**概述**

我们研究了大规模和延迟敏感的关键应用的可靠组通信。目标是设计一种协议，即使在频繁出现节点和链路故障的情况下，该协议也能在瓶颈网络链路上实现低负载，并提供稳定的吞吐量和快速传送多播消息。为此，我们提出了 GoCast 协议。 GoCast 构建了一个弹性的覆盖网络，该网络具有距离感知和平衡的节点度。多播消息通过嵌入在覆盖网中的高效树来快速传播。在后台，节点与其覆盖的邻居交换消息摘要（八卦），并拾取因基于树的多播中断、而丢失的消息。

在基于真实互联网数据的模拟表明，与传统的基于八卦的多播协议相比，GoCast 可在没有节点故障时，将多播消息的传递延迟减少8.9 倍 ，或者在20%节点故障情况下减少 2.3 倍。

# 简介

随着 Internet 的快速发展，越来越多的应用程序正在为（或移植到）广域网开发，以便利用在不同地理位置的可用资源：例如网格、对等数据共享，以及协同工作。作为 IBM 高级操作环境 (AOE) 计划的一部分，我们正在开发一种智能基础架构来支持这些大规模分布式应用程序。此基础架构的设计目标包括：自组织以最大限度地减少手动配置、自愈以应对故障、自调整以提高服务质量，以及通过机器学习技术进行自学习、以从过去的经验中提取知识以建议自我组织、自我修复和自我调整组件。

在本文中，我们重点关注基础设施中的可靠组通信协议。群组通信有效地将消息传递给大量接收者。它是编写分布式应用程序的基本实用程序，可用于多种用途，例如：分发系统事件以方便用户监控分布式系统[8]的代理，传播共享状态的更新以保持缓存一致性。

适用于大规模和延迟敏感的关键任务应用的可靠组通信协议应至少满足以下基本要求：

* 可靠的消息传送：即使面对频繁的丢包和节点故障，系统也应该保持稳定的吞吐量；仅针对友好环境进行优化的系统是不可接受的。
* 快速消息传递：许多关键任务应用程序具有实时约束，例如航空公司控制和监控系统。当错过最后期限，消息将变得无用；即使在最后期限内到达，消息价值也会随着时间的推移而贬值。
* 可扩展性：该系统应该自适应处理动态节点加入和离开。随着系统的增长，效率、可靠性和消息延迟的下降应该是适度的。
* 高效的网络资源消耗。在应用层向大量接收者多播消息时，底层网络链路承载不同的流量。重要的是不要对任何物理链路施加极高的负载。

有两个现有协议：可靠多播和八卦多播 有可能满足上述部分但不是全部要求。可靠的多播（如SRM [7]）通过树发送消息、并依赖重传来处理故障。在友好的环境中，它会快速传播消息。然而，先前的研究 [14] 表明，少量受干扰的节点会导致整个系统的吞吐量急剧下降。因此，可靠多播不是可靠组通信的可扩展解决方案。

在八卦多播（例如，双峰多播 [2]）中，节点周期性地选择一些随机节点来传播消息 ID 的汇总（所谓的“八卦”），并从八卦中拾取丢失的消息。八卦路径中的冗余解决了节点和链路故障。Gossip多播即使在不利的环境中也能提供稳定的吞吐量，但多播消息的传播比可靠的多播慢，因为延迟与八卦周期成正比，且总在实际消息之前交换八卦，会导致额外的延迟。此外，我们的评估表明：由于对网络拓扑的无视，大型系统中的随机八卦会对某些底层网络链路施加极高的负载。

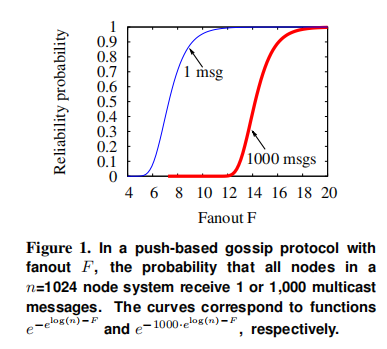
我们提出GoCast（gossip‑enhanced overlay multicast）来解决现有协议的局限性。它结合了可靠多播的优点（拓扑感知和快速消息传播）和八卦多播的优点（稳定的吞吐量和可扩展性），同时避免了它们的局限性。 GoCast 构建了一个弹性的、接近感知的覆盖网络，具有平衡的节点度。与可靠多播一样，消息通过嵌入在覆盖网中的高效树快速传播。在后台，节点与其覆盖网的邻居（而不是随机节点）交换消息摘要（八卦），并拾取由于基于树的多播中断而丢失的消息。节点收到包含消息I D 的八卦的次数（即冗余）由节点的覆盖邻居的数量控制。

## GoCast 与 Gossip多播

Gossip 多播协议（例如双峰多播）有两个关键要素：gossip路径的冗余和随机性。这些是八卦组播能够提供稳定的吞吐量和可靠的消息传递的根本原因。GoCast 保留了冗余和随机性的精神：通过覆盖网中的多个不相交路径实现冗余，并通过一些随机的覆盖链接实现受控随机性。与Gossip多播协议相比，GoCast 有几个优势：

由于Gossip多播中的完全随机性（与GoCast中的受控随机性相反），节点收到包含给定消息ID的八卦次数变化很大。在基于推送的八卦协议（如双峰多播）中，在接收到新消息时，节点将消息ID 八卦给 F 个随机节点：F被称为gossip的扇出。我们的模拟表明，在扇出为 5 的情况下，在1024个节点的系统中，大约 0.7% 的节点从未听到过给定消息，而一些节点听到该消息多达 19 次。

在使用带扇出 F 的基于推送的八卦协议的 n 节点系统中，所有节点收到通过八卦网传输一条指定消息的概率是e−elog(n)−F[6]，因此，所有节点听到大约 1 ,000 条消息的概率 e−1000·elog(n)−F，图 1 绘制了 1024 个节点系统的这些概率，即便系统没有任何故障，当扇出小于15时，完整接收到10000条消息的概率也低于0.5。注意0.5 的可靠性对于关键任务应用程序来说不够高。此外，使用扇出 1 5 引入的八卦流量是 GoCast 的三倍（第 2 节），八卦多播中多播消息的传播比 GoCast 慢几倍（第 3 节）。



