基于 Web 网络管理的推送与拉取

Jean-Philippe Martin-Flatin

Email: martin-flatin@epfl.ch

Fax: +41-21-693-6610

Web: http://icawww.epfl.ch

摘要

在本文中，我们展示了如何有效地利用 Web 技术：（i）解决传统 IP 网络管理平台的一些缺陷;（ii）使这些昂贵的平台成为冗余平台。基于 wellens 和 auerbach 提出的嵌入式管理应用的概念，提出了两种基于 Web 技术的网络管理应用设计模型。首先，拉模型基于请求/响应范式。它通常用于执行数据轮询。一些商业管理平台已经使用依赖于此模型的 Web 技术来提供即席管理；我们将演示如何将此扩展到常规管理。其次，推送模型是一种新的方法，它依赖于发布/订阅/分发模式。它比拉模式更适合常规管理，并且允许管理员节省网络带宽和管理站上的 CPU 时间。它可以看作是通知传递常用范例的一种概括。最后，我们介绍了折叠网络管理平台的概念，这两个模型共存。

关键词：基于 Web 的管理、网络管理、推模型、拉模型、嵌入式管理应用程序、折叠的网络管理平台。

1. 介绍

如果我们从软件工程的角度考虑 IP 网络管理应用程序的设计，这是一个相当简单的分布式应用程序的例子。对它没有严格的要求，例如实时约束或容错，甚至有些管理数据可能会丢失。它的复杂性仅源于两点：需要管理的节点数量很大，有时非常多；所有管理数据流量都被视为网络开销，因此应将其保持在最低水平。

同样，如果我们分析当今 IP 网络的典型管理方式（即网络管理平台的设计方式、作为访问协议的 SNMP 的效率以及管理器/代理范式固有的数据轮询原则的效率），很明显大多数网络管理应用程序都无法承受与现代分布式应用程序的比较。为什么不使用面向对象的分析、设计和实现，这是当今业界广泛采用的方法？为什么要受到现有的几个从代理收集数据的 SNMP 协议原语的限制？当这个选择在一段时间内保持不变时，为什么要让管理器重复地告诉每个代理它感兴趣的 MIB 变量的选择会导致网络开销？当数据在代理和管理器之间传输时，为什么不高效地压缩数据呢？当一个接口在主干路由器上断开时，为什么要使用不可靠的传输协议向管理站发送关键警报？为什么跨防火墙管理远程子公司如此困难？为什么管理数据传输经常是不安全的？

鉴于当今广泛使用的技术，IP 网络管理中的许多设计决策显得效率低下或过时。但在 1988-90 年，当第一个 SNMP 框架被设计出来时，他们并没有这样做。此外，如果我们把自己放在一个历史的角度，考虑到市场是如何发展的[13]，我们可以分析和理解当今商业网络管理平台中的许多缺陷。基于 SNMP 的网络管理之所以成功，很大程度上是因为它的简单性，因此事后批评这种简单性是不公平的。尽管如此，IP 网络在实践中的典型管理方式在整个 90 年代发展得很少。如果 IP 网络管理继续发展得如此缓慢，则存在从简单到简单的风险。这可能导致多个供应商提出过多的备选方案，并最终导致开放式集成网络管理。

正如我们在前面的工作[12]中所展示的，与传统的基于 SNMP 的管理相比，有几个替代方案：基于 Web 的管理、移动代理、主动网络、CORBA、智能代理等。在我们看来，Web 技术是短期内改善这种情况的最佳候选方案。原因是五倍。首先，我们在本文中描述的解决方案很简单，可以在不到一年的时间内进行设计和广泛部署；相反，移动代理需要目前没有人能够提供的安全环境（尤其是广域

网链路）；同样，用于 IP 网络管理的简单而高效的多代理系统仍然有待观察。其次，与 CORBA 不同，Web 技术在网络设备上的占用空间有限。第三，正如我们在本文中所展示的，Web 技术不仅为上述问题提供了解决方案，而且还提供了一条平滑的迁移路径，这是行业要采用的一个关键特性。第四，与 WBEM 不同，它们允许为开放式网络管理保持一致的单一框架。第五，也是最后一个，万维网最近在企业界取得了巨大的成功。它的简单性以及 Java 的可移植性使得它无处不在，很难找到任何软件工程领域，它不使用（或迁移使用）它的早期技术之一（Web 浏览器、HTTP、HTML 和 CGI 脚本）或其最新技术之一（Java 应用程序、applet、servlet、RMI 和 JDBC）。网络专业知识正在全球迅速发展，在网络管理中充分利用这一点是有意义的。

在 IP 网络管理中使用 Web 的想法并不新鲜。早期的 Web 技术（即 Web 浏览器、HTTP、HTML 和 CGI 脚本）的实验始于 1993-94 年。最初，他们只局限于次要任务。例如，人们开发 HTML 表单来标准化和自动化问题报告，这有助于 CallDesk 的工作。网络管理员还用安装在内部 Web 服务器上的电子版本替换了每日、每周和每月打印的报告。更有趣的是，管理员开始编写症状驱动的 HTML 表单，操作员可以使用这些表单进行常规的网络故障排除；事实证明，Web 提供的交互界面比操作员使用的大量程序更加用户友好。当网络设备文档以电子格式发送时，它们被放在内部 Web 服务器上；不仅更容易访问，而且管理员可以直接在症状驱动的 HTML 页面中嵌入指向文档相关页面的指针。文档、过程和工具的集成是网络故障排除中的一个进步。

当供应商开始在其网络设备中嵌入 HTTP 服务器时，就迈出了迈向基于 Web 的网络管理的第一步。Bruins[2]报告了 Cisco 在 1995 年所做的一些早期实验，即将整个命令行界面映射到 URL。例如，Web 浏览器可以向路由器请求 <url:http://router ou name/exec/show/interface/ethernet0/>，该路由器会将其视为以交互方式键入了命令行 show interface ethernet0。这为配置管理和症状驱动的 HTML 表单打开了新的大门，因为不再需要 telnet 进入网络设备。Mullaney[16]还描述了由 ftp 软件执行的工作，通过该软件，代理将静态、本地存储或动态生成的 HTML 页面发送回管理站，以响应 HTTP GET 或 POST 请求。

第 1995 个重要步骤是当 Java 小程序出现在 Netscape 著名 Web 浏览器中的时候。据我们所知，这项技术在网络管理领域开拓的新视野首次发表于 1996 年

7 月出版的《简易时报》上，并刊登了广告。Wellens 和 Auerbach[25]的创始文章介绍了嵌入式管理应用程序的概念，并展示了使用 HTTP 而不是 SNMP 在管理者和代理之间传输数据的优势。虽然作者在他们的文章中没有明确提到小程序，但是他们提出的解决方案是将一个附加程序（必须移植到许多不同的管理平台和操作系统）转换为一个可以在任何地方运行的小程序。此小程序存储在托管设备中，由管理员通过 Web 浏览器上载。小程序与其源代理之间的通信稍后依赖于 HTTP 而不是 SNMP。Bruins[2]在 Cisco 对原型工作的描述中明确提到了小程序；但是在他描述的场景中，一旦小程序被上载，与代理的后续通信依赖于 SNMP，而不是 HTTP，这是我们将在第 3.2.2 节中展示的小程序使用不当。

wellens 和 auerbach 基于 applet 的方法已经被许多网络设备供应商采用，他们在设备中嵌入了 HTTP 服务器和管理 applet，同时也被一些网络管理平台供应商采用，这些供应商支持 Web 浏览器作为其网络管理平台的前端。

自从这项提议提出以来，许多新技术已经出现在网络上。今天，除了 applet 和 Java 应用程序之外，我们还可以使用 servlet、RMI 等。所有这些技术都打开了新的可能性，并启用了网络管理应用程序的新设计。利用这些新技术，我们建议将韦伦斯和奥尔巴赫的想法再推进两步。首先，我们证明了他们提出的设计范式只是一个更通用的范式拉模型的一个实例，拉模型不仅可以像他们那样应用于特殊管理，而且可以应用于常规管理。其次，我们介绍了一种新的设计，称为推模型。与拉模型不同，它不是基于请求/响应范式，而是基于发布/订阅/分发范式。使用此方案，管理数据传输始终由代理启动，如 Web 前网络管理中的 SNMP 通知传递。推送模型减少了网络开销，并将部分 CPU 负担从管理器转移到代理。

本文的其余部分组织如下。在第 2 节中，我们总结了传统的基于 SNMP 的网络管理的主要缺点，并概述了 Web 技术如何解决这些缺点。在第 3 和第 4 节中，介绍了拉模型和推模型的工程细节，分析了 HTTP、套接字和 RMI 三种通信技术的优缺点。最后，我们在第 5 节中介绍了崩溃网络管理平台的概念，并对今后的工作作了一些展望。

2. 传统的基于 SNMP 的网络管理存在的问题

本节概述了传统的基于SNMP的网络管理（即Web时代之前的IP网络管理）中遇到的问题，并描述了 Web 技术如何解决这些问题。这些问题可以分为四类：网络管理平台、协议效率、安全性和传输协议。本文中使用的术语以及我们的分析所基于的网络管理平台模型都在[13]中详细介绍。

2.1.网络管理平台

在最近的一篇论文[13]中，我们介绍了网络时代之前的 IP 网络管理的简要历史，展示了人们如何使用特定于供应商的管理 GUI（当它们集成到网络管理平台时称为附加组件）。本文还详细阐述了网络化前 IP 网络管理的不足之处。总而言之，客户有四个不满：（i）网络管理平台在硬件和软件方面过于昂贵；不需要专门的硬件来管理网络；（ii）应该无限支持第三方 RDBMS；如今，客户受到已签署的对等协议的限制，或者没有在 RDBMS 供应商和网络管理平台供应商之间签署协议；如果他们希望后者支持他们已经拥有的另一个 RDBMS，他们将为“端口”收取巨额费用；（iii）为了网络管理的唯一目的，一些客户必须支持 Unix 系统，尽管他们运行一个业务实体 Ely 基于 PC 和/或 Mac；他们希望使用 PC 或 Mac，但不希望购买全新（昂贵）的网络管理平台。

