**无密钥签名基础设施：**

**如何构建全球分布式哈希树**

**Ahto Buldas, Andres Kroonmaa, and Risto Laanoja**

**Guardtime AS, Tammsaare tee 60, 11316 Tallinn, Estonia.**

**Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia.**

**摘要** 无密钥签名基础设施（KSI）是一种全球分布式系统，用于提供时间戳和服务器支持的数字签名服务。 创建全球每秒哈希树并发布其根哈希值。 讨论实际实施服务中出现的一些服务质量问题，并提出解决方案，避免单点故障，保证合理，稳定的延迟服务。 Guardtime AS已经运营了KSI基础设施5年。我们总结了KSI基础设施的建立以及服务运营期间的经验教训。

# 1．引言

无密钥签名是传统PKI签名的替代解决方案。 无密钥这个词并不意味着在签名创建过程中不使用加密密钥。 密钥对于认证仍然是必需的，但是签名可以被可靠地验证，而不需要继续保密密钥。 无密钥签名不易受到密钥泄密的影响，从而为数字签名的长期有效性问题提供解决方案。 传统PKI签名可能受到时间戳的保护，但是只要时间戳技术本身是基于PKI的，密钥妥协的问题仍然没有解决。

在无密钥签名系统中，签名者识别和证据完整性保护的功能被分离并发放给适用于这些功能的加密工具。例如，签名者识别仍然可以通过使用非对称加密来完成，但签名的完整性通过使用无密钥密码学技术来保护——也就是所谓的单向无冲突哈希函数，这是不涉及任何密钥的公共标准转换。

无密钥签名在实践中被实现为多签名，即一次签署许多文档。 签名过程涉及以下步骤：

1.哈希：要签名的文档是散列的，散列值用于在接下来的过程中表示这些文档。

2.聚合：创建一个全球临时的每循环哈希树来表示一轮中签名的所有文档。 轮次的持续时间可能会有所不同; 在所述的实现中固定为一秒。

3.发布：每轮聚合树的top3哈希值被收集到永久哈希树（所谓的哈希日历）中，并且该树的顶部哈希值作为信任锚发布。

为了在实践中使用这样的签名，需要一种类似于传统签名解决方案PKI的合适的无密钥签名基础设施（KSI）。 这样的基础设施由具有层次结构的聚合服务器组成，它们在协作中创建每一轮全球哈希树。 第一层聚合服务器——网关，它负责直接从客户端收集请求; 每个聚合服务器从一组较低级别的服务器接收请求，将它们一起散列成一个哈希树，并将该树的顶部哈希值作为请求发送给上级服务器。 然后，服务器等待来自较高层级服务器的响应，并且通过将接收到的响应与来自其自身的哈希树整合成合适的散列链来创建并递送给每个下级服务器的响应。

在本文中，我们讨论可能出现的一些服务质量和可用性问题并阐述解决方案。 该实现避免了单点故障，保证了相当低的并且稳定的业务延迟。 Guardtime AS已经运行了KSI基础设施5年，足以得出关于可用性和可扩展性的一些结论; 在实践阶段学到的实际经验。 本文介绍了KSI基础设施的主要组成部分和操作原理，并简要介绍了使安全风险最小化的设计决策。

# 2 哈希树和哈希日历

**哈希树：**哈希树聚合技术首先由Merkle [6]提出，首先用于Haber等人[5]的数字时间戳。哈希树时间戳使用单向散列函数将文档列表转换为与时间相关联的固定长度摘要。用户将某个文档的哈希值发送到一个服务，并接收到在给定的时间戳数据存在的签名令牌证明，并且通过特定的接入点接收请求。所有收到的请求都聚合成一个大的哈希树;并且树的顶部固定并保持每秒（图1）。签名令牌包含用于重建通过哈希树的路径的数据，该路径从签名的哈希值（叶子节点）开始到顶部哈希值。例如，为了验证x**2**（图1）的令牌y，我们首先将y与x**1**（作为签名令牌的一部分保留）并且计算哈希值y**2** = h（x**1** | y）用作下一个散列步骤的输入，直到我们达到顶点哈希值，即在示例情况下为y**3** = h（y**2** | x**34**）。如果y**3** = xtop，那么可以确信y在原始的哈希树中是存在的。