相比之下，GoCast 并不完全依赖随机性：它通过调整节点度来完全控制节点接收有关消息的八卦的次数。只要覆盖网保持连接，节点就会至少收到一次关于消息的八卦。

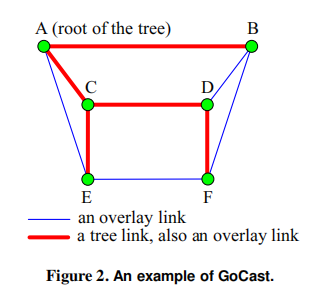
我们的评估表明，对于具有数千个节点且每个节点有六个邻居的系统，即使在 25% 的节点同时发生故障后，GoCast 中的覆盖仍然保持连接。

在本文其余部分安排如下：第2节介绍 GoCast 协议；第 3 节 通过广泛的模拟比较了几个可靠的组通信协议；相关工作在第 4 节中讨论；第 5 节 总结本文。

# GoCast协议

GoCast 为延迟敏感的大规模关键应用提供可靠的群组通信。无论系统多大，它都会在每个节点上产生恒定的低开销。节点加入或离开只会影响少数其他节点，这些节点会在本地处理更改。 GoCast 是自调整的：随着对底层网络的了解越多，它的效率和消息延迟会迅速提高。

GoCast 将节点组织成覆盖网，并通过覆盖网中嵌入的高效树和覆盖邻居之间交换gossip八卦来传播多播消息。以下假设覆盖网和树已经建好，先描述消息传播协议，然后再描述覆盖网和树的构建协议。



## 快速及可靠的消息传播

在正常操作下，多播消息通过覆盖网中嵌入的高效树快速传播（图2）。

树链接也是覆盖链接。覆盖链接和树链接都是无向的，可在任一方向上传播消息。我们将通过覆盖链接或树链接直接相连的两个节点分别称为覆盖邻居或树邻居，两个树邻居也是重叠邻居。仅出于维护的目的，树在概念上有一个根，但任何节点都可在不先将消息发送到根的情况下启动多播。

多播消息沿着树链接，从消息源传播出去。我们通过图2中的示例，对此进行说明：假设节点 D 想要向系统中的所有节点多播一条消息。节点 D 将消息发送到其树邻居 C 和 F 。节点 D 不将消息发送到节点B，因为 B 和 D 不是树邻居，尽管它们是覆盖邻居。每个收到消息的节点立即将消息转发给它的树邻居，除了消息到达的节点。每个节点记住它最近转发的消息的ID，以避免重复转发消息。如果没有故障，消息会迅速传播到所有节点（对于1024个节点的系统，不到半秒）且每个节点只接收一次消息。

然而，错误是不可避免的。在后台，节点与其覆盖网的邻居交换消息摘要，并拾取由于基于树的多播中断而丢失的消息。每隔 t 秒，节点 X 以循环方式选择其覆盖邻居中的一个 Y 来发送消息摘要（也称为“八卦”）。

八卦周期 t 是根据消息速率动态可调的，在我们当前的设计中 t =0.1 秒，这是双峰多播 [2] 建议的。注入系统的每条消息都有一个唯一的标识符。节点 P 注入的消息标识符是 P 的IP地址和 P 本地分配的单调递增序列号的串联。节点X 发送给节点 Y 的八卦包括 自从上次X发给Y的gossip后，X收到的消息，但不包含从Y那收到的消息ID。

收到八卦后，如果节点Y发现八卦中包含一些没收到过的消息ID，则 Y 向节点 X 请求这些消息。然而，在大多数情况下，消息通过树快速传播，使得节点在从其覆盖邻居接收到关于该消息的八卦之前，已收到来自树的消息。

如果节点X有s个覆盖邻居，它每s\*t秒向它的邻居Y发送一个八卦，其中 t 是八卦周期。因为 t 很短并且 s 很小（通常s=6，见第 2 .2 节），通常在两个覆盖邻居之间交换八卦的时间少于一秒一次。如果在此期间没有多播消息，则八卦可被保留。

八卦包很小，只包含不到一秒内收到的消息ID。节点 X 只向它的每个邻居传播一次消息ID。在将消息ID 闲聊给最后一个邻居后，节点X 等待一段时间b，然后如果X没有收到对该消息的进一步请求，则回收该消息的内存。等待周期 b 应至少是节点 X 与其邻居之间的最大往返时间 (RTT) 的倍数，以便为邻居提供足够的时间来请求消息。我们将等待时间 b 设置为两分钟。

在一些短暂时刻，由于节点或链路故障，树可能断成片段。在树片段间，消息通过覆盖邻居之间交换的八卦传播；在树片段内，消息通过连接该片段的其余树链接不停地传播。在图 2 中，假设节点B开始多播。它将消息发送到它的树邻居A。如果节点 A 在将消息转发到树邻居C 之前发生故障，则基于树的多播不会将消息传递到节点C、D、E和F。在某些时候，节点B 向节点D 发送八卦消息。节点D 发现它没有收到消息，因此向节点B 请求消息。节点D 收到消息后，立即将消息转发给它的树邻居C和 F 。消息进一步立即沿着连接树片段的其余树链接不停地转发消息。断开的树会很快被修复（第2.2 和2.3节），消息将再次纯粹地沿着树传播。

多播消息既无条件地通过树传播，也有条件地通过重叠邻居之间同时发生的八卦传播。节点通过两个通道接收到冗余消息的可能性很小。假设一个节点收到一个包含消息ID的gossip，发现自己没有收到消息。它从八卦的发送者那里获得消息，但稍后消息又从树链接到达。换句话说，消息通过八卦比通过树传播得更快。发生这种情况的可能性非常低，因为消息通常支持通过高效树的速度要快得多。我们的模拟表明，在具有典型Internet延迟和0.1秒八卦周期的1024节点系统中，上述情况发生的概率仅为0.02，即平均每个节点接收消息1.02次。