基于 Web 的网络管理对委屈的回答是（一）网络管理平台的崩溃，本文将逐步介绍。虽然 JDBC 解决了 Java 解释字节码的执行速度差（即使使用加速技术，如 JIT 编译器），但 JDBC 解决了（Survivices）（II）。通过 Java 的平台无关性和 Web 浏览器提供的通用接口可以解决抱怨（III）。

另一方面，网络设备供应商主要对其设备支持特定于设备的管理 GUI 所需的巨大成本感到不满。对于客户来说，无论底层使用的是什么管理平台，给定的 GUI 看起来或多或少都是相同的。但对于网络设备供应商来说，则不然。当发布新的管理 GUI 时，必须将代码移植到许多不同的操作系统（Windows 95、Windows 98、Windows NT 4.x、Solaris 2.x、HP-UX 10.x、HP-UX 11.x…）和许多支持不同 API 的不同管理平台（HP OpenView、Cabletron Spectrum、Sun Solstice、IBM NetView…）。随着时间的推移，尽管主要管理平台供应商的数量相对较少且稳定，但每个供应商支持的设备数量以及要移植的操作系统数量都增长得如此之大，以至于这些管理 GUI 的维护成本猛增。

通过 Web 技术，这个问题通过 applet 来解决，正如我们在第 3 节中看到的：同一个插件的多个化身都用一个代码替换，管理小程序，用 Java 编写。

客户和网络设备供应商还有两个共同的问题。首先，他们都希望缩短管理 GUI 的上市时间。当他们购买一台全新的设备时，客户希望能够通过他们最喜欢的管理平台立即对其进行管理。但是，从市场宣传的新网络设备最终发布并销售给客户，到其特定于供应商的管理图形用户界面移植到所有操作系统和所有现有网络管理平台的时间，可能要经过几个月。在许多环境中，网络的持续可用性对于业务的顺利运行至关重要，除非能够对其进行管理，否则无法购买网络设备。因此，对于与所有主要管理平台供应商签订点对点协议的大型公司，存在一个时间窗口，在此期间他们不能向这些客户销售；这对客户和供应商都是一个问题。对于小公司，尤其是那些专门从事尖端技术的初创公司来说，这个问题更为严重。由于它们的市场份额接近于零，管理平台供应商对它们没有兴趣，因为他们不需要与它们签订对等协议。因此，想要从小公司购买的客户被减少到使用不友好的 mib 浏览器或在设备旁边的专用 PC 上运行定制软件来管理他们的网络设备。因此，许多市场对这样的初创企业关闭，这些企业迫切希望获得综合网络管理。

第二个问题涉及客户和供应商，即版本控制[16]。当升级特定于供应商的 MIB，从而升级特定于供应商的管理 GUI 时，客户和网络设备供应商希望处理集成到管理平台的附加组件与代理支持的 MIB 版本级别不同的情况。今天，由于没有 MIB 发现协议这样的东西，管理员要么必须手动指定哪个设备支持什么 MIB，这很繁琐，要么他们必须避免自己使用在上一个和上一个 MIB 版本之间发生更改的 MIB 变量，这可能会导致问题。

在基于 Web 的管理中，小程序再次解决了最后两个问题。购买时，基于 applet 的管理软件会嵌入到网络设备中，因此您可以立即管理您的代理。升级代理上的软件时，也可以轻松升级管理小程序。通过将管理软件从代理转移到经理，我们可以确保特定于供应商的 MIB 的版本在双方都是相同的。

2.2.协议效率

从一开始，基于 SNMP 的网络管理就受到两个协议工程决策的阻碍，这两个决策大大降低了其效率。首先，对于 snmpv1 框架，smiv1[20]和 snmpv2 和 snmpv3 框架，都必须对 smi mib 数据使用 ber 编码[10]。不幸的是，这种编码以其效率低下而闻名。Mitra[15]和 Neufeld 和 Vuong[17]对此问题进行了详细描述，并表明

与实际数据（内容）相比，传输的管理数据量（标识符和长度）非常大。由于 ASN.1 本身并不要求使用任何特定的编码规则，因此定义了其他更有效的方案，如[11]。但是，它们没有通过 SNMP 框架。第二个问题是 SNMP 本身。snmp varbind 列表比较昂贵，因为用于命名变量的 OID 通常比值占用更多的空间。此外，缺乏有效的表检索机制意味着总协议效率受到重复的消息交换（以及代理端的重复计算）的影响。

在基于 Web 的网络管理中，通过使用 HTTP 1.1 而不是 SNMP 在管理器和代理之间传输 SMI MIB 数据来解决这些问题。其优点是四倍。首先，这种迁移使得放弃 BER 编码成为可能，并使用一种新的用于 SNMP 的 mime 内容类型，或者简单地将 smi-mib 数据编码为 html[16]。第二，使用持久连接[6]，这是 HTTP 1.1 的一个关键特性，可以减少由多个 TCP 连接设置和拆卸引起的网络开销和延迟。第三，pipelining[6]是 HTTP1.1 的另一个关键特性，它允许管理器在不等待每个响应的情况下发出多个请求。这减少了延迟，但也允许非常有效地使用 TCP 连接，当与持久连接结合时：如果每个持久连接的超时值大于该代理的轮询频率，则可以在管理器和每个代理之间无限期地使用相同的 TCP 连接。第四，通过执行透明的数据压缩，可以减少网络带宽的使用。与 SNMP 不同，HTTP 支持内容类型和内容传输编码的 mime 概念来传输数据。因此，可以在代理上压缩 HTTP 数据包（例如 gzip）的有效负载，并在管理器上解压缩，而管理应用程序甚至不知道数据在传输时被压缩。因为有效负载是纯文本的，所以预期的压缩率相当高。

HTTP 唯一没有解决的问题是缺乏有效的表检索机制。这可以通过向上面提到的新 mime 内容类型添加新的原语来处理。RMI 和对象序列化提供了一个更整洁的解决方案，因为它们用直接的对象到对象通信替换了像 SNMP 或 HTTP 这样的通信协议。snmp varbind 列表替换为序列化对象，缺少有效的表检索机制只会影响代理，我们将进一步介绍。但是 RMI 也有问题，我们将在第 4.2.2 节中看到。

2.3.安全性

安全性是 snmpv1 和 snmpv2 框架的弱点[22]。缺乏安全的 snmp get 和 set 阻碍了远程子公司多年的管理。使用 SNMP，企业如何能够合理地管理跨越 Internet 或某种公共网络的 VPN？最近发布的 snmpv3 框架这方面的情况有了很大改善。但现场测试仍然显示，管理员对部署中的新安全框架很满意。如今，要求通过公

共广域网链路（如银行）进行安全通信的组织通常使用昂贵的设备，这些设备在广域网链路边界的低协议层执行透明的加密和解密。无论数据是 SNMP 还是其他，它总是加密的。

由于 ssl[23]，在安全性方面，HTTP 已经被证明优于 snmp 好几年了。SSL 是一种安全协议，位于传输层协议（TCP）和应用层协议（HTTP）之间，防止窃听、篡改和消息伪造。当一个 URL 以（即，具有一个 RFC-1738 方案名等于）https 开始时，HTTP 客户端连接到服务器上的端口 443/TCP[9]，而不是标准的 HTTP 端口 80/TCP[9]；然后通过 SSL 发送 HTTP 流量。和网络管理一样，IETF 和 W3C 在网络安全方面也做了很多工作。这包括 ssl 的继承者，tls[5]，在这上面可以对 http 进行分层[18]，还包括对强安全电子事务的支持，以及在 http 中集成强身份验证和隐私方案。

2.4.传输协议：TCP 与 UDP

使用 HTTP 而不是 SNMP 的想法非常直观：它们都是请求/响应协议，使用客户机/服务器技术实现，并且一个 snmp get 或 set 可以很好地映射到一个 http get 或 post。但是，这些通信协议依赖于不同的传输协议这一事实立即引发了一个问题：管理数据应该通过可靠的协议（如 TCP）或不可靠的协议（如 UDP）传输吗？这个问题已经辩论多年了。人们经常对此有强烈的意见。作为 SNMP 的设计者之一，Rose 强烈反对使用可靠的传输协议[19]。相反，韦伦斯和奥尔巴赫则谴责他们所称的“崩溃的脊梁”的神话[25]。他们的分析考虑到了这样一个事实，即在 1996 年，HTTP 1.0 由于缺少持久连接而受到影响，因此对每个 SNMP 使用 TCP 连接会带来很大的开销和延迟。如我们所知，HTTP1.1 解决了这一问题，这使得今天他们的论点更加有力。

我们认为网络管理员应该有选择。不仅应该在企业级自定义传输协议的选择，而且还应该在网络设备级，甚至可能在 MIB 变量级。不幸的是，今天没有这样的选择：每个人都在 UDP 上使用 SNMP，尽管没有什么可以阻止在 TCP 上使用 SNMP。根据我们的专业经验，我们可以声明有些设置中管理数据比用户数据更重要。在这些情况下，由于管理数据流量和用户流量在 IP 世界中共享相同的媒介，因此管理数据要么被分配到链路容量的某个（低）比例（一些设备供应商已经提供的特性），要么在路由器中被赋予更高的优先级（这一特性在今天几乎没有使用过，但可能会变得很普遍）。在不久的将来，随着实时多媒体流量的增加）。

显然，管理者的职责是确保管理数据的比例与链路容量相比仍然较低，因为网络的原因是承载用户数据，而不是管理数据！与广域网链接不同，大多数内部网都是超大规模的，而且损失很少是由于持续的网络饱和造成的。它们更像是由于普通的原因，如以太网冲突，或者在流量激增的情况下路由器中出现临时缓冲区溢出。因此，在我们看来，在这种网络中，管理数据应该以可靠的协议进行传输。相反，对于具有令人印象深刻的 MTBFS 的主干路由器，或对于昂贵的洲际广域网链路，管理数据通常远不如用户流量重要；在这些情况下，管理数据应通过不可靠的传输协议（如 UDP）进行传输。