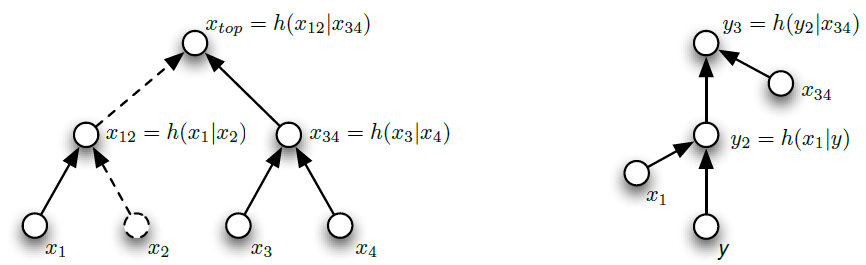


图1. 计算哈希树（左），并在x**2**的位置验证y。

**哈希日历：**每秒钟的顶部哈希值在称为哈希日历的全球唯一的哈希树中链接在一起，以便新叶子节点仅添加到树的一侧。 时间值被编码为日历的形状，其修改对于其他用户将是显而易见的。 日历的顶部哈希值周期性地在广受关注的媒体中定期发布。有一个确定的算法来计算链接哈希树的顶部，从而在每一秒给出不同的顶级哈希值。 此外，还有一种每秒从链接哈希树的形状中提取时间值的算法，为每个发出的令牌提供难以修改的时间值。

**反向回溯安全性：**恶意服务器无法向已发布的哈希树添加新请求。 如果散列函数在普通术语（单向性，碰撞等）中是安全的，并且聚合树的大小有限，则该方案确实是安全的，已经展示在[4,3,2]。

# 3. 服务的架构

## 3.1 设计原则

该系统提供全球范围的时间戳和服务器辅助的数字签名服务; 它是可靠的，并且具有最小的信任要求。 次要目标是易于审计和认证，效率和稳健性。

基础数据结构保证不可能发布假的，可回溯的或其他误导性的签名令牌­­——即使流氓客户端和流氓服务提供商串通一气也是如此。致力于全球唯一和公开的哈希日历使篡改系统，特别是时钟值的行为对于所有用户都非常显而易见。系统安全性不依赖于私钥的长期保密性，因为不可能证明密钥没有被实际泄漏。基本的加密原语可能容易地改变，例如 在算法明显脆弱的情况下。有一种情况下，基础设施必须停止，比如当系统完整性或时钟精度有疑问时，这种情况可能会发生。签名令牌本身只能由第三方使用公共信息和算法进行独立验证; 即使服务提供商停止运营，也必须进行验证。

为了提供高可用性服务，根除单点故障。 对系统可靠性的要求是不同的：全球唯一的核心集群必须由最好的信任授权实践运营，但是服务交付网络可以使用商业虚拟服务器，而不需要对运营环境的要求，如可靠的“挂钟时间”（译者注：wall clock 也叫响应时间（response time）、消逝时间（elapsed time）等。这些术语均表示完成任务所需的总时间，包括了硬盘访问、内存访问、I/O操作和操作系统开销等一切时间。）或持久存储 。 隐私和机密性风险很小，因为基础架构只负责把散列值聚合起来。

## 3.2 哈希树实战

尽管Merkle哈希树的概念比较简单，但是组建一个全球范围的服务器网络来构建这样一个树结构也是比较复杂的。 简单的树形服务架构是不符合需求的，因为每个节点都存在潜在的单点故障。 因此，聚合树必须由冗余的服务器集群来构建。 我们还必须在处理树的唯一顶部节点方面设计有冗余，因此我们面临着潜在的顶部哈希值差异，群集分区等问题。

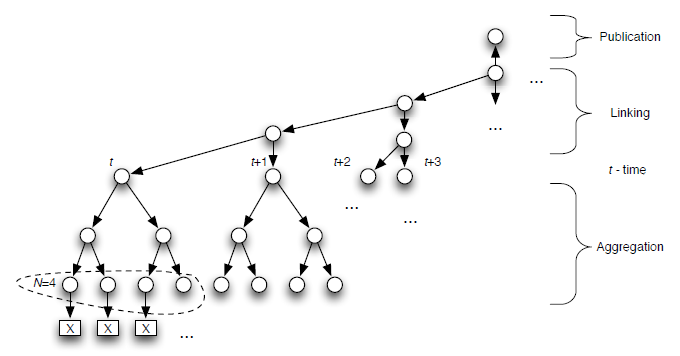


图2. 基于时间戳的二叉哈希树

在每个聚合期间，聚合树的顶部存储在日历数据库中。为了在日历数据库中不可撤销地提交有序的值序列，它们被链接到一个二叉哈希树中，使得叶子节点被添加到一侧，就像时间线上的刻度一样（参见图2中的t） 。该数据库是全球独一无二的，公开开放供所有的第三方审核，归档或验证。聚合由分布式聚合器网络处理。链接，基于日历数据库的内容进行,可以连贯地本地执行。从链接树的任何叶子节点到顶部的路径的形状编码了聚合期间的UTC时间，当此叶子节点产生时。篡改已发布的签名令牌的时间值与重排公共固定顶级Merkle树的节点一样困难。