这 2% 的开销可以通过多种方式进一步减少：

1. 如果节点X 已收到或正在接收通过八卦发现的多播消息，而另一个节点Y 正试图通过树链接向 X 发送相同的消息，则 X 立即中止与 Y 的传输。因此，当消息很大时，这 2% 的开销不是完全多播消息。
2. 当节点收到包含消息ID 的 gossip 时，它会延迟向 gossip 的发送者请求消息，直到消息至少在 f 秒 前被源注入系统。选择阈值 f 以允许消息有足够的时间先通过树传播。我们建议将 f 设为多播消息通过树到达节点的 90% 延迟。对于1024个节点的系统，设置f=0.3秒对多播消息的传递延迟几乎没有影响，同时将节点收到冗余多播消息的概率降低到0.0005。这种优化要求多播消息和八卦携带自消息被注入系统以来所经过的时间，这可通过搭载和累加消息从源传播时的传播延迟和等待时间来估计。

在本节中，我们已经描述了假设覆盖网和组播树已经准备就绪的消息传播协议。接下来，我们继续描述构建覆盖网和组播树的协议。

## 构建覆盖网协议

我们的目标是构建一个度数受限且连接丰富的覆盖网络，且主要由低延迟链接组成。由我们协议构建的覆盖网具有几个显着特征。 (1) 在现有协议构建的覆盖网中，节点度数没有得到严格控制。相比之下，GoCast 中的每个节点都有大致相同数量的覆盖邻居，以分散维护开销和八卦开销。 (2) 在现有协议中，节点要么没有随机邻居，要么随机选择至少一半邻居。相比之下，GoCast 中的大多数节点只有一个随机邻居，而所有其他邻居都是根据网络邻近性选择的。此方法生成的叠加层既稳健又高效。

我们仔细选择系统参数，以在弹性和效率这两个相互冲突的目标之间取得良好的平衡。覆盖网的连通性（即两个节点之间不相交路径的数量）直接影响 GoCast 在节点或链路故障时的可靠性。更高的节点度导致更好的连通性，但引入更高的协议开销，因为节点需要维护更多邻居，八卦也会发送给更多邻居。假设节点具有相似的容量，我们还希望节点度数尽可能统一，以便施加在每个节点上的协议开销大致相等（根据节点容量调整节点度数可包含在本协议中，但超出了本文的范围）。覆盖网是非结构化的；它不要求特定的拓扑结构。无论覆盖网的初始结构如何，它都会自动适应，以便几乎所有节点都能收敛到设计时选择的目标节点度数。

除了目标节点度，另一个重要的设计选择是选择节点邻居的方式。它会影响覆盖网的连通性、消息延迟以及对底层网络链路的压力：一方面，根据随机图论，在随机节点之间添加链接可提高覆盖网的连通性；另一方面，在网络中就近的节点之间添加低延迟链路可降低消息延迟，消耗更少的网络资源，并减少瓶颈网络链路的压力。

GoCast 通过投入少量覆盖链接来连接随机节点，并根据网络邻近性选择所有其他覆盖链接，来实现良好的平衡。我们的评估表明这种方法产生的叠加层具有低延迟和高连通性。

在深入研究我们的协议之前，我们定义了一些符号。组件由一组节点组成，这些节点通过覆盖链接直接或间接连接。我们将连接随机选择的邻居的覆盖链接称为随机链接，将基于网络接近度选择的覆盖链接称为附近链接。与随机链接直连的两个节点是随机邻居，由附近链接直连的两个节点是附近邻居。令随机度Drand(X)和附近度Dnear(X)分别表示节点 X 的随机邻居和附近邻居的数量。让Cdegree、Crand和Cnear分别记下目标节点度、目标随机度和目标附近度，则有：

Cdegree = Crand + Cnear。

Cdegree、Crand和Cnear是在设计时选择的常数。理想情况下，每个节点X 具有相同的度数：

Drand(X) = Crand和 Dnear(X) = Cnear。

本文的一个主要贡献是为这些参数找到了一个好的设置：Crand = 1 和Cnear = 5 。我们发现，没有任何随机邻居(Crand = 0)的情况下，即使没有任何节点或链接，覆盖网也会被分割失败。这是因为附近链接不连接远程组件。每个节点只有一个随机邻居（Crand = 1），其覆盖的连通性几乎与每个节点使用多个随机邻居的覆盖的连通性一样好。直观地，附近链接连接一组靠近的节点，随机链接连接远程节点组。例如，假设一个系统由 500 个美国节点和 500 个亚洲节点组成。只有附近链路，系统被分解成对应于两个地理区域的两个组件。在内部，每个组件是紧密相连的。通过向每个节点添加一个随机链接（总共500个随机链接，一个链接连接两个邻居），我们预计平均有250 个随机链接连接美洲组件和亚洲组件，这大大增强了整个系统节点的连通性。此外，我们发现每个节点六个邻居提供了足够的连通性。例如，使用此配置，具有数千个节点的系统即使在 25% 的节点同时发生故障后仍保持连接。

目标节点度数是在设计时选择的。下面我们描述了在运行时强制执行节点度数，并为覆盖网选择高质量链接的协议。

### 节点加入

每个节点都要知道系统中节点的随机子集。这信息是通过在覆盖邻居之间交换的八卦上，搭载一些随机节点的 IP 地址来维护的。由于篇幅限制，我们省略了这个部分成员协议的细节。有兴趣的读者可以参考 [ 5, 16]。[5]已表明，对于八卦协议，“均匀”随机的部分成员列表几乎与完整的成员列表一样好。

当一个新节点 N 加入时，它通过某种带外方法知道至少一个已在覆盖网中的节点P。节点 N 联系节点P 以获得 P 的成员列表S。节点 N 暂时接受S作为其成员列表，节点 N稍后可能会向S中添加或删除节点。

节点 N 随机选择S中的Crand个节点作为其随机邻居，并与它们中的每一个建立 TCP 连接。覆盖邻居之间的所有通信都通过这些预先建立的 TCP 连接（换句话说，非重叠邻居节点之间的通信使用UDP，例如非邻居节点之间的RTT测量）。