上面，当我们说“管理数据”时，我们指的是数据收集和网络监控。SNMP 通知是完全不同的情况。只有在出现重大问题时才发送。因为什么是主要问题的定义是特定于站点的，所以大多数（如果不是所有的）网络设备允许管理员过滤哪些通知应该或不应该发送到 NMS。不幸的是，在当前的 SNMP 技术中，这些通知也通过不可靠的传输协议发送。在我们看来，他们应该被给予最高的优先权。如果一个接口在主干路由器中发生故障，路由器立即将此事件报告给 NMS 比在备用路径上实际路由用户流量更重要。使用 udp 而不是 tcp 意味着这个重要的数据包可能会因为一个愚蠢的原因而丢失，比如 NMS 本地网段上的以太网冲突。虽然 TCP 不能保证包的传递，但至少它支持自动重传和确认。UDP 不这样做，并将重新传输丢失数据报的负担置于管理应用程序上，使其变得不必要的复杂。

总之，数据收集和网络监控可能使用可靠或不可靠的传输协议，这取决于特定站点的需要。对于大量使用的主干路由器来说，基于 TCP 的网络管理可能是一个坏主意，因为它们都是平放使用的，需要 24x24 和 7x7 工作，而且很少遇到任何问题。但它似乎非常适合内部网，尤其是中小企业的网络。至于通知传递，TCP 可能比 UDP 更好，因为主要问题确实应该通过可靠的传输协议报告给 NMS，以最大限度地提高通知到达目的地的机会。

选择 UDP 或 TCP 作为传输协议的一个重要副作用是防火墙问题。越来越多的组织通过设置防火墙来保护他们的内部网络免受互联网入侵者的入侵。在商业世界，防火墙有望在不久的将来变得无处不在。大多数防火墙的设置都是为了让 HTTP 流量通过，因为 Web，并且过滤掉 UDP 流量，因为它可能是已知攻击的载体[4]。对于需要通过广域网链接管理远程办公室的公司来说，这是一个问题。一些防火墙系统支持动态了解 UDP 流量的 UDP 中继，并尝试确定测量的模式是

攻击模式还是正常使用模式。这种中继需要对防火墙进行特殊配置。安全或网络管理员是否愿意开放（甚至部分）对内部 NMS 的外部访问主要是安全策略问题，因此是特定于站点的。但许多人可能不想这样做，尤其是那些拥有远程办公室的中小企业，他们在防火墙方面没有内部专业知识。对于这些公司来说，当他们决定在防火墙后面保护自己时，使用 TCP 作为传输协议可能是最好的选择。

3. 拉力模型

在第 3.1 节中，我们首先解释了拉和推模型是如何工作的。在第 3.2 节中，我们随后介绍了基于拉的即席管理的工程细节，并展示了 Web 技术如何补充 NMS 提供的管理功能。在这种情况下，任何运行 Web 浏览器的机器都可以进行网络故障排除，而 NMS 仍然负责常规管理。最后，在第 3.3 节中，我们展示了 Web 技术还可以处理常规管理，并使网络管理平台成为故障排除和数据收集的冗余平台。

3.1.拉与推：报纸的比喻

在软件工程中，拉模型和推模型指定了两种众所周知的方法来在两个远程实体之间交换数据。报纸的比喻就是这些模式的简单说明：如果你想每天读你最喜欢的报纸，你可以每天早上去买，或者订阅一次，然后在家里自动接收。前者是拉的例子，后者是推的例子。pull 模型基于请求/响应范式（在传统的基于 SNMP 的网络管理中称为数据轮询或简单轮询）；客户机向服务器发送请求，然后服务器以同步或异步方式响应。这在功能上相当于客户机从服务器上提取数据。在这种方法中，数据传输总是由客户机（即经理）发起的。相反，推送模型基于发布/订阅/分发范式。在此模型中，代理首先公布他们支持的 MIB 以及可以发送的 SNMP 通知；然后，管理员将管理器（NMS）订阅他/她感兴趣的数据，指定管理器接收此数据的频率，并断开连接。稍后，每个代理分别主动通过调度程序（例如，用于网络监控）或异步（例如，发送 SNMP 通知）向管理器推送数据。

3.2.特别管理

applet 技术最简单、最直观的应用是即席管理。在本节中，我们总结了特殊管理和常规管理的特点，然后回到 Wellens 和 Auerbach 的模型并研究了其工程细节；我们展示了通用的 gui 可以类似地被编码为小程序，最后给出一个图说明了特殊管理如何完全依赖于 Web 技术。

3.2.1. 特设管理与常规管理

当他们描述嵌入式管理应用程序方法时，Wellens 和 Auerbach 考虑到了特殊的管理。临时管理始终是手动的，并且需要用户（管理员或操作员）通过一些 GUI 与管理软件交互。它是典型的临时任务：连接到网络设备，检索一些数据以检查

某些内容，然后在不久后断开连接。相反，常规管理涉及持续的数据收集、网络监控和事件处理。它在很大程度上是自动化的，并且通常连续运行。

实际上，所有公司都进行了特别管理。在能够负担专门负责监控网络（运营商）或完全依赖自动常规管理的大型组织中，临时管理是常规管理的补充。例如，管理员可能偶尔希望弹出一个 GUI 来配置路由器，或者操作员可能在调查网络问题时查看一系列错误率。相反，在中小企业等小型组织中，临时管理通常取代常规管理。没有运营商，也没有专用的网管：管理软件只是偶尔使用，临时使用。自组织网络管理通常包括故障排除（即刚出现的网络问题，管理员尝试手动识别和修复问题）或配置管理（例如，管理员设置新路由器，或检查路由器是否按预期配置）。

3.2.2. 供应商特定的管理 gui 编码为小程序（嵌入式管理应用程序）

正如我们在第 2 节中看到的，小程序解决了基于 SNMP 的网络管理中的许多问题。它们降低了供应商管理 GUI 的开发成本；它们解决了在同一网络中拥有不同版本的供应商特定 MIB 的问题；它们将管理 GUI 的上市时间缩短到零；它们独立于运行 Web 浏览器的机器：这可以是运行 Windows 或 Linux 的 PC、Mac、Unix W 工作站等

网络浏览器上传小程序如图 1 所示。但是我们如何为给定的设备指定一个小程序呢？与传统的网络管理平台一样，用户（管理员或操作员）的入口点是从内部 Web 浏览器检索到的网络地图。这个映射可以是一个定制的小程序，或者更简单地说是一个用作 HTML 敏感映射的 GIF 图像：当用户单击它以选择网络设备的图标时，（x，y）坐标通过 CGI 脚本映射到代理，相应的 URL 是从所选代理请求的。在代理上运行的 HTTP 服务器从本地存储（例如从 EPROM）中检索其特定于供应商的管理小程序，并将其发送回 Web 浏览器。一旦小程序被 Web 浏览器上载，有两种方法可以继续：使用 SNMP 或 HTTP。

如果我们使用 SNMP，管理器（在任何机器上运行的 Web 浏览器）和代理（网络设备）之间的交互如图 1 所示。步骤 1 和 2 发生一次，以传输小程序，而步骤 3 和 4 是一个迭代过程。步骤 2 的虚线箭头是一个视觉辅助工具，显示小程序从代理传输到管理器（而不是两个实体之间简单地通信）。实际上，这种传输发生在 Web 浏览器的 HTTP 客户端和代理的 HTTP 服务器之间。一旦小程序被上传，用户就可以在 NMS 中拥有与之交互的插件。小程序将图形交互转换为

SNMP 命令（例如，在重置按钮的绘图上单击鼠标可以映射到一个 SNMP 集）。换句话说，我们有一个 Java API，在下面做 SNMP 调用。

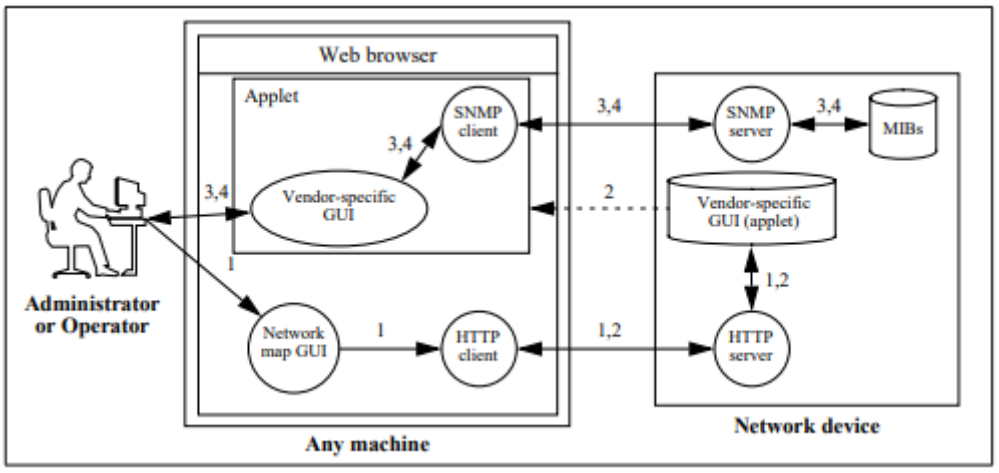

图 1. 拉模型：HTTP 和 SNMP

但是我们如何在浏览器中加载这个 SNMP 堆栈呢？最简单的方法是在 applet 中包含一个完整的 snmp 堆栈，正如 Bruins[2]所描述的，因为 applet 安全模型阻止它从本地文件系统检索这个堆栈。因此，每次从网络设备上载管理小程序时，都需要移动整个 SNMP 堆栈。这显然是低效的，尤其是当这种传输发生在广域网链路上时。这方面的一个改进是通过套接字分别检索 SNMP 堆栈。由于小程序的安全模型，套接字只能在小程序与其源服务器之间打开；因此我们需要一个代理来充当源服务器。管理小程序首先由 Web 浏览器向代理请求；其次，代理联系网络设备并检索不带 SNMP 堆栈的小程序；第三，将小程序传递给执行它的 Web 浏览器；第四，小程序向代理打开一个套接字，代理必须运行套接字的服务器端（即像 unix 守护进程一样的应用程序；第五，snmp 堆栈通过套接字传输，套接字关闭；之后，管理器和代理之间的所有 SNMP 通信都由代理转发。