有确定的算法来计算链接哈希树的顶部，在每个聚合周期给出覆盖所有叶子节点的不同顶层哈希值 。 这个值是定期固定的，并且发布在难以修改和广泛见证的媒体上。 目前，每月使用报纸和流行的微博平台发布一次; 还有一个电子分布式和数字签名的出版物列表用于自动使用。为了适应不断变化的情况，还可以迁移到不同的发布模式。

图2展示了系统在回合（t，t + 1，t + 2，...）中如何运行，每个回合产生一个聚合树。 系统容量（*N*）由最大树大小*N* = 2l确定，其中l表示聚合树的深度。

如第2节所述，树的有限大小（N）对于该方案的可证明的安全性很重要。

# 4. 基础设施概览

高级系统架构如图3所示。文档散列发生在应用程序中，使用提供的软件工具形成签名（或时间戳）请求。该请求被发送到网关——将服务传递给最终用户的系统组件。 网关对其聚合周期中收到的请求进行初始聚合，然后将其聚合请求发送到上游聚合集群。请求通过多层聚合器服务器进行聚合，并且核心集群选择全球唯一的顶部哈希值。响应通过聚合层立即被返回。

每个聚合周期的顶部哈希值都会被收集在“日历数据库”中，并通过“日历缓存”层分发到扩展服务，通常它与网关主机位于同一位置。 客户端应用程序使用扩展服务进行验证。

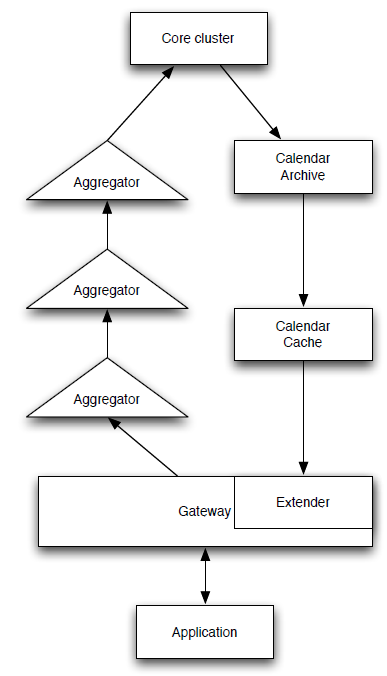


图3. 高级系统架构和聚合网络。

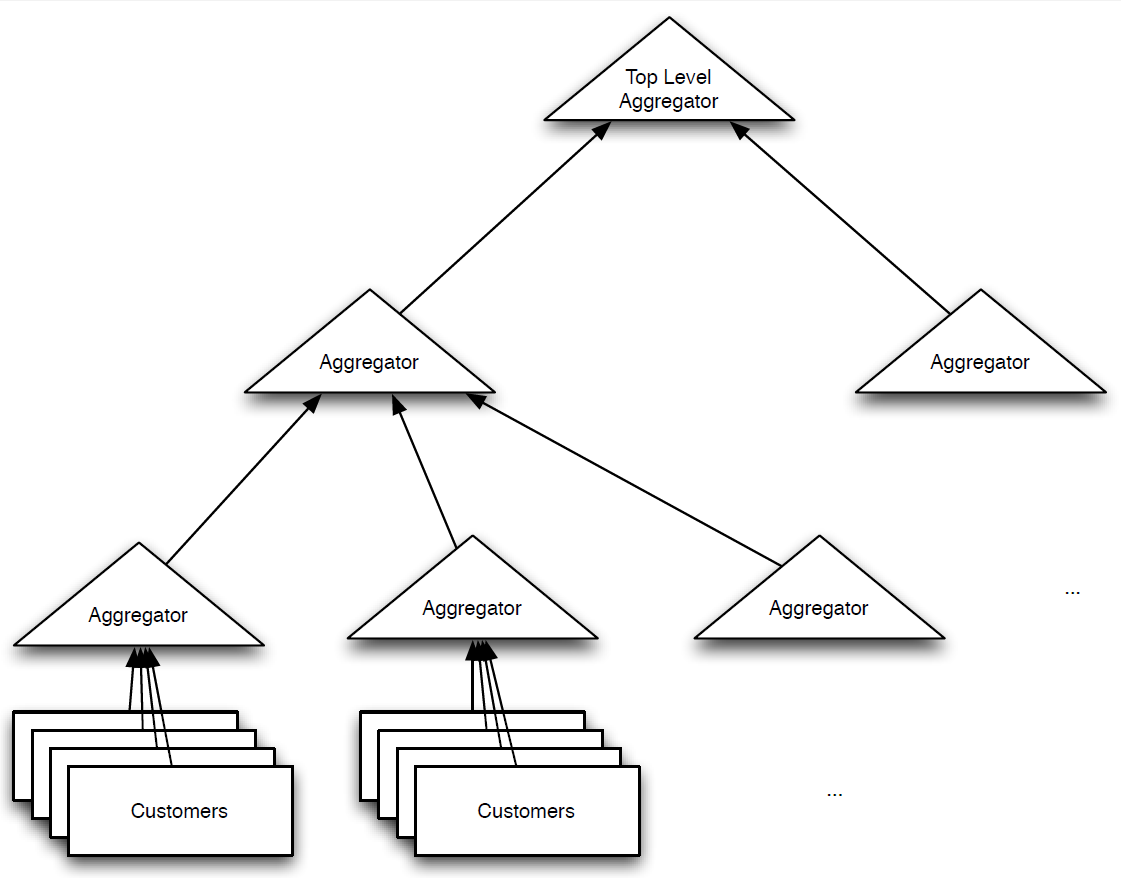


图3. 高级系统架构和聚合网络（为了图片更清晰，原图被分成两部分——译者）。

## 4.1 聚合网络

聚合器是从所有传入请求构建哈希树并将顶部哈希值传递到上游系统组件的系统组件。 聚合器是按照回合工作的,每个回合的时间相同。 在一个回合中收到的请求被聚合到相同的哈希树中。 在从上游组件接收到响应之后，聚合器立即将响应与其自己的树的哈希路径一起发送到所有子聚合器。 由于忽略来自同一轮的上游组件的后续响应，因此可以并行运行多个聚合器以获得更高的可用性。

