组的 CoMesh，即由单个中心化设备负责所有例程的

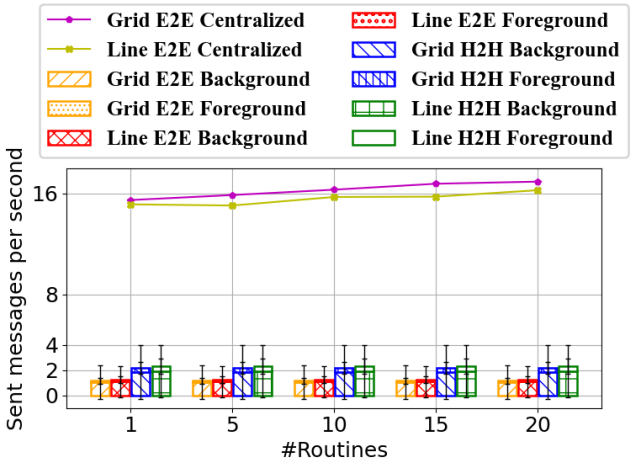


图 11： 带宽：去中心化 *CoMesh* 与中心化 *CoMesh*。

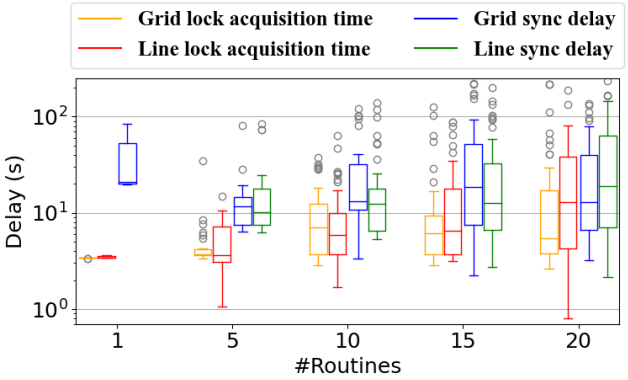


图 12： 锁获取时间和同步延迟。

监控和管理——这类似于使用中央家庭中枢（第 1 节）。我们观察到：a）与集中式方法相比，去中心化 CoMesh 将带宽降低了一个数量级（超过 10 ×）。b）端到端带宽和逐跳带宽均具有可扩展性，即随着例程数量的增加仍保持平稳。c）前台操作（例如选举、k 组迁移等）的带宽与不可避免的后台带宽（例如设备监控）相比仍然很小。

启动例程的等待时间： 图 12 展示了任意给定例程的两项指标。第一项是锁获取时间，即设备锁变为可用后获取它所需的平均时间，范围为 3.41 秒至 13.02 秒。随着例程数量增加，网格的中位延迟保持稳定，而线形的中位延迟略有上升。第二项是同步延迟，即例程的最后一个设备变为可用到例程启动之间的时间；随着例程数量增加，该指标保持稳定，中位数介于 10.23 秒至 20.833 秒之间。

组间*k*操作延迟： 表 4 显示，CoMesh 的常见操作速度很快：法定人数操作的中位耗时为 1.08 秒，选举为 1.17 秒，纪元变更期间的状态转移为 2.07 秒。

8 相关工作

一致性哈希与委员会： 点对点 (p2p) 网络系统，例如 Chord [54], Dynamo [11], Cassandra [32]

*k*组延迟	中位数	mean	P90
法定人数回复	1.0805	1.4021	1.4962
领导者选举	1.1675	1.4913	1.9249
状态转移	2.0750	2.8468	3.1390

表 4： *CoMesh* 的 *k*组延迟（秒）。

采用一致性哈希 [30] 来查找存储键值对的节点。CoMesh 的 *k*-组机制受到 [23], 中选举协议所用一致性子组的启发，但该机制既不具备动态性，也不是为物联网环境设计的，也不具备局部性。

共识： 针对无线环境，人们已经提出了多种一致协议，例如 Wireless Paxos [45], 和无线传感器网络拜占庭共识 [59]。不同于这些需要多轮消息交换的共识算法，CoMesh 的 *k*组高效且快速，在常见情况下只需一个通信轮次即可达到法定人数，因此能够快速运行。