在S 的节点中，理想情况下，节点 N 应该选择那些对 N 有最小延迟的节点作为 N 的附近邻居。但是 S可能很大，包括数百个节点。N 测量到 S 中每个节点的RTT 会引入大量流量和漫长的等待。相反，节点 N 使用算法来估计网络距离，并选择S中具有最小估计延迟的Cnear个节点，来作为其初始的一组附近邻居。随后，节点 N 逐渐测量到S中节点的RTT，并将长延迟链路切换到低延迟链路，随着时间的推移提高覆盖效率。我们使用三角启发式 [13] 来估计延迟时间。限于篇幅，细节不再赘述。

如果新节点N选择一个节点X作为它的邻居，N向X发送一个请求。只有当X的节点度不是太高时，才会接受这个请求。对于添加一个随机链路，Drand(X) < Crand + 5；用于添加附近链接Dnear(X) < Cnear + 5。如不满足约束，则节点 N 必须尝试另一个节点。对于节点 X 接受节点 N 作为其附近邻居，它还必须满足第2.2.3 节中描述C2条件：直观地，这个条件规定，如被接受，节点 N和 X 之间的链路一定不是 X 维护的最差覆盖链路。

节点加入协议保证节点没有过多的邻居，但不能确保节点度Drand(X)和Dnear(X)最终收敛到目标度Crand和Cnear。下面描述的覆盖网适配协议实现了这一目标，它们还自动处理节点离开和故障。当一个节点离开后，它以前的邻居会选择其他节点作为新邻居。

### 维护随机邻居

每r 秒，每个节点 X分别执行维护其随机邻居的协议和维护其附近邻居的协议。周期 r可根据覆盖网的稳定性（即维护需要）动态调整，在我们当前的设计中 r = 0 .1秒。虽然周期r很短，但在维护期间，大部分时间不需要操作。

另请注意，节点的维护成本和八卦开销与系统的大小均无关。

如果节点X的随机度Drand(X)等于目标随机度Crand，则不需要采取任何行动；

如果Drand(X) < Crand （这可能是由于X的随机邻居故障），X从其成员列表中随机选择一个节点，并建立到该节点的随机链接；

如果Drand(X) > Crand，则节点X 尝试通过以下操作之一删除一些随机邻居。

1. 如果Drand(X) ≥ Crand + 2，节点X随机选择它的两个随机邻居 Y 和 Z，并要求Y 建立到 Z 的随机链接。然后节点X 丢弃它到节点Y 和 Z 的随机链接。通过做这样，节点X 的随机度减少 2 ，而节点 Y 和 Z 的随机度没有改变。
2. 如果节点 X 的随机邻居之一 W 的随机邻居数量多于Crand 个随机邻居，则节点 X 会丢弃 X 和 W 之间的随机链接。这会将 X 和 W 的随机度降低1，同时仍保持它们的随机度 ≥ Crand。
3. 如果以上两个条件都不满足，则节点 X 的随机度必须为Crand + 1，并且 X 的所有随机邻居的随机度必须等于或小于Crand。在这种情况下，不采取任何行动，节点 X 的随机度保持在Crand+1。

可以证明，当覆盖稳定时，每个节点最终都有Crand或Crand + 1个随机邻居。我们的评估表明，大约 8 8% 的节点具有Crand随机邻居，而12% 的节点具有Crand + 1 个随机邻居。

### 维护附近邻居

除了维护随机邻居外，每隔r 秒，节点 X 还执行一个协议来维护它的附近邻居。该协议不同于用于维护随机邻居的协议，因为它在添加或删除链接时考虑网络邻近性。它试图将节点X 的附近度数限制在Cnear 或Cnear+1，同时选择靠近 X 的节点作为 X 的附近邻居。节点 X 运行三个子协议：

* 一个用低延迟链路替换 X 的长延迟附近链路；
* 一个是当Dnear(X)<Cnear时添加更多的附近链接；
* 一种是当Dnear(X)≥Cnear+2时丢弃附近的长延迟链路。

#### 替换附近邻居

节点X 根据估计延迟对它的成员列表S中的节点进行排序。从具有最小估计延迟的节点开始，节点 X一个接一个地测量与S中节点的实际延迟。在每个维护周期（每 r 秒）中，节点X 仅测量S中一个节点的RTT。

随着覆盖网的稳定，改进的机会就会减少。可以相应地增加维护周期r以减少维护开销。我们将 r 的动态调整作为未来工作的主题。

假设节点 X 在当前维护周期内测量到节点Q 的 RTT。如果满足以下所有条件，节点 X 将添加节点Q 作为其新的附近邻居，并删除其现有的附近邻居U。

C1) 节点 X 至少有一个附近邻居U ，其当前附近度不是太低：Dnear(U) ≥ Cnear - 1。  
否则，节点X的所有附近邻居的度都低到危险的程度。删除与其中之一的链接会危及叠加层的连通性。  
在满足此条件的节点 X 的附近邻居中，要替换的节点U 被选为对节点X具有最大延迟的邻居。

C2) Dnear(Q) < Cnear + 5。这要求新邻居候选Q的邻近度不能太高；

C3) 如果Dnear(Q) ≥ Cnear，则RTT(X, Q ) < max nearby RTT(Q)必须成立。这里的RTT(X, Q)是节点X和新邻居节点Q之间的RTT，max nearby RTT(Q)是节点Q和Q的附近邻居之间的最大RTT。

如果不满足此条件，则节点 Q 已经有足够的附近邻居，且节点Q 和 X 之间的链路比 Q 当前拥有的任何附近链路都还差。即使节点 X 此时将链接添加到节点Q，Q 也很可能很快就会断开链接。因此节点 X 不添加此链接。

C4) RTT(X, Q) ≤ RTT(X,U)。这里节点 Q 是新的邻居候选者，节点U 是要替换的邻居（由条件 C1 选择）。为了避免徒劳的小调整，这个条件规定只有当 Q 明显优于当前邻居 U 时，节点X 才会采用新的邻居Q。