聚合树水平方向上分为四层，构建基础设施，使顶层与核心集群相邻; 两个中间层提供地理分布。 最底层与网关绑定，并托管在离最终用户最近的地方（类似CDN——译者注）。 每个聚合器通过将其下游客户端的名称进行散列并加入到哈希树中来标记它们。 这些前向安全标签形成了用于标识不同服务端点的分层命名空间。

扩展聚合树比较容易; 为了使系统容量加倍，我们必须向签名令牌添加一个哈希值。 当前的哈希树深度固定为50层，理论最大容量为每秒250≈1015个签名。 我们认为这种初始配置在可预见的将来足以应付可能发生的签名需求。 每个网关和聚合服务器都会生成不依赖于负载的上游网络流量。 这样不仅隔离了客户端，不会泄露有关实际服务使用情况的信息，还可以隔离拒绝服务式攻击。 服务能力是呈线性增长的。

## 4.2核心集群

聚合网络最顶层的是核心集群。 核心是一个分布式同步的机器，负责实现每个聚合时间段的最顶层的值的一致性，可持续存储在日历数据库中并将其返回到聚合网络。 核心保持准确的时钟值，作为每个签名令牌的发布时间呈现给客户。 日历数据库使用“哑”缓存层分发到扩展器服务器进行签名令牌验证。

## 4.3网关

网关用作协议适配器，接受特定应用格式（RFC3161，OpenKSI）的请求，并将其转发到指定的一个或多个聚合器。 在实践中，第一级聚合发生在网关主机 - 在聚合器和网关之间提供低的并且可预测的通信带宽。

网关还托管一个扩展器（也称为验证者）服务——签名验证助手。 使用日历数据库的新副本，扩展器服务返回从签名数据构建完整哈希链到最新发布的哈希值所需的丢失哈希值。 这些丢失的哈希值在签名时尚未知晓。 在扩展器的帮助下创建的哈希链由客户端API验证。 令牌有效性在应用层决定，网关不能被视为受信任方。 客户端应用程序可以通过创建所谓的“扩展”令牌来存储具有用于重新创建散列链的完整信息的令牌。网关还托管一个扩展器（也称为验证者）服务 - 签名验证助手。 使用日历数据库的新副本，扩展器服务返回从签名数据构建完整哈希链到最新发布的哈希值所需的丢失哈希值。 这些丢失的哈希值在签名时尚未知晓。 在扩展器的帮助下创建的哈希链由客户端API验证。 令牌有效性在应用层决定，网关不能被视为可信方。 客户端应用程序可以通过创建所谓的“扩展”令牌来存储具有用于重新创建哈希链的完整信息的令牌。