物联网应用管理： 物联网例程和应用管理系统已得到广泛研究，但它们采用中心化控制平面，例如 HomeOS [13], Beam [51], DepSys [41], eHome [47], 和 SafeHome [3]。Rivulet [7] 采用分布式架构，并利用智能设备交换消息、将状态备份到影子节点。然而，与 CoMesh 不同，Rivulet 侧重于容错，而未涉及例程管理。其他有关物联网容错的研究 [27, 42, 55],也没有管理例程执行。据我们所知，CoMesh 是首个完全去中心化的物联网例程管理系统。

9 总结

我们介绍了 CoMesh 系统，它以完全去中心化的方式管理智能空间（建筑、家庭或校园）中的例程，无需依赖任何中心组件。对 CoMesh 技术的分析表明，该系统实现了连续性（继承）、安全性、活性和进展。我们在各种拓扑结构下对 CoMesh 进行实现和仿真，结果表明，随着设备数量增加至数百台，以及 f（故障数量）的值增大，系统均能良好扩展。CoMesh 还能在智能设备之间以及不同时段有效均衡负载。CoMesh 为实现当今智能空间系统的自管理版本开辟了道路。未来的研究方向包括构建集中式—分布式混合管理系统、考虑智能设备的异构性以及能源考量。

参考文献

[1] 2019. 新款树莓派 4 2GB 上传速度慢。https://forums.raspberrypi.com/viewtopic.php?t=244852 [2] G. Aarti 和 K. Vineet。2022。航空航天与国防市场中的物联网：按组成部分（硬件、软件、服务）、部署模式（本地部署、云）、连接技术（蜂窝网络、Wi-Fi、卫星通信、射频）和应用（车队管理、库存管理、设备维护、安全、其他）划分：全球机遇分析与行业预测，2020—2030 年。https://www.alliedmarketresearch.com/internet-of-things-in-aerospace-and-defense-market。[3] S. B. Ahsan、R. Yang、S. A. Noghabi 和 I. Gupta。2021。Home, SafeHome：通过可视性和原子性提升智能家居可靠性。发表于*EuroSys*。[4] I. F. Akyildiz 和 X. Wang。2005。无线网状网络综述。*IEEE Commun. Mag.* 43, 9 (2005)。[5] Amazon Alexa 2014。Amazon Alexa。https://developer.amazon.com/alexa。[6] G. I. Apecechea。2019。区块链中的密码学抽签：可验证随机函数的重要性。https://medium.com/witnet/cryptographic-sortition-in-blockchains-the-importance-of-vrfs-ad5c20a4e018