再保证两边near连接度前提下，用一个低rtt的链接替换高rtt的near链接

在许多启发式测试中，上述条件在解决许多冲突方面特别有效。

目标：在适应过程中保持覆盖网的连通性，在没有全局信息的情况下最小化链路改变数，并快速收敛到稳定状态。例如，条件 C1 是我们做出权衡的一个很好的例子。由于条件C1，节点U 的邻近度在节点 X 断开到U的链接之后和 U 在下一个维护周期中添加更多附近链接之前的短暂时刻可低至Cnear‑ 2。

如果我们将条件C1从 Dnear(U) ≥ Cnear - 1更改为Dnear(U) ≥ Cnear，则该下限可以增加到Cnear - 1 。然而，我们的评估表明，这种变化会产生一个覆盖网，其链接延迟明显高于我们当前解决方案产生的链接延迟，因为更少的邻居满足这个新条件，并有资格成为被替换的候选者。

使用我们推荐的设置：Crand = 1 和Cnear = 5，自适应过程中节点度数的下限为4，这足以在短瞬态期间维持覆盖网的连通性。

最初，节点X 将其成员列表S中的节点按估计延迟递增方式进行排序，并逐一测量它们的RTT。一旦测量了S中的所有节点，就不再使用估计延迟。但是节点 X 仍然不断尝试通过以循环方式考虑S中的候选节点来替换其当前附近邻居。希望以前不满足某些条件 C1‑C4 的一些节点现在可以满足这些条件，并可用作新的附近邻居。

#### **添加附近邻居**

如果节点X的附近邻居少于Cnear ，则 X需要添加更多的附近邻居以维持覆盖的连通性。为了分散负载，在每个维护周期中，节点X 最多添加一个新的附近邻居。类似于替换附近邻居的过程，节点 X以循环方式从其成员列表S中选择一个节点Q，如果满足条件C1 和 C2 ，则将Q 加为其新的附近邻居。这些条件规定Q 没有过多的邻居，并且节点 X 和Q 之间的链路不比 Q 当前拥有的最差附近链路差。

#### **删除附近邻居**

如果节点X的附近邻居数量过多（例如，一些新节点已添加到 X 的链接），则 X 开始丢弃一些附近邻居以减少不必要的协议开销。尽管目标附近度为Cnear，但只有当Dnear(X) ≥ Cnear +2时，节点 X 才开始丢弃附近邻居。这允许节点的邻近度稳定在Cnear或Cnear+1。我们的评估表明，在这种策略下，最终大约 70% 的节点有Cnear附近邻居，大约 30% 的节点有Cnear +1附近邻居。一种替代方法是在Dnear(X) = Cnear + 1时再丢弃一个附近的邻居。我们的评估表明，与我们目前的解决方案相比，这激进的策略将链接更改的数量增加了近三分之一，并且需要更长的时间来稳定覆盖。

当Dnear(X) ≥ Cnear + 2 时，节点 X 尝试丢弃Dnear(X)－Cnear个附近邻居。要丢弃的候选邻居是那些满足上述条件C1 的节点，即附近度数没有危险地低的节点U， Dnear(U) ≥ Cnear - 1。同样，避免丢弃到低度数节点的链接有助于维护节点连通性适应期间叠加。节点 X 对满足此条件的附近邻居进行排序，并将那些具有最长延迟的邻居丢弃，直到 X 的附近度降低到Cnear或没有附近邻居满足条件C1。

## 构建组播树协议

GoCast 以去中心化的方式选择覆盖链接来构建嵌入在覆盖网中的高效树。树跨越所有节点并快速传播消息。构建树的算法在本质上类似于经典的距离矢量多播路由协议(DVMRP) [ 15]，但请注意 GoCast 只需要一棵树。树在概念上有一个根，树链接是根和所有其他节点之间的最短路径（就延迟而言）上的覆盖链接。如果根失败，它的邻居之一将接管它的角色。最初，覆盖网中的第一个节点充当根。每隔 15 秒，根节点就会在覆盖网中的每个链接中泛洪心跳消息，以帮助检测覆盖网和树的故障（例如，分区）。由于空间限制，我们省略了树协议的细节，因为它是标准的。

# 实验结果

我们构建了一个事件驱动的模拟器来评估GoCast。该模拟器由 6100 行 C ++ 代码组成，在 Linux 上运行。它模拟了一个完整的系统，包括消息传播、节点和链路故障、网络拓扑和链路延迟。为了能扩展到数千个节点，我们不模拟网络级数据包细节。在 2 .4GHz 机器上模拟一次 8192 节点系统的运行大约需要三个小时。

模拟器使用来自 King 数据集 [ 4] 的真实互联网延迟，该数据集是从互联网中 1740 个 DNS 服务器之间的 真实RTT测量中提取的（原始数据集包含超过 1740 台服务器，但我们排除了那些测量值为空的服务器）。

我们将 RTT 除以二以获得单向延迟。平均和最大单向延迟分别为91 毫秒和 399 毫秒。当模拟的节点数量大于实测的 DNS 服务器数量时，我们在单个 DNS 服务器站点模拟多个节点。

除非另有说明，否则我们将在 1024 节点系统上报告结果，模拟工作如下：最初，所有节点同时启动，随机指定一个节点作为树的根。对于目标节点度Cdegree，每个节点初始连接到Cdegree/2个随机节点。初始化后，节点的平均度为Cdegree，所有的邻居都是随机选择的。然后，覆盖网和树会根据 GoCast 的维护协议在模拟时间内适应500秒。500 秒后，多播消息以每秒100条消息的速率从随机源节点注入覆盖网。