不过，这种方法的兴趣有限，因为在 Web 浏览器和代理之间不使用 HTTP，我们无法从第 2 节中介绍的 HTTP 优于 SNMP 的优势中获益：通过在特定的 mime 类型中编码 smi mib 数据（mib 变量），或将其嵌入到 HTML 结构化文档中，从而提高协议效率；改进安全性 ty，通过使用 ssl 等，现在让我们来研究一下循环步骤 3 和 4 基于 http 而不是 snmp 的情况。图 2 显示了该解决方案的工程细节。这一次，我们不再在管理器和代理之间使用 snmp；因此，在 Web 浏览器中不需要 snmp 客户机，因此不需要代理。

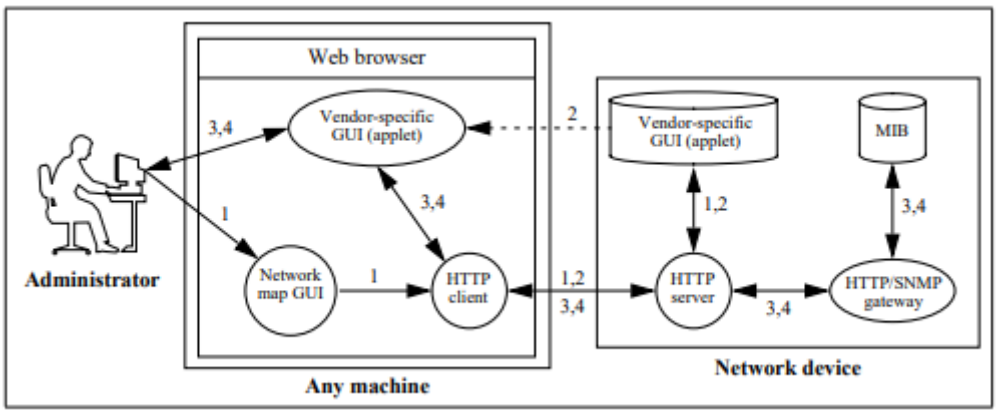

图 2. 拉模型：http 而不是 snmp

在图 2 中，网络设备中的插 ib 图标表示网络设备的特定于供应商的 MIB。此图标表示为 MIB 存储在本地存储上，就像管理小程序一样，因为 MIB 是一个虚拟数据存储库；实际上，当然，它不存储在本地存储上，因为这样效率会非常低，但是作为内存中的数据结构。

当小程序请求 MIB 变量时，将向代理运行的 HTTP 服务器发出请求。然后，此 HTTP 服务器启动 HTTP 到 SNMP 网关以访问本地 MIB。根据代理运行的代码的优化程度，此网关可以直接访问内存中的 MIB 数据结构，也可以执行显式的 snmp get 或 set。这为网络设备供应商提供了一条有用的迁移路径。

供应商特定的管理小程序本质上是商业化的。就像您有可能购买附加组件并将其集成到 Web 前的网络管理平台中一样，您也可以在购买网络设备时购买管理小程序。根据您的网络设备供应商的定价政策，如果您已经拥有多个相同的设备，您可能会损失，因为所有小程序（彼此独立销售）的总成本超过了附加程序（由所有设备共享）的价格。如果你有一个非常异构的网络，你也会得到很多。因为 applet 是商业软件，所以它需要某种许可证执行技术。而不是每次下载小程序时都检查有效的许可证，这将需要在所有网络设备中使用额外的软件，而且还需要管理 HTML 敏感映射调用的 CGI 脚本中的许可证密钥，供应商可能更容易将 HTTP 服务器作为其设备中的额外芯片销售，并使其小程序自由使用。在他们的网站上。许可证强制执行将不再由基于密钥交换的软件执行，而是由硬件执行。

3.2.3. 通用管理 GUI 编码为小程序

到目前为止，我们遵循了 Wellens 和 Auerbach 的原始思想：只有特定于供应商的管理 GUI 被编码为 applet。但是将通用的 gui 编码为 applet 也是非常合理的。

在这种情况下，即席管理完全依赖于 Web 技术：基于 SNMP 的网络管理平台只需要满足常规管理。

与特定于供应商的管理小程序不同，通用管理小程序可以免费提供，也可以出售。对于广泛使用的 MIB，如 MIB-II 或 RMON，人们可以非常合理地期望高质量的免费软件可以从 Internet 检索。Adventnet 的 SNMP 包[1]和 Sun 的 JMAPI[24]是 MIB-II 的两个示例。其他更神秘的 MIB（如 UPS MIB）可能需要网络管理软件供应商销售的软件，因为它们可能不会引起互联网免费软件开发社区的兴趣。

图 3 描述了该解决方案的技术方面。Web 服务器可以是 Internet 或 Intranet 上的任何计算机。网络设备中的插 ibs 图标表示代理支持的所有通用 MIB，以及其特定于供应商的 MIB。在这个图中，我们添加了一条虚线，以表示管理器和代理之间可能存在防火墙。我们将继续讨论防火墙，但让我们指出，对于即席管理，我们只需要让 HTTP 流量通过防火墙。

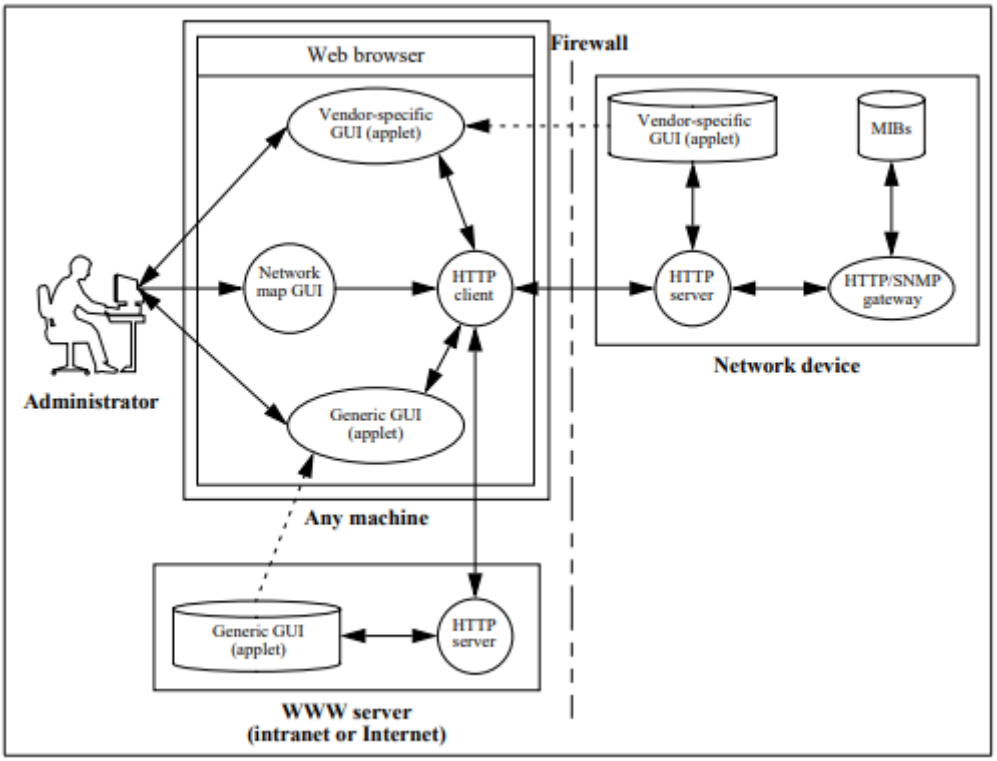

图 3. 拉模型：基于 HTTP 的即席管理

在本节中，我们展示了仅依赖网络设备临时管理的中小企业可以通过使用 Web 技术来节省网络管理平台的成本。因此，根据网络设备供应商的定价政策，从基于 SNMP 的网络管理转向基于 Web 的网络管理可以完全重塑该市场的中小企业细分市场。使用基于 SNMP 的即席管理，客户需要为管理平台、每种设备附

加组件和每种设备对 SNMP 的支持付费。使用基于 Web 的即席管理，它们只需要为 HTTP 的每个设备支持付费。如果网络设备供应商正确定义了后者的价格，他们可以在增加收入的同时减少客户的账单。因此，如果网络设备供应商设法使嵌入式 HTTP 服务器的价格保持在足够低的水平，他们就可以消除网络管理平台市场的底层，通过降低网络管理费用来满足客户的需求。在这种情况下，中小企业很难抵制基于 Web 的网络管理。

3.3.定期管理

正如我们在第 3.2.1 节中所解释的，许多组织也需要定期管理。对于他们来说，网络管理在很大程度上需要自动化，包括数据轮询和事件处理，而我们目前提供的解决方案并没有处理这些问题。让我们先集中讨论数据轮询。首先，让我们假设

有两个独立的管理平台，并排：一个用于基于 SNMP 的常规管理，另一个用于基于 Web 技术的临时管理。我们将逐步描述如何整合它们。

第一步是在网络管理平台中集成一个 Web 浏览器，以便为临时和常规管理提供统一的界面。网络地图图形用户界面加载在 Web 浏览器中，并作为与管理员或操作员进行所有后续交互的入口点（如临时管理）。但对于即席管理，网络地图 GUI 可以是静态图像，例如 HTML 敏感地图。这次，我们希望事件相关器更新网络图，使图标变红、变绿等。为了能够动态地更改其 GUI，网络图必须编码为小程序。在图 4 中，我们展示了这个小程序是从 WWW 服务器加载的，它存储在管理软件存储库中。更新此映射最简单的方法是在 Applet 与事件相关器之间打开一个套接字，该事件相关器是 Java 应用程序中的对象。但是，由于我们不一定想要一个网络地图 GUI，我们在网络地图 GUI 和事件相关器之间添加了一个中介，即网络地图注册表。因此，我们可以独立注册零个、一个或多个网络地图 GUI。[12]