[7] M. Ardekani、R. Singh、N. Agrawal、D. Terry 和 R. Suminto。2017。Rivulet：面向智能家居应用的容错平台。发表于 Middleware。[8] AWS。2022。AWS IoT Core 额外计费详情。https://aws.amazon.com/iot-core/pricing/additional-details/ [9] R. Chandra 和 S. Collis。2021。面向小规模生产者的数字农业：挑战与机遇。CACM 64, 12 (2021)。[10] M. Datar、N. Immorlica、P. Indyk 和 V. S. Mirrokni。2004。基于 p-稳定分布的局部敏感哈希方案。发表于 SoCG。[11] G. DeCandia、D. Hastorun、M. Jampani、G. Kakulapati、A. Lakshman、A. Pilchin、S. Sivasubramanian、P. Vosshall 和 W. Vogels。2007。Dynamo：亚马逊高可用键值存储。发表于 SOSP。[12] E. W. 等人 Dijkstra。1959。关于图相关两个问题的札记。Numerische mathematik 1, 1 (1959)。[13] C. Dixon、R. Mahajan、S. Agarwal、A. J. Brush、B. Lee、S. Saroiu 和 P. Bahl。2012。面向家庭的操作系统。发表于 NSDI。[14] Z. Fatima、M. H. Tanveer、S. Zardari、L. F. Naz、H. Khadim、N. Ahmed 和 M. Tahir。2022。利用物联网和工业5.0实现生产工厂与仓库自动化。Applied Sciences 12, 4 (2022)。[15] Y. Feng、M. Li、C. Zeng 和 H. Liu。2020。战场物联网（IoBT）的鲁棒性：有向网络视角。Entropy 22, 10 (2020)。[16] M. A. Fox、J. L. Breese 和 G. Vaidyanathan。2019。现场音乐演出与物联网。[17] Market Research Future。2022。智能体育场市场预计将达到243亿美元，2021—2030年复合年增长率为23%。https://www.globenewswire.com/news-release/2022/06/01/2453900/0/en/Smart-Stadium-Market-Estimated-to-Hit-24-3-Billion-with-a-CAGR-of-23-during-2021-2030-Report-by-Market-Research-Future-MRFR.html。[18] H. Gan 和 W.S. Lee。2018。智能农场导航系统的开发。IFAC-PapersOnLine 51, 17 (2018)。[19] H. Garcia-Molina。1982。分布式计算系统中的选举。IEEE TOC C-31, 1 (1982)。[20] Google Home 2016。Google Home。https://store.google.com/us/product/ google_home。[21] M. Grothaus。2019。Google 大规模服务中断导致部分 Nest 用户无法开门或使用空调。https://www.fastcompany.com/90358396/that-major-google-outage-meant-some-nest-users-couldnt-unlock-doors-or-use-the-ac [22] X. Guo、Lo. Shangguan、Y. He、N. Jing、J. Zhang、H. Jiang 和 Y. Liu。2022。Saiyan：面向 LoRa 反向散射系统的低功耗解调器设计与实现。发表于 NSDI。[23] I. Gupta、R. Van Renesse 和 K. P Birman。2000。适用于大型组的概率正确领导者选举协议。发表于 DISC。[24] P. Hunt、M. Konar、F. P. Junqueira 和 B. Reed。2010。ZooKeeper：互联网规模系统的无等待协调。。发表于 ATC。[25] ISO中央秘书处。2016。系统与软件工程——极小型实体（VSE）的生命周期配置文件——第1部分：概述。标准 ISO/IEC TR 29110- 1:2016。国际标准化组织，瑞士日内瓦。https: //www.iso.org/standard/62711.html [26] Bojan J. 2021。2021年物联网统计数据——拆解万物。https://dataprot.net/statistics/iot-statistics/。[27] A. Javed、K. Heljanko、A. Buda 和 K. Främling。2018。CEFIoT：适用于边缘与云的容错物联网架构。发表于 WF-IoT。[28] D. B. Johnson 和 D. A. Maltz。1996。自组无线网络中的动态源路由。发表于 Mobile computing。[29] Mohamad K.、Anthony W. 和 Vamsi T. 2021。简化反向散射部署：全双工 LoRa 反向散射。发表于 NSDI。[30] David Karger、Eric Lehman、Tom Leighton、Rina Panigrahy、Matthew Levine 和 Daniel Lewin。1997。一致性哈希与随机树：用于缓解万维网热点的分布式缓存协议。发表于 STOC。654–663。[31] K. Krishna 和 M. Narasimha Murty。1999。遗传K均值算法。IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 29, 3 (1999), 433 439 10 1109 3477 764879 – 。https://doi.org/ 。 / 。[32] A. Lakshman 和 P. Malik。。Cassandra：一种去中心化结构化存储系统。IGOPS 44, 2 (2010)。[33] G. Lampropoulos、Siakas K. 和 T. Anastasiadis。2019。工业4.0背景下的物联网：概述。IJEK (2019)。[34] Chenning Li、Xiuzhen Guo、Longfei Shangguan、Zhichao Cao 和 Kyle Jamieson。2022。通过并发传输优化 LoRa 网络吞吐量的 CurvingLoRa。发表于 NSDI。[35] K. Lin、W. Wang、M. Bi、Y.and Qiu 和 M. M. Hassan。2016。基于惯性传感器和指纹的工业物联网人员定位。Elsevier COMNET 101 (2016)。[36] D. Liu、T. F. Abdelzaher、T. Wang、Y. Hu、J. Li、S. Liu、M. Caesar、D. Kalasapura、J. Bhattacharyya、N. Srour、J. Kim、G. Wang、G. Kimberly 和 S. Yao。2022。IoBT-OS：优化战场物联网的感知到决策闭环。发表于 ICCCN。[37] C. McNally。2022。Google Nest Wi-Fi 2022 年评测。https://www.reviews.org/ internet-service/google-nest-wifi-review/ 2010