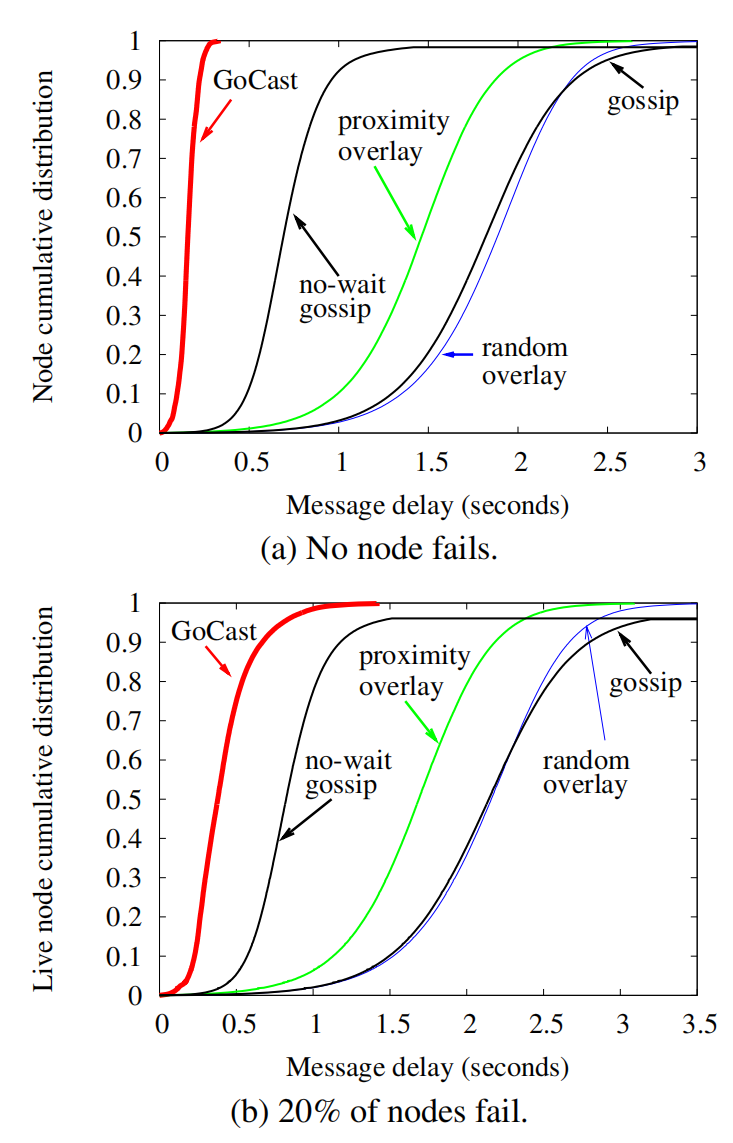


图3：多播消息在不同协议下的消息传输延迟(1024节点)，Y轴为节点累计分布

GoCast的参数如下。节点每 t = 0.1 秒发送一次八卦（由双峰多播[2] 建议）。每 r = 0 .1 秒 ，一个节点就会醒来以维护其邻居。默认目标节点度数为Crand = 1和Cnear = 5。

我们先阐述不同多播协议下的消息传播延迟的比较结果。延迟是从随机源节点注入的 1000 条多播消息的平均值。图 3 (a) 显示了没有节点故障时的理想结果。 “GoCast”曲线表示完整的 GoCast 协议，其中消息通过树和覆盖邻居之间交换的八卦传播。

“Proximity overlay”和“random overlay”曲线代表Gocast简化版本，消息只通过覆盖邻居之间交换的八卦传播，系统既不维护也不使用树。“Proximity overlay”和“random overlay”的区别在于“proximity overlay”中的每个节点维护5个附近邻居和1个随机邻居，而“random overlay”中的每个节点只维护6个随机邻居。

图 3 (a) 中的gossip曲线表示类似于双峰多播中使用的基于推送的八卦协议。每 t = 0.1秒，每个节点都会向随机节点发送一个八卦。 gossip fanout 为5，即一个节点将接收到的多播消息的ID gossip给5个随机节点（每个 gossip 周期一个节点）。如果 gossip 的内容表明接收者不知道这些多播消息，则 gossip 的接收者会向发送者请求一些多播消息。“no-wait gossip”曲线所代表的系统则不同。在“no-wait gossip”中，在收到多播消息后，一个节点立即将消息八卦给其他5个节点，而无需等待下一个八卦周期（换句话说，八卦周期 t = 0）。当消息速率很高时，“no-wait gossip”比“gossip”引入更高的八卦流量。我们在这里使用它来展示八卦多播的基本性能限制。

完整的 GoCast 协议（“GoCast”曲线）分发消息的速度明显快于所有其他协议，因为多播消息主要通过由低延迟链路组成的高效树不停地传播。平均超过 1000 次运行，多播消息在 0.33 秒内到达每个节点。八卦多播（“gossip”曲线）是消息传播中最慢的。此外，对于扇出5，由于八卦的完全随机性，一些节点从未收到 1000 条多播消息中的一些。较短的八卦周期可以以增加八卦流量为代价减少消息延迟。然而，即使是“nowait gossip”也没有“GoCast”快，原因有二个:

(1) GoCast 中的树链接具有低延迟；

(2) 为了避免冗余发送潜在的大组播消息，“nowait gossip”总是先发送八卦，然后根据请求发送实际的组播消息，这会产生额外的延迟。与“gossip”一样，“nowait gossip”中的一些节点永远不会收到一些多播消息。

“random overlay”中的消息延迟类似于“gossip”中的消息延迟，但“random overlay”中的每个节点都会收到消息，这是由覆盖网的连通性保证的。由于其低延迟链接，“Proximity overlay”消息传播速度明显快于“random overlay”和“gossip”。

图 3 (b) 对不同的协议进行了压力测试，当 20% 的节点在模拟的 500 秒内同时发生故障时。失效节点是统一以随机方式选取的。节点故障后，多播消息被注入系统，但系统不执行任何GoCast 维护协议来修复覆盖网或树。Y 轴表示活动/正常节点的累积分布。

20% 节点关闭的情况下，overlay 仍然保持连接。“GoCast”、“Proximity overlay”和“random overlay”仍能将每条多播消息传送到每个活动节点。