Applet 安全模型要求 Applet 从 Java 应用程序运行的同一台机器上下载。所以图 4 底部的 WWW 服务器和右上角的 WWW 服务器实际上是同一台机器。它们分别表示以使图 4 更具可读性。

第二步是对网络管理平台提供的所有 gui 使用小程序（有关这些 gui 在传统网络管理平台中扮演的角色的定义，请参见[13]）。这些是特定于供应商的管理小程序、通用管理小程序、轮询定义和调度程序小程序以及报表定义和调度程序

小程序。与以前一样，特定于供应商的小程序是从网络设备加载的。所有其他的 gui 彼此独立，因此可以在单独的小程序中分离，如图 4 所示。为了使这个数字更具可读性，我们假设后一个小程序是从同一个 WWW 服务器加载的，这不一定是这样的。

第三步是使数据存储库独立于网络管理平台。在本文中，我们假设数据存储在第三方 RDBMS 中；但是我们可以使用另一种技术，例如纯文本文件系统或面向对象的数据库。数据存储库存储在我们称之为数据服务器的机器上。这可以是任何机器，不一定是 WWW 服务器，也可以是运行 Web 浏览器的机器。为了存储或检索数据，我们建议使用 JDBC；其他技术也是可能的，它们实现了 applet 和数据存储库之间的某种粘合。为了使图 4 更容易阅读，我们假设我们有一个用于轮询、报告定义和调度的数据存储库。当然，这不一定是。在数据存储库是 RDBMS 的情况下，这是一个非常合理的假设。

下一节将研究 SNMP 通知的传递和事件的处理。这个问题与数据轮询不同，因为通知传输是由代理启动的，而不是由管理器启动的。这里，我们简单地假设，在传统的管理平台中运行的事件处理程序和事件相关器运行的 Java 应用程序之间，我们有某种胶粘代码。

第四步是使用 HTTP 来收集数据作为 Java 应用程序来收集数据。启动后，轮询引擎从 RDBMS 中检索所有轮询定义和调度，并根据内部时钟定期通过 HTTP 轮询所有代理。轮询数据解释器检查为网络监控检索到的数据，如果推断出问题，可能会生成事件。然后，这个事件被发送到事件相关器，后者可以反过来决定用一个红色图标更新网络图。

管理器和代理之间的通信基于图 4 中的 HTTP。这个选择的原因是保持代理简单：它只需要一个 HTTP 服务器，这是当今越来越多的网络设备提供的一个功能。不过，还有许多其他的选择，我们将在第 4 节中看到。

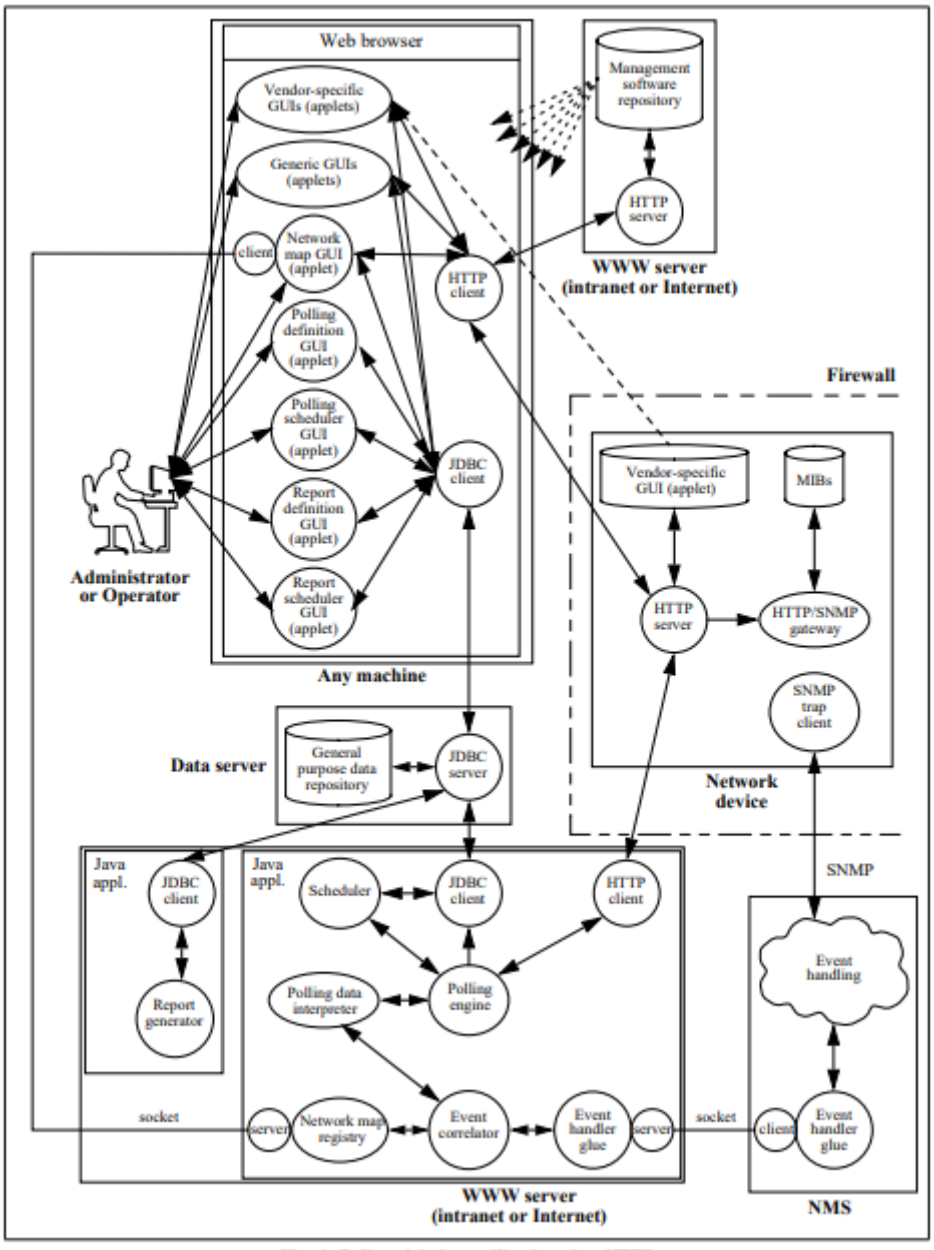

图 4. 拉模型：基于 HTTP 的数据轮询

轮询引擎如何知道通过轮询定义小程序或轮询计划小程序所做的修改？一个非常低效的解决方案是轮询引擎定期对数据存储库进行轮询。或者，我们可以在轮询引擎和轮询定义小程序之间有一个套接字，在轮询引擎和轮询调度程序小程序之间有一个套接字。否则，我们可以在两个小程序中有一个 RMI 客户机，在轮询引擎中有一个 RMI 服务器，这样每次更改定义或计划（即创建、修改或删除）时，都可以通过 RMI 通知轮询引擎。另一种解决方案是使用活动数据库而不是普通的 RDBMS，这样每当数据库中的轮询定义或调度发生更改时，触发器就会通过套接字或 RMI 将此信息发送到轮询引擎。

第五步也是最后一步是将报告生成迁移到 Web 技术。这可以通过 Java 应用程序简单地实现，通过 JDBC 访问数据存储库。这些数据由轮询应用程序存储，并由报表生成器独立访问。为了简单起见，我们描述了两个 Java 应用程序，即报表生成和数据轮询，它们是在同一台机器上进行的，但事实并非如此。报表定义小程序和报表调度程序小程序不必直接与报表生成器交互，因此小程序安全模型不存在任何约束。

到目前为止，我们还没有讨论的一个例子是在无人值守模式下执行常规管理。一些公司依靠完全自动化的网络管理，通过运行一个执行常规管理的网络管理平台，而没有运营商。当事件相关器检测到它认为是一个非常严重的问题时，它会在紧急模式下与管理员联系，以适应以下情况：警报器、寻呼机、电子邮件等。在这种情况下，没有必要用 Web 浏览器永久地阻塞机器，因为没有人会查看此浏览器。事件相关器仅与适用于无人参与模式的事件处理程序通信，并且没有套接字连接到网络映射注册表。

在这个阶段，数据收集和网络监控完全依赖于 Web 技术。他们不再需要昂贵的网管，专用于网络管理。SNMP 通知传递和事件处理程序是仍然以传统方式执行的唯一任务。

4. 推送模型

正如我们在第 3.1 节中看到的，推送模型基于发布/订阅/分发范式。它概括了网络监控和数据收集，以及当前过滤和传递 SNMP 通知的方式。推送模型最近被推送技术在网络上取得的巨大成功（例如，Pointcast、Backweb 或 Tibco 的 Tib 软件套件）所吸引。尽管这个模型在软件工程中是众所周知的，但它一直局限于 IP 网络管理中的 SNMP 通知传递：据我们所知，目前没有管理平台将它用于网络监控或数据收集。然而，我们声称它的设计使得它比拉模型更适合常规管理。本节将介绍如何在基于 Web 的网络管理中使用推送模型。[1]