[38] Memoori。2022年。智能商业建筑物联网全球市场。https://memoori.com/portfolio/the-internet-of-things-in-smart-commercial-buildings-2022-to-2027/. [39] S. Micali、M. Rabin 和 S. Vadhan。1999。可验证随机函数。发表于 FOCS。[40] S. J. Moore、C. D. Nugent、S. Zhang 和 I. Cleland。2020。物联网可靠性：综述并提出 5 个关键研究方向。CCF TPCI 2（2020）。[41] S. Munir 和 J. A. Stankovic。2014。DepSys：面向智能家居的信息物理系统依赖性感知集成。发表于 ICCPS。[42] M. Norris、B. Celik、P. Venkatesh、S. Zhao、P. McDaniel、A. Sivasubramaniam 和 G. Tan。2020。IoTRepair：系统性解决商品物联网设备中的设备故障。发表于 IoTDI。[43] OLSR.org。2021。优化链路状态路由协议。https://tinyurl.com/olsrd- wiki。[44] C. Perkins、E. Belding-Royer 和 S. Das。2003。RFC3561：自组织按需距离矢量（AODV）路由。[45] V. Poirot、B. Al Nahas 和 O. Landsiedel。2019。无线 Paxos：空中共识。发表于 EWSN。[46] Emergen Research。2022。农业物联网市场：按系统（自动化与控制系统、传感与监测设备、牲畜监测硬件、水产养殖硬件）、按应用及按地区划分，预测至 2030年。[47] D. Retkowitz 和 S. Kulle。2009。智能家居中的依赖关系管理。发表于 DAIS。[48] M. Ryu、J. Yun、T. Miao、I.-Y. Ahn、S.-C. Choi 和 J. Kim。2015。面向智能农业系统的互联农场设计与实现。发表于 Sensors。IEEE。[49] Samsung Smart Things 2012。Samsung SmartThings。https://www.smartthings. com/.[50] H. Shahid、M. A. Shah、A. Almogren、H. A. Khattak、I. U. Din 和 C. Kumar、N. Maple。2021。基于机器学习的雾计算赋能战场物联网。ACM TOIT 21, 4（2021）。[51] C. Shen、R. P. Singh、A. Phanishayee、A. Kansal 和 R. Mahajan。2016。Beam：终结互联设备的单体应用。发表于 ATC。143–157。[52] Arun Narenthiran Sivakumar、Sahil Modi、Mateus Valverde Gasparino、Che Ellis、Andres Eduardo Baquero Velasquez、Girish Chowdhary 和 Saurabh Gupta。2021。冠层下农业机器人的学习式视觉导航。发表于 Robotics: Science and Systems。[53] J. Song、S. Han、A. Mok、D. Chen、M. Lucas、M. Nixon 和 W. Pratt。2008。WirelessHART：在实时工业过程控制中应用无线技术。发表于 RTAS。[54] I. Stoica、R. Morris、D. Karger、M. F. Kaashoek 和 H. Balakrishnan。2001。Chord：面向互联网应用的可扩展点对点查找服务。SIGCOMM 31, 4（2001）。[55] P. H Su、C.-S. Shih、J. Y.-J. Hsu、K.-J. Lin 和 Y.-C. Wang。2014。面向智能物联网/机器对机器中间件的去中心化容错机制。发表于 WF-IoT。[56] L. Turchet、C. Fischione、G. Essl、D. Keller 和 M. Barthet。2018。音乐物联网：愿景与挑战。IEEE access 6（2018）。[57] D. Vasisht、Z. Kapetanovic、J. Won、X. Jin、R. Chandra、S. Sinha、A. Kapoor、M. Sudarshan 和 S. Stratman。2017。FarmBeats：用于数据驱动型农业的物联网平台。发表于 NSDI。[58] L. Wang、A. Hamad 和 V. Sakthivel。2022。用于仓库管理的物联网辅助机器学习模型。Journal of Interconnection Networks 22, Supp02（2022）。[59] Mi. Xu、C. Liu、Y. Zou、F. Zhao、J. Yu 和 X. Cheng。2021。wChain：适用于多跳无线网络的快速容错区块链协议。TWC（2021）。[60] R. Yang、J. Wang、J. Hu、S. Zhu、Y. Li 和 I. Gupta。2022。Medley：面向物联网网络的成员服务。IEEE TNSM 19, 3（2022）。[61] Y. Yang、J. Wang 和 R. H. Kravets。2005。为网状网络设计路由度量。[62] V. M. Zolotarev。1986。一维稳定分布。数学专著译丛，第 65 卷。277 页。