比较图3(a)和3(b)中的“gossip”曲线，我们看到，对于gossip 多播，更多活动节点在节点故障时不会接收到一些多播消息。

在消息延迟方面，“GoCast”比其他四种协议对节点故障更敏感。这是因为，在没有任何修复措施的情况下，GoCast 中的树被分解成碎片，消息必须通过八卦在碎片间传播。一旦消息命中树片段中的一个节点，它就会通过连接片段的剩余树链接不停地传播到片段中的其他节点。这是“GoCast”仍然明显优于“Proximity overlay”的主要原因。

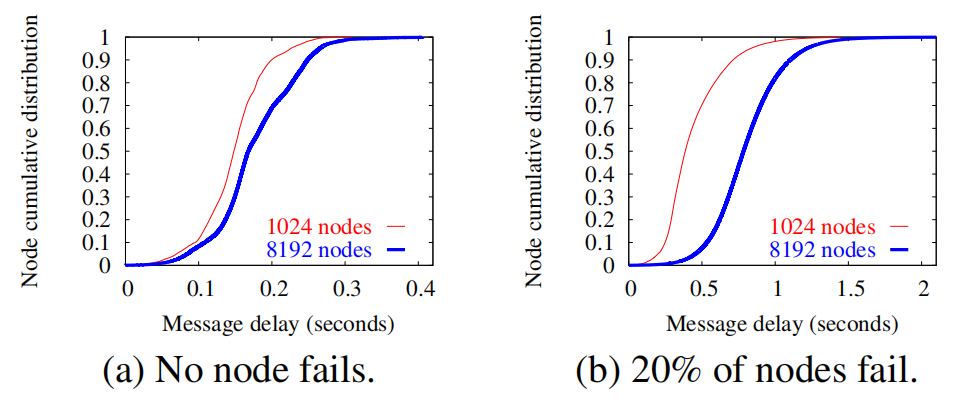


图 4.比较1024节点系统和8192节点系统之间GoCast中的消息延迟。

图 4(a) 和 4(b) 比较了1024节点系统和8192节点系统之间完整GoCast协议中多播消息延迟。当没有节点发生故障时，两个系统之间的差异很小。对于8192个节点的系统，平均超过1000条多播消息，一条消息在0.42秒内到达所有节点。

当 20%的节点发生故障时，两个系统之间的差异就会变大。8192节点系统的曲线尾部较长。8192个系统中少量节点的消息延迟比1024个节点系统中最长的延迟长约 60%。当很大一部分节点同时发生故障时，较大系统中的树会比较小系统中的树分裂成更多碎片。多播消息通过缓慢的八卦在树碎片之间传播，这就是图 4 (b) 中的差异比图 4 (a) 中的差异更显着的原因。总的来说，随着系统规模增加八倍，消息延迟的增加是适度的，表明 GoCast 是可扩展的。

接下来我们评估GoCast 自适应覆盖网和树的能力。这个实验模拟了一个 1024 节点的系统。最初，所有节点同时启动，每个节点发起三个随机链接；初始化后，平均节点度为六；GoCast 的维护协议随时间调整覆盖网和树：目标随机度为1，目标附近度为5。

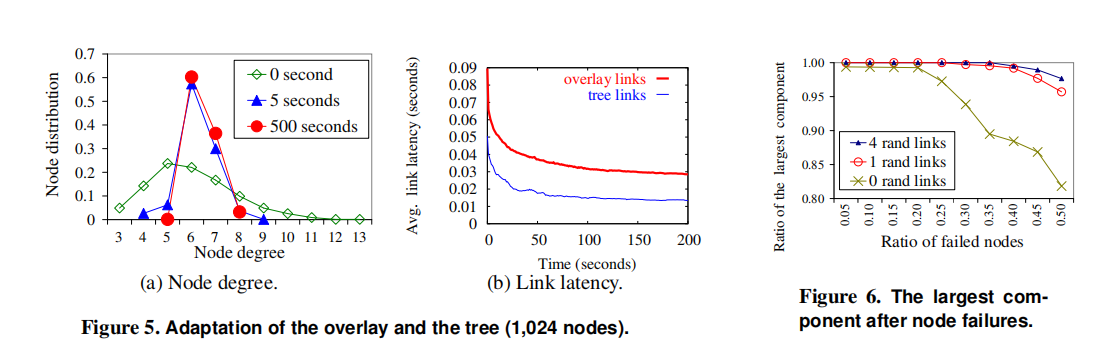


图 5 (a) 显示了节点度（即随机度和附近度之和）随时间的分布。

* 最初，节点度数变化很大（见“0秒”曲线）；只有22% 节点具有目标度数 6 。
* 在 GoCast 维护协议下，节点度快速收敛。5秒后，57% 节点的度数为6；
* 500秒后，60% 的节点度数为6，平均节点度数为6.4。

图 5 (b) 显示了覆盖链接和树链接随时间的平均延迟。此图仅绘制前 200秒，以使有趣的开始阶段可读。

覆盖链接的延迟是随机链接(Crand = 1) 和附近链接(Cnear = 5 )的平均值。

最初，所有覆盖链接都是随机选择的。在开始阶段，随着许多长延迟链路被低延迟链路取代，链路质量迅速提高。

这种改进是图3 中“Proximity overlay”比“Random overlay”传播消息更快的主要原因。

大约60秒后，随着节点越来越难找到更好的邻居，改进速度变慢。

树链接的延迟低于覆盖链接，因为 GoCast 倾向于选择低延迟的覆盖链接来构建树。100秒后，树链接的平均延迟仅为15.5毫秒，而随机节点之间的平均延迟为91毫秒。这种良好的链接质量是“GoCast”显着优于图 3 中所有其他协议的部分原因。

我们现在将注意力转向随机链接数量对覆盖网弹性的影响。在图 6 中，我们将失败节点的比率从 0.05 更改为0.5（即 5 % 到 50%），并报告最大连接组件中活动节点的比率 q （如果覆盖保持连接，则 q=1）。

在没有任何随机链接的情况下，覆盖已经被分区，即使没有任何故障，因为远程组件没有连接。

只有一个随机链接，即使在 25% 的节点同时发生故障后，覆盖仍保持连接。每个节点一个随机链接和每个节点四个随机链接之间的弹性差异很小，这证明我们对每个节点使用一个随机链接是合理的。