# 5. 可用性和服务质量

## 5.1聚合网络的可用性

为了提高服务的可用性，必须避免单点故障，我们在系统中的任何地方都使用了冗余。 每个聚合服务器都被换成地理位置分散的并行工作的聚合服务器集群，这样,下层服务器向群集的每个成员发送请求，并使用第一个收到的有效答复。 如果单个聚合服务器的可用性系数假定为0.99（即，大概停机时间约为每年3.5天），则具有两台服务器的集群的可用性约为0.9999，假设停机事件完全独立。 请注意，集群可以不停机的扩展和重新配置。

## 5.2核心的可用性

最上面的聚合步骤也由一组冗余服务器执行。有必要确保全球哈希日历的唯一性，因此在最上层的服务器（称为核心）之间必须存在一些协议协议。必须处理的可能情况包括：（1）某些请求可能不会到达核心的所有服务器，（2）核心服务可能会以不同的顺序接收请求，（3）某些节点可能会断开，关闭，维护等。原则上，所有核心服务器可能会有不同的顶部哈希值。解决问题的方法是，服务器尝试通过使用多方协议将其顶部哈希值传播到所有核心节点，在多方协议中，直到每个可能重叠的回合大多数核心服务器具有相同的顶级哈希值集，具有不同集合的服务器将知道它们不属于“大多数”。只有属于“大多数”的核心服务器才会根据他们所达成一致的集合中的顶部哈希值填充日历数据库中的值，并响应他们收到的请求。只要这些服务器能相互通讯，这个协议就可以工作。服务器会退出核心，因为连接或本地托管问题会自动恢复。

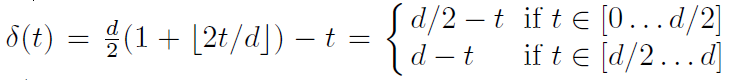
平均协议时间是已连接的核心服务器之间往返时间（可以理解为类似于ping通信延时——译者注）的1.5倍。 这是可以实现的，因为我们不假定恶意安全模型，并且不需要解决完整的“拜占庭协议问题”（“拜占庭协议问题”也被称之为“拜占庭将军问题”，是由分布式系统祖师爷级的人物莱斯利兰伯特（Leslie Lamport）于1998年3月提出的点对点通信中的基本问题。 在分布式计算上，不同的计算机透过讯息交换，尝试达成共识；但有时候，系统上协调计算机（Coordinator / Commander）或成员计算机 （Member / Lieutanent）可能因系统错误并交换错的讯息，导致影响最终的系统一致性。拜占庭将军问题就根据错误计算机的数量，寻找可能的解决办法 ，这无法找到一个绝对的答案，但只可以用来验证一个机制的有效程度。——译者注）。 在全网状连接模式中，使用不同的路由广播每条消息。

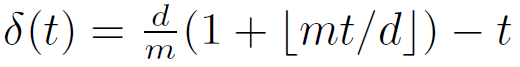
## 5.3服务质量优化

下面我们将介绍如何优化延迟并消除KSI服务响应时间的“长尾效应”。

**简单方法** 聚合网络是冗余的 – 集群有m个聚合器而不是一个聚合器。 每个聚合器具有一定的聚合周期d（以时间为单位）。 大量的聚合周期会产生大的服务延迟，因为在d = 2个时间单位之后，随机时间内收到的请求将被聚合（即，建立的Merkle树，顶部哈希计算并发送给父集群）。 这意味着从客户端到核心集群的路径中的每个聚合器都会添加d = 2个时间单位的服务延迟。

如果聚合回合从0开始并以d结束，则到达t（在[0 ... d]之间）的请求将具有服务延迟d - t，即t越大服务延迟越小。 稍后（在回合关闭之前）到达的请求具有较小的服务延迟。

我们调整聚合器在同一个集群中的轮次，使得请求的平均延迟最小化; 如果我们在集群中有两个聚合器，且每个聚合器均有长度为d的回合（以时间为单位），并且第二聚合器的轮次在时间d/2（而不是0）开始，则（因为每个请求被发送到两个聚合器）， 平均服务延迟为d/4而不是d/2。 这是因为在t处接收的请求的延迟现在是以下函数：