附录 A：形式化分析

我们对 CoMesh 的性质进行了形式化分析。对于希望跳过本节的读者，我们将研究结果总结如下：

- 继承：当 k 组发生变化时，旧领导者和新领导者所持有的状态相同。
- 安全性：不允许两个会操作重叠设备集合的例程同时执行。
- 活性：不会发生例程死锁。
- 进展：每个例程都会取得进展。
- 我们还计算了超过 f 个设备发生故障时系统的（概率）可用性。

A.1 继承

定义 1. 继承意味着，对于给定的 k-组，在纪元变更前后，或领导者故障后，k-组的旧版本和新版本保持相同的状态。

对于设备 k-组，需要转移给新领导者的状态包括：设备当前的可用性和锁队列（如适用）。对于例程 k-组，新领导者应接收其状态，包括：例程阶段、已经被锁定的设备的标识符（如当时适用），以及已经被释放的设备的标识符（如当时适用）。

引理 1. 当至多一个节点发生故障时，在状态传输阶段之后可保证继承。也就是说，在至多一个节点发生故障的情况下，状态传输阶段之后，旧 k-组领导者持有的状态与新 k-组领导者持有的状态相同。

证明。如果在状态传输阶段期间有新请求到达，请求处理将被延迟，直到状态传输阶段完成。状态传输阶段发生在两种情况下：领导者故障和/或纪元变更。

情况 1：当没有发生纪元变更时，k-组领导者L_o发生故障后，新领导者 L_n 会向所有 k-组成员发送消息，请求获取其本地 k-组状态。所有成员都会将其本地状态回复给 L_n。当且仅当至少有一个收到的状态包含 e时，L_n 才会在重建状态中包含状态条目 e。具体而言，对于可能包含锁队列的设备 k-组，锁队列中的每个例程请求都带有一个序列号，用于表示该请求到达 D_k的 k-组领导者的时间。当且仅当至少有一个收到的队列包含 e时，L_n 才会在重建的锁队列中包含例程条目 e。由于旧领导者 L_o发生故障之前，锁队列中的每个例程条目 e 都已复制到至少法定人数的 k-组成员，因此可以保证L_n 收到完全相同的一组例程请求。L_n 会根据收集到的请求的序列号对队列排序。因此，重建的锁队列与 L_o的最新锁队列完全相同；这也确保了 L_n重建的状态与 L_o的最新状态完全一致。

情况 2：发生纪元变更后，系统进入状态传输阶段：新 k 组的领导者 L_n 请求旧 k 组领导者 L_o 的状态，L_o 据此作出相应回复。状态传输阶段有 4 种可能情况：a) 旧领导者发生故障，b) 新领导者发生故障，c) 旧 k 组的非领导者成员发生故障，d) 新 k 组的非领导者成员发生故障。

情况 2a：如果旧领导者 L_o 在状态转移开始后、完成前发生故障，新领导者 L_n 会向旧 k 组的所有成员请求本地状态，并像情况 1 中那样重新创建 L_o 的锁队列。

情况 2b：如果新领导者 L_n 在从旧领导者 L_o接收状态之前失效，则会选出另一位新领导者 L_n，并向 L_o发送新的状态请求。L_o 会再次回复 k 组状态。