使用推送技术的主要优势是节省网络带宽，并将部分 CPU 负担从管理者转移到代理。pull 技术造成的大部分网络开销都是由于数据收集和网络监视是非常重复的：管理器不断询问所有代理的内容中存在大量冗余。例如，在网络监控中，检查机器是否仍然活动的一种常见方法是每隔几分钟请求一个 MIB 变量，通常是其 sysobjectid（MIB-II）。如果管理器在每个轮询周期中无休止地将此 OID 封送给所有代理，则此方案非常低效。相反，使用推送模型，管理器与每个代理联系一次，订阅一次此 OID（推送数据定义），并指定代理应以何种频率（推送频率）发送此 MIB 变量的值（推送数据计划）；然后，管理器与代理之间不再有通信：所有后续通信都将从 m 代理到管理器（除非在管理器希望更改其感兴趣的 OID 列表或推送频率的罕见情况下）；代理记住管理器订阅的 MIB 变量以及发送这些变量值的频率。代理的方法是将推送定义和计划存储在本地存储上；如果这是 EPROM，则代理可以在重新启动后自行检索这些定义和计划；如果这是 RAM，则需要从将它们存储在数据服务器中的管理器检索它们。

将部分 CPU 负担从管理器转移到代理的关键是减少在 CPU 和内存方面对 NMS 的需求。大型网络的网络管理平台通常是大型的 Unix 或 Windows NT 服务器，购买和维护成本很高。另一方面，代理比以前更强大，大多数代理都可以在本地合理地进行一些处理（参见 Goldszmidt 通过委托计划进行管理的基本原理[8]或 Wellens 和 Auerbach 关于愚蠢代理的神话[25]）。

推送模型的另一个优点是，它可以与多播结合使用，以使网络管理更加健壮。当管理员订阅某些管理数据（即 MIB 变量或 SNMP 通知）时，他/她会告诉代理应该将数据发送到哪个管理器。他/她可以指定多播地址，而不是指定单播 IP 地

址。对于代理，将数据发送到单播或多播地址是透明的；唯一的要求是它们支持 IP 多播，而 TCP/IP 堆栈的现代实现通常都支持这种多播。因此，管理数据并行发送给多个管理器，这使得网络管理系统更加健壮；例如，一个管理器可能崩溃，而另一个管理器可以透明地接管（所谓的揾 ot Standby）。当使用 pull 模型时，我们也可以让代理将管理数据发送给多个管理器；但是在这种情况下，所有管理器都必须独立地请求数据，这会显著增加网络开销，并且会消耗更多的 CPU 周期。在这里，使用推送模型，备用管理器可以被动接收数据，直到配置为替换前一个管理器为止。即使这离容错系统还有很长的路要走，这个新特性对于那些网络对于业务的顺利运行至关重要的组织来说也是非常有吸引力的。

与拉模型相比，推模型引入了一个新问题：同步。如果管理器和代理具有不定期同步的内部时钟，它们可能会分离。对于网络监控来说，这不是问题，因为管理员不关心何时将代理配置为每 5 分钟向其管理器发送一次心跳，不管管理器是每 299 秒接收一次还是每 301 次。但对于数据收集来说，这是一个小问题，因为管理器可能接收到过多的数据，或者比预期的要少。为了建立报告，一些数据必须被任意丢弃，或者一些数据被插入。因此，当使用推送模型时，建议定期同步网络设备的时钟。这可能依赖于使用像 NTP 这样的协议，或者可能要求管理器每小时左右联系所有代理，并交换一些同步数据包。在这两种情况下，同步开销与从拉到推所带来的网络和 CPU 节省相比可以忽略不计。

现在让我们研究推模型的工程细节。第 4.1 节将介绍发布和订阅阶段，而第 4.2 节将介绍分发阶段的不同场景。

4.1.发布和订阅阶段

在第一个阶段，网络设备（代理）发布它支持的 MIB 以及它可以向管理器发送的 SNMP 通知。实现这一点的一个简单方法是使用小程序，如图 5 所示。首先，用户（管理员或操作员）在网络地图小程序上选择一个代理，并从该代理加载一个众所周知的 HTML 页面，其中列出了存储在该代理上的所有管理小程序。每个小程序都会发布一个由代理支持的 MIB（特定于供应商或通用），但发布此代理支持的 SNMP 通知的 MIB 除外。

在第二阶段，管理员将管理器订阅到 MIB 变量和 SNMP 通知。MIB 数据订阅小程序允许他/她选择 MIB 变量和推送频率。推送频率可以在 MIB 变量级别指定：对于给定 MIB 的所有变量，推送频率不必相同。推送频率等于第 3 节中考

虑的轮询频率。显然，通知订阅 applet 不必指定推送频率，因为通知本身是异步的。实际上，通知订阅小程序只是一个过滤器：它指定管理器感兴趣的通知。其他通知被通知生成器丢弃（参见图 6）。

我们可以使用一个单一的小程序来订阅任何 MIB 的任何 MIB 变量，而不是使用多个 MIB 数据订阅小程序（每个 MIB 一个）。但是，就像人们不喜欢在传统的网络管理平台上使用 MIB 浏览器一样，因为它们太基础了，而且更喜欢使用为每个 MIB 定制的管理 GUI，如果没有为每个 MIB 定制的可视化辅助工具，人们就不会高兴订阅 MIB 数据。

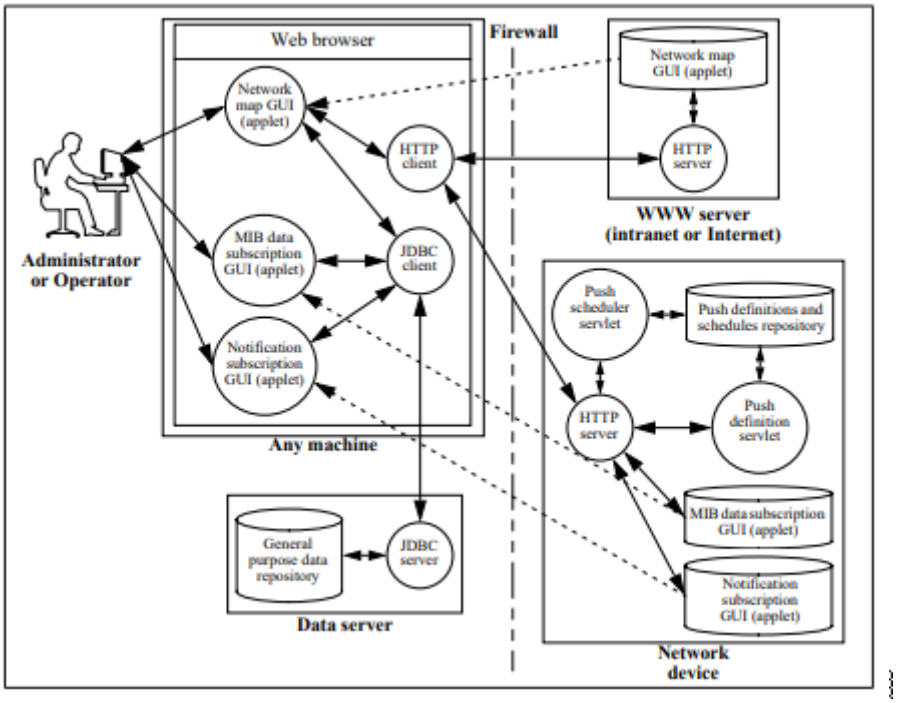

图 5. 推送模型：发布和订阅阶段

发布和订阅阶段如图 5 所示。MIB 和通知订阅的详细信息存储在数据服务器中。如果代理丢失了所有推送配置数据，那么管理器可以在无人参与模式下重新发送该代理的所有定义和计划：管理员不必通过 GUI 手动重新输入所有定义和计划。数据服务器的通用数据存储库包括（i）管理器订阅的 MIB 数据的定义和调度，（ii）管理器订阅的通知的定义，以及（iii）网络映射小程序用于构建其 GUI 的网络拓扑定义。在现实生活中，这三个逻辑数据存储库实际上可以存储到不同的数据库中，或者单个数据库中。

4.2.分配阶段

在分发阶段，数据收集和网络监视的情况与通知传递和事件处理的情况略有不同。为了便于与 pull 模型的比较，本文将重点讨论数据采集和网络监控，即如何用 push 技术取代数据轮询。通知传递和事件处理在[14]中有详细介绍。但是，我们要强调的是，下面描述的解决方案同样适用于这两种情况：代理和管理器之间的通信问题是相同的，只有在管理器侧运行的 Java 应用程序是不同的。

图 6 中描述的通用数据存储库包括七个不同的存储库：第 4.1 节中列出的三个存储库，加上（iv）事件处理程序定义存储库，（v）事件处理程序调用日志，（vi）推送数据存储库，以及（vii）推送通知存储库。和以前一样，在现实生活中，所有这些逻辑上不同的数据存储库实际上可能驻留在一个或多个数据库中。