，并且在[0 ... d]中该函数的平均值为d/4。一般来说，如果我们在集群中有m个聚合器，并且集群中第i个聚合器的轮次在时间i /m开始，则，平均延迟为。

一般来说，如果我们在集群中有m个聚合器，并且集群中第i个聚合器的轮次在时间i/m开始，则平均服务延迟将为d/(2m)。

我们通过交织集群中的聚合轮次来减少服务延迟。 简化方法仅在延迟几乎完全随机，且具有与聚合周期相当的标准差的情况下才有用。 这种极端条件在实践中非常罕见。

**实用方法** 子聚合器C和父聚合器P之间的网络延迟由几个组成部分组成：

- 由基本物理学引起的传播延迟，取决于C和P之间的线缆长度。这种延迟不能消除。

- 由网络路由器的全球云引起的序列化延迟，网络路由器选择网络中用于将数据从C发送到P的路径。

- 抖动 主要是由不同的利用率引起的，这会产生处理队列并导致重新传输。

所有这些分量延迟都产生不均匀但是相当尖锐的钟形曲线的概率分布。 如果我们知道95％的请求（C到P）在25-40ms（毫秒）之间有延迟，那么我们可以调整C和P的轮次，以便它们的轮次（如果它们的持续时间d相等） ）分别从t和t + 40ms开始。 这意味着P到C发送的请求的95％的延迟时间不到40ms。 实际上，延迟远小于d/m，其中m是集群中聚合器的数量，d是聚合周期。

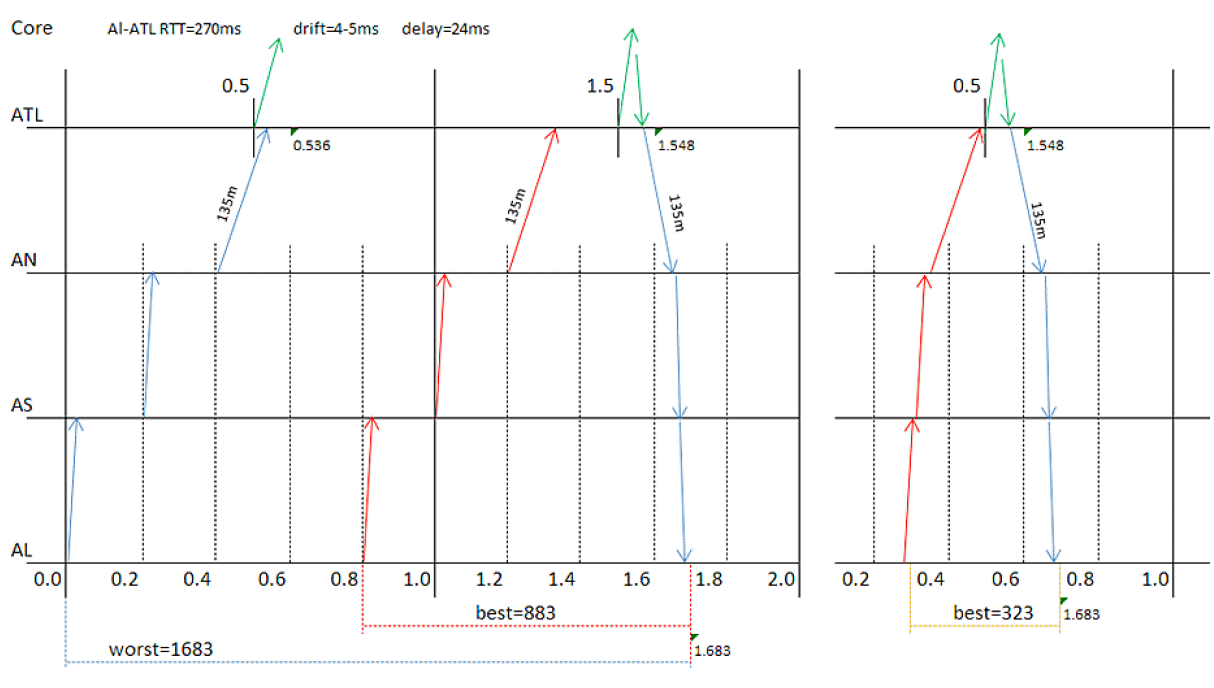


图4. 消息流通过聚合层（AL - > AS - > AN - > ATL - > core and back）。 水平轴表示以秒为单位的时间流。

聚合层之间的消息流如图4所示。 纵轴表示聚合树的层，横轴表示以秒为单位的时间流。 左边手绘图说明了两个不同步的请求。 第一个请求是最坏的情况，第二个是最好的。 当请求上游时，它等待每个层聚合的结束。 第一个请求错过了1秒顶级聚合周期的结束; 第二个请求在结束之前到达，并且两个请求的响应同时到达。 右边手绘图描绘了同步层的理想情况。