情况 2c：如果旧 k组中的非领导者成员失效，状态转移不会受到影响，新 k组领导者 L_n 将收到正确的状态。

情况 2d：如果新 k组中的非领导者成员失效，将招募另一名成员来替代它。新 k组领导者会将状态分发给新成员。□

引理2。 当最多有 f个节点失效时，状态传输阶段结束后即可保证继承。

证明。如果没有纪元变更（引理 1 情况 1），或者纪元变更期间仅非领导者成员发生故障（引理 1 情况 2c、2d），则每个例程条目 e 都会至少保留在 $f + 1 - f = 1$ 个存活成员上。因此，可以像引理 1 情况 1 那样正确地重建状态。

纪元变更时，如果旧领导者 L_o 在新领导者 L_n 获取状态之前发生故障，L_n （引理 1 情况 2a）可以重建出相同的状态，因为每个状态条目 e 都至少由一个存活的旧 k-组成员维护。如果 L_n 在获取或重建状态之前持续发生故障，最新的新领导者也可以用类似的方式重建状态。

如果 L_o 存活，而 L_n 在纪元变更期间发生故障，工作流程与引理 1 情况 2b 相同。□

A.2 安全性

定理 1。 **[安全性]**在至多发生一次节点故障的情况下，任何两个在设备上存在冲突（即它们触及的设备集合不互不相交）的例程都不能同时执行。

证明。假设例程 R_1 和 R_2 共享设备。我们分别分析 SLA 和 OLA。

顺序锁获取策略（SLA）：首先，我们将分析顺序锁获取策略（第 4.2.1 节）。设 D_k为共享设备中 ID最小的设备，其 k-组领导者记为 L_k。根据定义，如果 L_k 没有发生故障，则至多只有一个例程能够达到法定人数，因此可以获取该设备上的锁。

领导者转换期间不会处理新请求。这些请求会被放入新领导者（L_n）的等待队列。只有在领导者转换完成后，才会启动将此类请求加入锁队列的请求/法定人数流程。因此，证明的其余部分仅处理已获批准的未完成请求。

如果领导者 L_k 发生故障，会出现几种情况：

情况 1：如果 L_k 在将来自 R_1 的请求复制到 k组节点的法定人数之前发生故障，则不会有后续操作。如果未收到确认，R_1的 k组领导者会在超时 T 后重新发送 R_1的请求。

情况 2：如果 L_k 在将 R_1 的请求复制给 k 组成员中的法定人数后发生故障，新领导者 L_n 会在重构阶段向（节点法定人数的）k 组成员查询其状态片段时，将 R_1 添加到锁队列中。

情况3：如果 L_k 在 R_1 的请求到达锁队列顶端后发生故障，则会出现两种子情况：a) 如果该请求已获得节点法定人数的批准（即已转发给节点，且法定人数已批准 R_1 获取锁），新领导者 L_n 会将 R_1 标记为已批准例程，并通知 R_1 的 k 组领导者。b) 如果该请求尚未达到法定人数（无法批准该例程获取锁），则新领导者 L_n 会将 R_1 添加到锁队列中，并继续执行其正常处理流程。

情况4：如果 L_k 在向 R_1的 k 组发送已锁定消息后发生故障，新领导者 L_n会：1）根据幸存节点重建锁队列；2）重新通知 R_1 的 k组领导者，告知其已批准。

乐观锁获取策略（OLA）： 对于同时被 R_1 和 R_2触及的每个设备 D_k，只要 L_k 未发生故障，这两个例程中只有一个例程的锁请求会先由 D_k的 k组领导者 L_k 处理，并就该设备的预锁定达成法定人数。根据定义，只有当所有其他所需设备也可用，且每个所需设备的D_k的 L_k 均未发生故障时，该例程才会继续锁定设备。

如果领导者 L_k 确实发生故障，则会出现两种情况：

情况 1：如果 L_k 在获得预锁定复制的确认法定人数之前发生故障，新领导者 L_n 将不会有关于它的记录。超时后，R_1的 k组领导者会重新发送请求。

情况 2：如果 L_k 在复制了 R_1的预锁定请求后、但在收到 R_1的领导者关于后续操作（锁定以执行，或因预锁定失败而释放）的回复前发生故障，新领导者 L_n 将知道 R_1 已从其组成员处预锁定了 D_k，并将预锁定信息重新发送给 R_1。