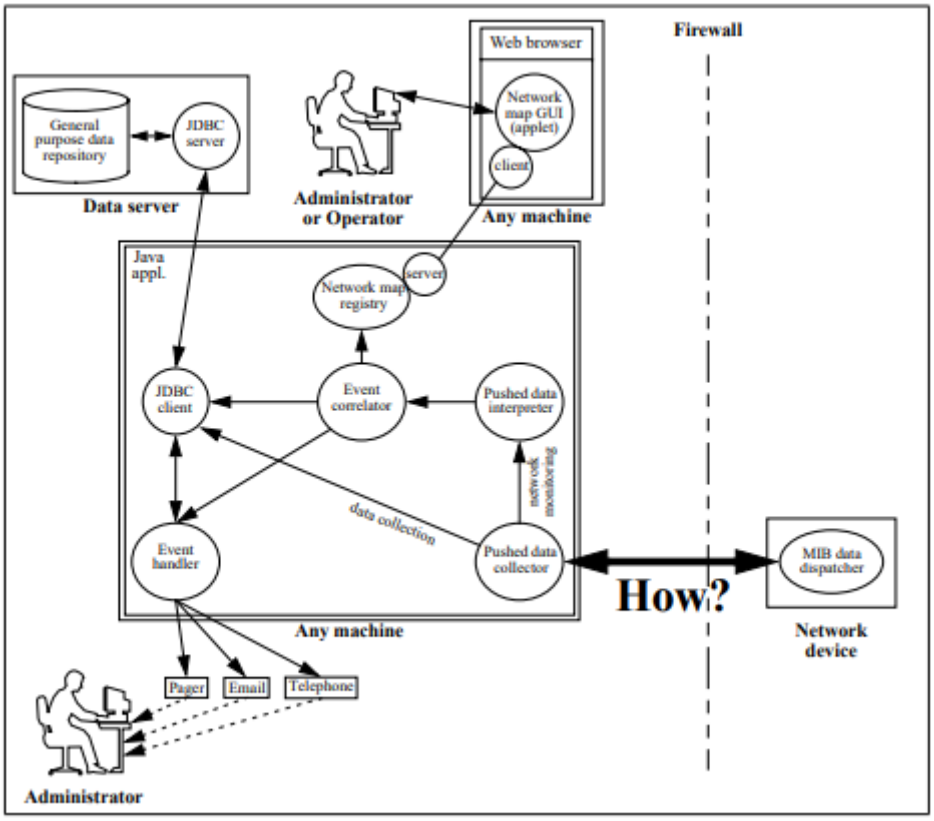

图 6. 推模型：分布阶段

与图 4（pull 模型）相比，我们不再有轮询引擎；相反，我们有一个推送的数据收集器，它收集与网络监视或数据收集相关的数据。这些数据通过 JDBC 存储在数据服务器上。由于 Java 代码的执行速度较慢，因此通过大量存储数据来提高性能可能是一个好主意。与 pull 模型中的轮询引擎一样，推送数据收集器将收集用于网络监控的数据发送给推送数据解释器。如果推送的数据解释器检测到异常情况，例如设备不再发送任何数据，则会以发送到事件相关器的事件的形式生

成警报。事件相关器还接收通知形式的事件（此处未显示），并标识网络问题。当事件未被另一个事件屏蔽时，它可以调用事件处理程序，在这种情况下，对事件处理程序的调用将记录在数据服务器中。

从“拉”到“推”的主要困难在于，数据传输现在由代理启动，而不是由管理器启动，而 HTTP 客户端仍在管理器端，HTTP 服务器在代理端（见图 6）。不知怎么的，客户机和服务器站错了方向！我们希望服务器启动通信，而通信总是由客户机/服务器体系结构中的客户机启动。为了解决这个问题，我们可以在三种通信技术[21]之间进行选择：http、sockets 和 rmi。对于其中的每一个，我们现在都将提出确保客户机和服务器之间某种持久连接的方法。[1]

在本节中，我们还将讨论第 2.4 和 3.2.3 节中已经提到的一个问题：如何跨越防火墙？这个问题并不特定于推模型，我们现在将描述的工程权衡同样适用于 Pull 模型（例如，在图 4 中，轮询引擎和网络设备之间的通信不一定依赖于 HTTP）。

4.2.1. Socket

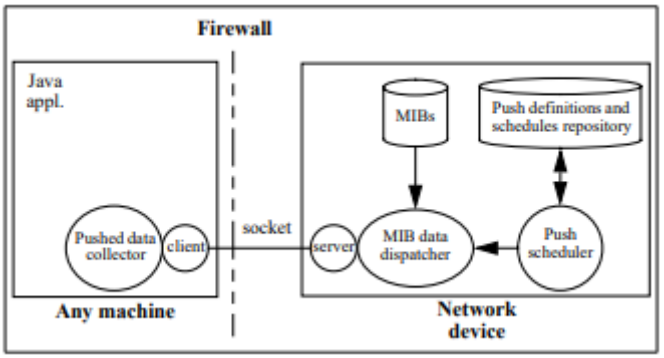

图 7. 推送式：通过插座分配

套接字是双向的这一事实解决了服务器启动的通信问题：我们可以像往常一样打开从客户机到服务器的套接字，然后，只使用它将 MIB 数据从套接字服务器发送到套接字客户机。为了确保此连接保持持久性，在管理器端，推送的数据收集器在创建套接字时在其上设置无限的超时值。如果底层 TCP 连接因任何原因超时，则管理器（即推送的数据收集器）有责任通过创建新的套接字重新连接到代理。

这种基于套接字的解决方案具有一个很大的优势：简单性。套接字编程非常简单，尤其是在 Java 中。但它也存在两个潜在的缺点。首先，如果管理器或代理的底层操作系统一直在超时连接（例如，因为管理员无法控制套接字的超时值，而这个超时值恰好低于推送频率），那么这个解决方案显然是不合适的。重复的

套接字创建和超时不仅会导致网络和 CPU 开销，更糟糕的是，我们不能冒险让通知传递依赖于这样一种通用类型的持久连接；如果以前的超时，代理（而不是管理器）必须有一种方法来创建新的连接。第二，如果我们需要跨越管理器和代理之间的防火墙，那么套接字可能有问题。正如我们在第 2.4 节中看到的，大多数防火墙过滤掉了 UDP，并且只允许几个 TCP 端口通过。因此，无论我们使用 TCP 还是 UDP 套接字，防火墙在默认情况下都不会让套接字通过。因此，为了使这个基于套接字的解决方案工作，需要修改防火墙系统。正如我们在第 2.4 节中提到的，对于大型组织来说，这可能不是问题，因为他们要么在防火墙方面拥有专业知识来设置 UDP 中继，要么更改 TCP 过滤规则，要么他们可以负担昂贵的外部顾问来完成这项工作。但对于中小企业来说，这可能是一个问题，他们通常缺乏这种专业知识，而且对于他们来说，昂贵的外部顾问可能不是一个选择。

4.2.2. RMI

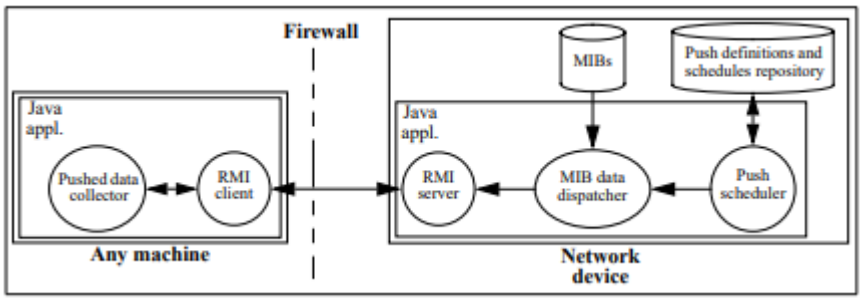

图 8. 推送模式：通过 RMI 分配

和套接字一样，RMI 提供了一个双向关联：一旦一个 RMI 客户机绑定到一个 RMI 服务器，两个客户机都可以向另一个发送数据。在设计方面，RMI 是一个很好的解决方案，因为它提供了一个完全面向对象的网络管理视图。它为高于 MIB 变量的网络管理应用程序设计器提供了语义，并使复杂应用程序的设计更加容易[12]。

但 RMI 也有一些缺点。首先，它要求在所有代理中嵌入一个完整的 JVM（而不是像 EmbeddedJava 平台中包含的那样的轻量级 JVM）。今天很少有网络设备提供这个特性，主要是因为 Java 芯片没有遇到 Java CAMP 所期望的成功。此外，许多设备在一段时间内不会拥有完整的 JVM，因为该软件在价格敏感的设备（如打印机服务器）范围的底部占用了大量空间。其次，当前的 RMI 实现非常缓慢，在内存和 CPU 方面使用了大量资源；因此，基于 RMI 的网络管理不可扩展。这在未来的实现中可能会有所改善，但其他分布式面向对象平台（如 CORBA 或

DCOM）也面临同样的问题，这一事实促使我们相信，像 RMI 这样的面向对象技术在未来几年内最多仍将是网络管理领域的一个利基市场。第三，RMI 通信实际上基于对应用程序透明的套接字；因此，我们再次面临防火墙问题。事实上，RMI 的情况比以前更糟，因为我们不再控制套接字使用的端口。RMI 套接字对应用程序是透明的，因此即使 RMI 服务器运行在已知端口（1099/TCP 或 1099/UDP[9]）上，RMI 客户端也可以绑定到任何端口（在前一种情况下，管理员可以控制客户端和服务器端使用的端口）。因此，为了跨防火墙使用 RMI，必须将 RMI 特定软件添加到防火墙系统中；并且，目前所有防火墙系统都不支持 RMI 中继（如果 RMI 随着时间的推移而成功，它们可能在将来）。

基于所有这些原因，我们不能合理地期望在不久的将来有很大比例的网络设备支持 RMI。目前，套接字似乎是一种更好的网络管理通信方式。

4.2.3. 超文本传输协议

HTTP 不共享我们为套接字和 RMI 开发的属性。HTTP 服务器无法通过预先存在的持久连接启动数据传输。所有 HTTP1.1 方法都依赖于请求/响应协议，因此服务器在没有事先收到请求的情况下无法发送响应。通过让 HTTP 服务器从 HTTP 客户机向单个请求发送无限多的响应是不可能解决这一问题的：响应可能是碎片化的，但使用 100（continue）状态是有限的[7]。在这方面，SNMP 和 HTTP 是不同的。两者都基于客户机/服务器通信模型；但是，除了一个 snmpv2 陷阱外，snmp 是所有操作（get、set、inform…）的请求/响应协议，它依赖于单向异步传输协议；相反，http 是所有方法（get、post、head…）的请求/响应协议。

为了允许管理器和代理之间基于 HTTP 的通信，因此我们必须依赖不同的工程解决方案。我们建议在代理上添加一个 HTTP1.1 客户机，在管理器上添加一个 HTTP1.1 服务器，这样客户机/服务器通信就正常了。