# 6 实证结果

## 6.1测试设置

该测试是在向日本扩展服务期间进行的，在与欧洲不同管辖区域之间分布的核心集群的拓扑非常遥远的位置。 该服务在早期的实验室环境中进行了广泛的测试; 其在非理想网络条件下的性能已经反映出来。 在单一大陆运行的服务延迟已被证明是令人满意的。 我们从位于东京和长野的五家服务提供商租用物理和虚拟服务器。

负载是远程生成的，并且在网关主机上进行测量，以便客户端应用程序到网关的连接不会影响测量。 测试运行24小时为一个周期，至少连续三天。 已经周密的考虑到了最坏的情况。

作为次要目标，我们寻找具有独立资源的服务提供商，特别是和ISP对等并测试了不同规模的物理和虚拟服务器群集提供的服务质量。 我们还测量了系统参数（如聚合周期）的效果以及提供的数据，以协助起草服务层面的协议。

## 6.2成果

主要目标是提高服务质量——为最终用户提供签名服务的最小和确定性延迟; 次要目标是找到一个可以提供合理的可用性的具有成本效益的设置。 进展在图5和图6中前后响应时间的直方图中体现。第一个图描述了初始的现实中的签名响应时间分布。 请注意，由于所有聚合层上都有冗余和自动重试机制，所以没有失败的请求。

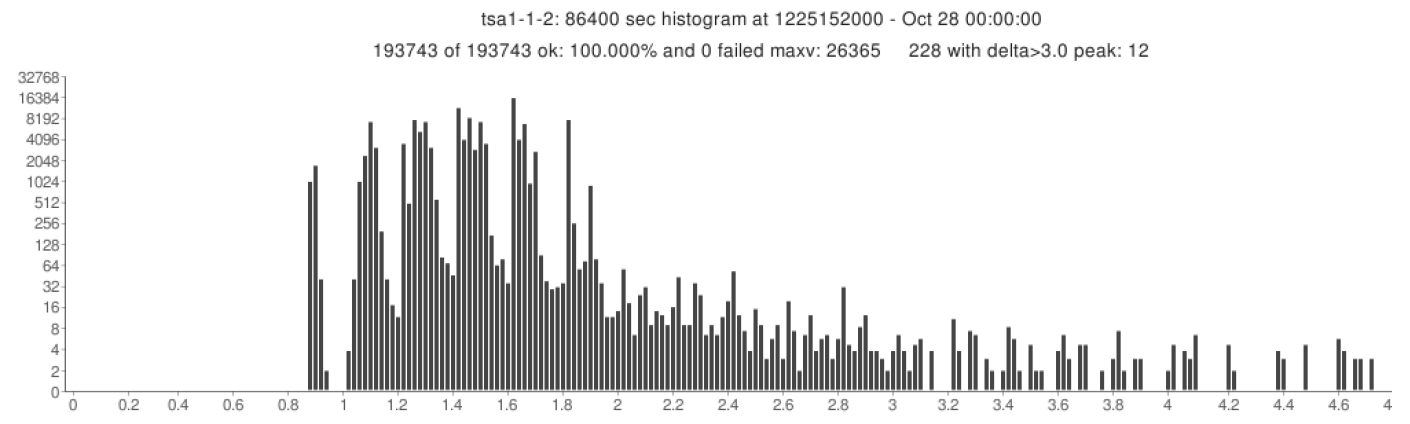


图5.优化之前的真实响应时间直方图。 垂直对数刻度为样本数，水平刻度为秒响应延迟。

图6直方图说明（原文存在笔误，原文为illistrates，但是猜测应该为illustrate，动词，说明、显示的意思——译者注）了聚合层同步和IP网络优化后的结果。 延迟主要由底层网络决定; 从AN到ATL的往返延迟大约为270ms，其他网络延迟要小得多。 所有层的时钟漂移小于4ms，核心协议投票时间为48ms。