情况3：如果 L_k 在收到 R_1 关于获取锁或释放的消息后、但在获得副本复制的法定人数确认前发生故障， R_1的 k组领导者将在超时后重新发送请求，锁定过程将继续进行。

情况4：如果 L_k 在锁定 D_k 并通知 R_1的 k组后发生故障，新领导者 L_n 1将：1）从幸存节点重新构建状态；2）重新通知 R_1的 k组领导者其批准。因此，OLA 策略同样是安全的。由此，CoMesh 在至多发生一次节点故障的情况下是安全的。□

A.3 活性

定理 2。 **[无死锁]** 任意两个例程都不会陷入死锁。

证明。每个 SLA 和 OLA 都违反了死锁所需的不同必要条件。对于 SLA，例程只会等待其已锁定的设备中标识符更高的设备。这意味着，例程不会等待其已锁定的设备中标识符更低的设备。因此，例程之间不存在“等待”循环。至于 OLA，例程不会等待任何已锁定的设备。如果某个设备已被（预先）锁定，新例程将中止，并在稍后尝试获取其涉及的设备的锁。在不存在“持有并等待”的情况下，就不会发生死锁。

领导者变更和故障不会影响 CoMesh 持有的设备锁。同时，引理 1、2 保证了继承。因此，锁分配不会受到领导者变更和故障的影响，也就是说，不会发生死锁。□

最后，我们证明 CoMesh 始终能够取得进展。我们通过以下定理证明归纳步骤（该定理可以迭代应用）。

定理 3。 **[进展性/活性]** 考虑一组正在执行的例程$\mathcal{R}$，其中每个例程仍在等待获取一个或多个锁，以及一组设备 D，它们正在等待这些设备。如果没有更多例程加入$\mathcal{R}$，并且使用 *SLA* 进行锁定，则：集合$\mathcal{R}$ 中至少有一个例程会取得进展（即获得其所需的另一个设备的访问权限）。

证明。考虑集合 D中的最高级ID 设备 D，以及当前持有设备 D的锁的例程 R。那么 R 不可能还在等待其他锁（否则 D 就不会是 D中最高级的ID），因此 R最终会完成。届时， D队列头部的例程将获准访问 D，从而取得进展。□

对于 OLA，对手可能始终能够阻止两个相互竞争的例程取得进展。不过在实践中，我们观察到例程在 OLA 中进展很快。尽管如此，鉴于这一点，我们在 CoMesh 实现中默认使用 SLA。

A.4 系统可用性

虽然 CoMesh 可以容忍 f次故障，但如果故障数超过 f会发生什么？在发生 F（$> f$）次同时故障的情况下，我们计算可用性 P（F），即 = CoMesh 仍能正常工作的概率，也就是说，所有 k组都至少包含（$f+1$）个无故障节点。假设我们有 S 台智能设备、 G k个组，且每组有k 个成员。

(1) When $F \leq f$, $P(F) = 100\%$
(2) When $F \geq f$, the *availability* is:

$$\begin{aligned} P(F) &= P(\text{at most } f \text{ failures across all } k\text{-groups}) \\ &= P(\text{at most } f \text{ failures in one } k\text{-group})^G \\ &= \left(\sum_{i=\max(0,k+F-S)}^{f} \frac{\binom{S-F}{k-i} \times \binom{F}{i}}{\binom{S}{k}} \right)^G \end{aligned}$$

单个 k-组的方程解释如下：给定 F次故障，分子表示选择一个包含 $(k-i)$ 个无故障节点（左项）和 i 个故障节点（右项）的 k-组的方式数，分母则表示选择一个 k-组的所有可能方式数。 i 从 f递减至 0（如果存活节点不足，则递减至更大的数值 ($k+F-S$)）。

从图 3a 可以观察到，当同时故障的数量 F变为 $> f$时，CoMesh 的可用性会下降。然而，即使同时故障数量 F增至 $9\times f$，CoMesh 仍能提供 50% 的可用性。图 3b 显示，要达到特定的可用性百分位数 p，随着组的数量增加，CoMesh 可容忍的 F 下降得非常缓慢——这表明其能够随组的数量扩展。