限于篇幅，我们简单总结一下其他仿真结果如下。详细数据将在本文的较长版本中呈现。

1. 从只有随机链接的随机结构开始，在我们的自适应协议下，覆盖网迅速收敛到稳定状态。随着时间的推移，每秒更改的链接数呈指数下降。
2. overlay 链接的平均延迟几乎与随机链接的数量呈线性增长，这再次证明我们每个节点只使用一个随机链接。
3. overlay是可扩展的；当系统规模从256个节点增加到8192个节点时，overlay的直径从6跳增加到10跳。
4. 与使用扇出5 的 基于推送的八卦协议相比，GoCast 将瓶颈网络链路上的流量减少4‑7 倍。本实验中使用的网络拓扑是互联网自治系统的大规模快照。
5. 不能通过简单地增加八卦扇出、来显着降低基于推送的八卦协议中的消息延迟。当fanout从5增加到9时，消息延迟只减少了5%左右；进一步将扇出增加到 15 对消息延迟几乎没有影响。

# 相关工作

GoCast 在覆盖网络、多播协议和八卦协议中找到上下文，其中 Araneola [12] 和 Bimodal Multicast [2] 与 GoCast 最相关。

在我们开发 GoCast 的同时，Melamed 和 Keidar 最近提出了Araneola [12]。它将节点组织成覆盖网络并在覆盖邻居之间运行八卦协议以实现可靠的多播。Araneola 和 GoCast 使用类似的协议来调整节点的随机邻居。 Araneola 和 GoCast之间有几个主要区别。

1. Araneola 仅通过覆盖邻居之间交换的八卦来传播多播消息。相比之下，大多数时候 GoCast 通过高效树不停地传播消息。
2. Araneola对节点保留的附近邻居数量没有严格控制。
3. 在Araneola 中，至少有一半的覆盖链接是随机选择的。

双峰多播[2]分两个阶段传播消息，首先通过尽力而为的不可靠多播，然后通过随机节点之间交换的八卦。

Bimodal Multicast 和 GoCast 都打算结合基于树的多播和 gossip 多播的优点，但它们的方法和结果性能却大不相同：

1. 双峰多播采用随机节点 gossip消息摘要，具有第1节中描述的缺点，例如，需要高扇出以实现高可靠性，并且由于其对底层网络的遗忘、而对瓶颈网络链路施加高负载拓扑。
2. 双峰多播使用手动配置的层次结构来处理不可靠的多播，而 GoCast 自动创建和调整树。
3. 在双峰多播中，在尽力而为的多播阶段错过消息的节点将只能通过八卦接收消息，这可能很慢。相比之下，即使树被分解成碎片，GoCast 仍会尝试利用剩余的树链接在树碎片内快速传播消息。

定向八卦[ 11] 旨在解决八卦协议对网络拓扑的遗忘问题。它假设节点连接到拓扑感知图中，并且更喜欢沿着低成本链接传播八卦。未指定如何构建和维护此图。相比之下，GoCast 会自动构建拓扑感知叠加层。

使用消息重传来提高可靠性的多播协议包括 SRM [7] 和 RMTP[10]

Narada [3]建立一个所有组播成员的网状拓扑结构，然后为每个源计算一个组播生成树。

另一方面，NICE[1] 在没有构建网格的情况下显式地形成多播树。

# 结论

我们建议使用 GoCast 进行快速可靠的群组通信。GoCast 构建了一个具有紧密控制节点度的接近感知覆盖网络。多播消息通过嵌入在覆盖网中的高效树快速传播。在后台，节点与它们的覆盖邻居闲聊消息摘要，并拾取由于基于树的多播中断而丢失的消息。我们做出了以下贡献。

* 率先通过在覆盖邻居之间交换八卦来增强基于树的覆盖多播，以实现稳定的吞吐量和多播消息的快速传递。
* 我们最早提出每个节点使用一个随机链接几乎与使用多个随机链接一样稳健。
* 我们的覆盖适配协议是首批产生具有严格控制节点度数的邻近感知覆盖的协议之一。
* 使用大规模真实互联网拓扑和评估延迟数据八卦协议，而不是仅仅根据抽象的八卦会合报告消息延迟。我们通过广泛的模拟系统地比较了几种协议。

GoCast 快速可靠地传送多播消息，并在网络链接上施加低负载。这些特性使GoCast 对许多应用程序具有吸引力。在 IBM Research 的可扩展智能基础设施项目中，我们使用 GoCast 来促进大型系统的管理。

# 参考

1. S.Banerjee,B.Bhattacharjee,andC.Kommareddy.Scalable Application Layer Multicast. In *SIGCOMM*, 2002.
2. K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.
3. Y. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *SIGMETRICS*, 2000.
4. F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *NSDI*, 2004. The network latency data are available at <http://www.pdos.lcs.mit.edu/p2psim/kingdata.>
5. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.
6. P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. From Epidemics to Distributed Computing. To appear in IEEE Computer Magazine, 2004. http://www.irisa.fr/paris/Biblio/Papers/Kermarrec/EugGueKerMas04IEEEComp.pdf.
7. S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang.   
   A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.
8. X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoSAssured Service Composition in Managed Service Overlay Networks. In *ICDCS*, 2003.
9. R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized Rumor Spreading. In *FOCS*, 2000.
10. J. C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. *INFOCOM*, 1996.
11. M.-J. Lin and K. Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *European Dependable Computing Conference*, 1999.
12. R. Melamed and I. Keidar. Araneola: A Scalable Reliable Multicast System for Dynamic Environments. In *the 3rd IEEE International Symposium on Network Computing and Applications (IEEE NCA)*, 2004.
13. T. S. Eugene Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, 2002.
14. O. Ozkasap, Z. Xiao, and K. P. Birman. Scalability of Two Reliable Multicast Protocols. Technical Report TR99-1748, Cornell University, 1999.
15. L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, 2000.
16. K. Shen. Structure management for scalable overlay service construction. In *NSDI*, 2004.