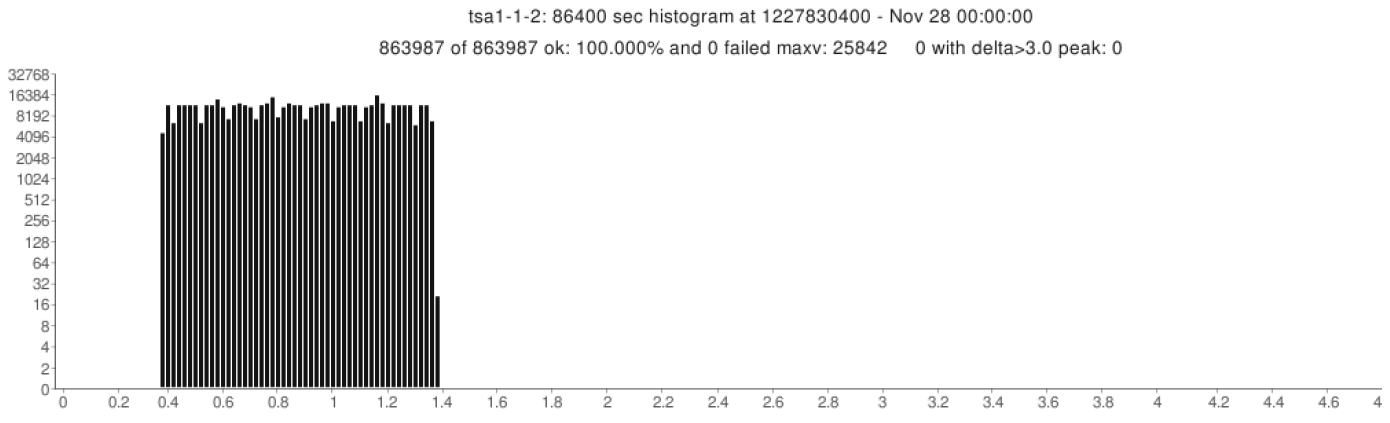


图6.优化之后的真实响应时间直方图。

最终优化和发现包括：

—— 同步聚合层是有益的，因此聚合周期以交错的方式开始。

—— 最低延迟是通过200 ... 400ms的聚合时间来实现的。 我们从200ms开始，之后恢复到400ms以减少数据流量。

—— 在冗余集群中，虚拟服务器是相当不错的。 三台虚拟服务器的成本较低，且可以提供比2台专用物理服务器更好的服务。

—— 虚拟化后的服务器可能会有不稳定的时间流; 它有助于保持本地磁盘IO的最小化。 我们发现有必要设置网络日志记录。

—— 虽然使用起来更简单，但基于TCP的网络协议却有一些不必要的怪癖，特别是“TCP空闲后慢启动”功能。

## 6.3实用意义

我们基于互联网的NTP时间同步建立了相当不错的配置，并根据每个聚合点的聚合层之间的测量RTT（Round-Trip Time: 往返时延——译者注）配置了最佳的时间偏移量; 两个或三个聚合服务器的群集提供了令人满意的结果。 核心集群不需要任何修改，因为它的时钟已经与使用了原子钟和质量时间源的UTC同步。

五年的测量期间系统总体可用性为99.952％; 停机事件发生在运营的头几年，是由于系统管理错误和聚合协议中的历史特征，将实验客户端API代码中的错误传播到核心。 必须从事件中吸取教训，因为发生了似乎不可能发生的事情：（1）不要让系统管理员一天内接触集群的多个成员，（2）网络协议必须具有版本控制机制，以支持部署新的版本，（3）测试必须在完全独立的系统上执行。

# 7结论和未来工作

交错聚合器层同步的思想表明需要一种自动调整聚合网络以达到最佳可能的服务时间的算法。 到目前为止，可能的服务改进没有证明复杂性增加和潜在的不稳定性。

这是对核心集群业务的一瞥。 在后续论文中有很多有趣的话题要讨论。

# 引用

1. Bayer, D., Haber, S., Stornetta, W.-S.: Improving the efficiency and reliability of digital timestamping. In: Sequences II: Methods in Communication, Security, and Computer Sci., pp. 329-334. Springer, Heidelberg (1993)

2. Buldas, A., Laanoja, R.: Security proofs for hash tree time-stamping using hash functions with small output size. In: Boyd, C., Simpson, L. (Eds.): ACISP 2013, LNCS 7959, pp. 235-250, 2013. Springer, Heidelberg (2013)

3. Buldas, A., Niitsoo, M.: Optimally tight security proofs for hash-then-publish time-stamping. In: Steinfeld, R., Hawkes, P. (eds.): ACISP 2010. LNCS 6168, pp. 318-335. Springer, Heidelberg (2010)

4. Buldas, A., Saarepera, M.: On provably secure time-stamping schemes. In: Lee, P.J. (Ed.): ASIACRYPT 2004. LNCS 3329, pp. 500-514. Springer, Heidelberg (2004)

5. Haber, S., Stornetta, W.-S.: How to time-stamp a digital document. Journal of Cryptology 3(2), 99-111 (1991)

6. Merkle, R.C.: Protocols for public-key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy, pp. 122-134 (1980)