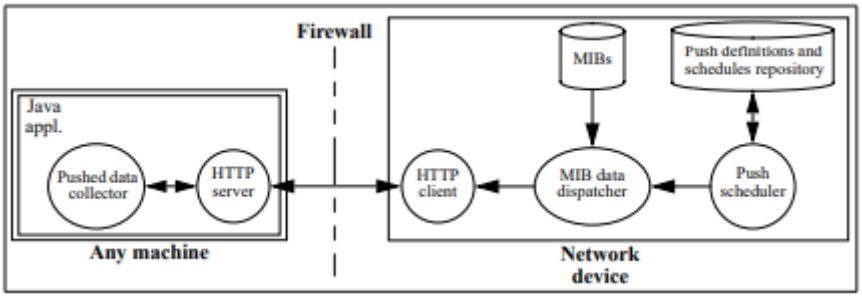

图 9. 推送模型：通过 HTTP 分发

这个解决方案有几个优点。首先，它不依赖于非直观的设计，如 4.2.1 和 4.2.2 节：客户机在代理端，服务器在管理器端。第二，在持久连接超时的情况下，代理可以立即重新连接：它不需要依靠管理器来完成这一操作；这提高了健壮性，并避免了代理希望向管理器发送数据但管理器尚未重新连接到代理时的时间窗口。第三，如果管理应用程序运行在组织的外部 Web 服务器上，则防火墙系统完全不需要更改；如果管理应用程序运行在其他计算机上，则需要对防火墙系统的设置进行细微更改。这个解决方案的主要缺点是它需要一个 HTTP 服务器包含在运行在管理端的 Java 应用程序中。这使得大型程序（网络管理应用程序）更大、更难调试、执行速度更慢，并在运行它的计算机上产生更大的内存占用。

5. 折叠的网络管理平台

如果将图 4、图 5 和图 6 结合起来，我们就会意识到我们已经完全拆除了网络管理平台。我们不再需要一个昂贵的管理站，专门用于网络管理，并运行昂贵的平台特定的软件。相反，我们可以在运行任何地方的 Web 浏览器中下载小程序，我们可以在装有 JVM 的任何机器上运行 Java 应用程序，我们可以使用现有的 RDBMS 来存储管理数据仓库，并且我们可以使用已经无处不在的内部 WWW 服务器。我们将其称为崩溃的网络管理平台，与崩溃的主干网概念进行类比。

6. 结论

基于 Web 的网络管理不仅仅是一种时尚。我们在本文中证明，它是基于可靠的技术基础，并提供了成本效益的替代品，传统的 IP 网络管理平台目前在业界使用。我们相信这将是一个光明的未来，因为它可以为这个市场的几乎所有参与者节省大量的资金。首先，它使客户能够节省昂贵的网络管理平台的成本，并有助于利用现有的投资，如 RDBMS。其次，它允许网络设备供应商大幅降低其特定于供应商的管理 GUI 的开发成本；他们不再需要将附加组件移植到运行在不同操作系统上的许多管理平台；现在，他们可以开发一个可由任何计算机上运行的任何 Web 浏览器下载和执行的单个小程序。第三，它通过允许供应商将其专有 GUI 的上市时间减少到零，为向 Web 技术迈进的供应商提供了竞争优势。最后，它使初创公司与规模更大、知名度更高的公司处于公平竞争中，尽管它们的市场份额很小，但仍能获得综合网络管理。事实上，只有网络管理平台供应商才会输给网络。他们不能强迫客户使用专有数据库，并且他们不能为他们出售的 Java 应用程序和通用 GUI 小程序收取高昂的价格。

本文提出了两种基于 Web 技术的设计方法：拉模型（pull model）和推模型（push model），这两种方法非常适合于即席管理。提出了两种模型的数据采集和网络监控的工程解决方案。附带的一篇文章[14]描述了如何使用推送模型执行通知传递和事件处理。为了实现推送模型，当数据分发依赖于 HTTP 或套接字时，托管设备中所需的更改将受到限制。对于 pull 模型，它们更为有限：我们只需要一个 HTTP 服务器和一个 HTTP-to-SNMP 网关。这些模型都没有要求在所有代理中嵌入完整的 JVM：顶级网络设备可以选择使用 RMI。

我们不久将开始实施基于 Java 的网络管理平台的原型，作为本文提出的概念的证明。该平台将同时支持拉式和推式模型。要管理的测试设备将由位于瑞士洛桑的路由器供应商 Lightning 提供。我们将在管理软件中添加一个 push 调度程序和一个 mib 数据调度程序，该软件已经包括一个 HTTP 服务器，并且将开发一个特定于供应商的小程序来支持其专有的 mib。我们希望在会议上报告，并在最后一篇论文中整合该原型所取得的成果和经验。

致谢

这项研究部分由瑞士国家科学基金会（FNRS）资助的 SPP ICS 500 3-45 311。作者要感谢 J.SH 等人讨论 BER 编码和 SNMP 的效率，G. Madhusudan 讨论分布式 Java 应用中的通信，H. Cogliati 对本文进行校对。

首字母缩略词

API	应用程序编程接口	NMS	网络管理站
ASN.1	抽象语法符号 1	NTP	网络时间协议
BER	基本编码规则	OID	对象标识符
CORBA	公共对象请求代理体系结构	PC	个人计算机
CPU	中央处理单元	PER	打包编码规则
DBMS	数据库管理系统	RAM	随机存取存储器
DCOM	分布式公共对象模型	RFC	征求意见
EPROM	电可擦可编程只读存储器	RMI	远程方法调用
GIF	图形交换格式	SME	中小企业
GUI	图形用户界面	SMI	管理信息结构
HTML	超文本标记语言	SNMP	简单网络管理协议
HTTP	超文本传输协议	SSL	安全套接字层
IETF	互联网工程工作队	TCP	传输控制协议
IP	Internet 协议	TLS	传输层安全性
JDBC	Java 数据库连接	UDP	用户数据报协议
JDK	Java 开发工具包	UPS	不间断电源
JIT	恰好及时	URL	统一资源定位器
JMAPI	接口	VPN	虚拟专用网络
JVM	Java 虚拟机	W3C	万维网联盟
LAN	局域网	WAN	广域网
MIB	管理信息库	WBEM	基于 Web 的企业管理
MTBF	平均故障间隔时间	WWW	万维网

参考文献

[1] AdventNet SNMP Package. See <URL:http://www.adventnet.com/products.html>.

[2] B. Bruins. “Some Experiences with Emerging Management Technologies”. In *The Simple Times*, 4(3):6-8, 1996.

[3] J. Case, K. McCloghrie, M. Rose and S. Waldbusser (Eds.). *RFC 1902. Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*. IETF, January 1996.

[4] D.B. Chapman and E.D. Zwicky. *Building Internet Firewalls*. O’Reilly & Associates, Sebastopol, CA, USA, 1995.

[5] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. Internet draft <draft-ietf-tls-protocol-05.txt> (work in progress). IETF, November 1997.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee (Eds.). *RFC 2068. Hypertext Transfer Protocol -- HTTP/1.1*. IETF, January 1997.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee (Eds.). *Hypertext Transfer Protocol -- HTTP/1.1*. Internet draft <draft-ietf-http-v11-spec-rev-03.txt> (work in progress). IETF, March 1998.

[8] G. Goldszmidt. *Distributed Management by Delegation*. PhD thesis, Columbia University, New York, NY, USA, December 1995.

[9] IANA. *Protocol Numbers and Assignment Services*. Available at <URL:http://www.iana.org/numbers.html>. This Web site updates RFC 1700 which is now obsolete.

[10] ITU-T. *Recommendation X.690. Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. ITU, Geneva, Switzerland, July 1994.

[11] ITU-T. *Recommendation X.691. Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)*. ITU, Geneva, Switzerland, April 1995.

[12] J.P. Martin-Flatin, S. Znaty and J.P. Hubaux. “A Survey of Distributed Enterprise Network and Systems Management”. To appear in *Journal of Network and Systems Management*, March 1999. Submitted in December 1997.

[13] J.P. Martin-Flatin. *IP Network Management Platforms Before the Web*. Technical Report SSC/1998/021, SSC, EPFL, Lausanne, Switzerland, July 1998.

[15] J.P. Martin-Flatin. “The Push Model in Web-Based Network Management”. Submitted to the *18th IEEE INFOCOM Conference on Computer Communications (INFOCOM’99), New York, NY, USA, March 1999* . July 1998

[16] N. Mitra. “Efficient Encoding Rules for ASN.1-Based Protocols”. In *AT&T Technical Journal*, 73(3):80-93, 1994.

[17] P. Mullaney. “Overview of a Web-based Agent”. In *The Simple Times*, 4(3):8-12, 1996.

[18] G. Neufeld and S. Vuong. “An overview of ASN.1”. In *Computer Networks and ISDN Systems*, 23:393-415, 1992.

[19] E. Rescorla. *HTTP Over TLS*. Internet draft <draft-ietf-tls-https-01.txt> (work in progress). IETF, March 1998.

[20] M.T. Rose. *The Simple Book: an Introduction to Networking Management*. Revised 2nd edition. Prentice Hall, Upper Saddle River, NJ, USA, 1996.

[21] M.T. Rose and K. McCloghrie (Eds.). *RFC 1155. Structure and Identification of Management Information for TCP/IP-based Internets*. IETF, May 1990.

[22] P. Sridharan. *Advanced Java networking*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[23] W. Stallings. *SNMP, SNMPv2 and CMIP: the Practical Guide to Network Management Standards*. Addison-Wesley, Reading, MA, USA, 1993.

[24] W. Stallings. “SSL: Foundation for Web Security”. In *The Internet Protocol Journal*, 1(1):20-29, 1998.

[25] Sunsoft. *Java Management API Architecture*. Revision A. September 1996.

[26] C. Wellens and K. Auerbach. “Towards Useful Management”. In *The Simple Times*, 4(3):1-6, 1996.

传记

J.P.Martin Flatin 目前正在准备 EPFL 的博士论文。1990 年至 1996 年，他在英国雷丁的欧洲中期天气预报中心工作，负责网络和系统管理、安全、网络管理和软件工程。从 1988 年到 1990 年，他在法国一个大城市的地理信息系统工作。1986 年，他从法国里昂的欧洲经委会（ECAM）获得了电子商务与我的混合理科硕士学位。他的主要研究方向是分布式网络管理。他是 IEEE 和 ACM 的成员。