JUNIPER NETWORKS | Engineering Simplicity

白皮书

JUNIPER APSTRA 架构

通过专为完整网络生命周期定制的自动化方案来解决极富挑战性的数据中心问题

目录

执行摘要

数据中心网络的构建、部署、运维、管理和故障排除工作可能很困难、成本高昂且需密集投入资源。2019 年的一份 Gartner 报告表明，简化 IT 基础架构属于当今企业的首要战略优先事项。Juniper® Apstra 专为使数据中心网络的完整生命周期实现自动化所设计。

本文将介绍数据中心网络团队当前面临的一些最困难的问题，并会诠释 Apstra 的架构可如何解决这些问题。

简介

当今的数据中心网络架构和运维团队面临着四项主要挑战：

- **组合。**将不同供应商提供的基础架构（每个基础架构都具有不同功能）整合到一个一致且可平稳运行的系统中可能非常困难。Juniper® Apstra 创建了一个连贯的整体（一份蓝图），其中包含根据所表达的意图进行系统部署所需的全部信息。这份蓝图会被推送到物理基础架构，并允许运维人员提供可靠且易于使用的服务。
- **可靠的变化。**变化是数据中心的常态。变化可能来自尝试添加新服务或进行扩容的运维人员，也可能因基础架构的故障情况所导致。无论出于哪种原因，运维人员都必须找到更好的变化应对之法。近年来的各项研究表明[1]，有 65% 到 70% 的网络服务中断是由执行计划变化时出现人为配置错误所导致的。
- **可扩展性。**为鼓励创新，并顺应技术领域不可避免的快速变化，数据中心需要能够轻松、顺畅地添加各种新功能。随着技术持续发展，供应商会不断创新并推出各项新功能，而运维人员可以利用这些功能来保持竞争力。这种演变会导致设计时变化，从而需要使用新的行为契约（或参考设计）来控制要部署的服务。换言之，组合必须具有可扩展性，使得进一步创新或满足新需求成为可能。
- **可扩展性。**为大规模解决所有这类问题，数据中心网络必须能够实现发展，并顺应创新和不断增加的复杂性。

Apstra 架构的首要目标就是直接解决这些问题。

主要挑战：组合

现如今，运行计算/网络/存储基础架构的运维人员面临的主要挑战便是组合。组合是指能够创建一个连贯的整体（而不仅仅是其功能组件的总和），并在出现不可避免的变化时随着时间的推移保持这种连贯性，同时向消费者提供网络服务。就其核心而言，Apstra 旨在应对组合挑战。

现代数据中心就是横向扩展的计算机。这些数据中心要求操作系统提供资源管理和进程隔离等功能，这些功能与主机操作系统目前在单台计算机上所提供的功能相似。

计算虚拟化已为单一服务器计算实现了这样的结果。然而，对于作为横向扩展计算工具的数据中心而言，数据中心运维人员首先需要将其组合起来，只有这样，您才能提供资源分区。

[1] 示例包括：https://www.computerweekly.com/news/2240179651/Human-error-most-likely-cause-of-datacentre-downtime-finds-study；https://www.networkworld.com/article/3142838/top-reasons-for-network-downtime.html；https://www.ponemon.org/library/national-survey-on-data-center-outages。

为什么组合会这么难?

组合之所以如此具有挑战性,有两个主要原因。首先,用于组成一个连贯整体的要素可能来自不同的供应商、具有不同的功能或者只能通过不同的 API 来使用。

其次,您可能需要组合您的基础架构,这样才能交付各种服务,而每个服务都具有多个功能层面,如可达性、安全性、体验质量和可用性。

当您将多个服务实例进行实例化时,除非您完全了解如何将这些服务映射到实施机制之中,否则,这些组件之间的交互可能会导致出现服务中断。随着基础架构的功能随时间推移而不断发展,您需要能够在不破坏现有系统的情况下利用并引入新的创新举措。

Apstra 借助参考设计应对挑战

参考设计是 Apstra 架构用于解决组合挑战的核心概念。参考设计是一种行为契约,它定义如何将用户的业务意图映射到实施机制,以及必须满足哪些期望才能将意图视为已实现。

参考设计所决定的是:

- 物理和逻辑组件的角色和职责
- 如何将服务映射到实施机制
- 需要满足的期望(即:要观察的情况)

在参考设计中,物理和逻辑要素的角色和职责将得到很好的定义,而这又会对建模范围带来限制,同时将支持最小但完整模型的规范。参考设计还将决定意图到实施机制的映射方式。要知道,这样的映射可让故障排除实现自动化,并可提供基于系统组合方式的知识的强大分析功能,而不是会对您的网络中所发生的情况进行逆向工程。

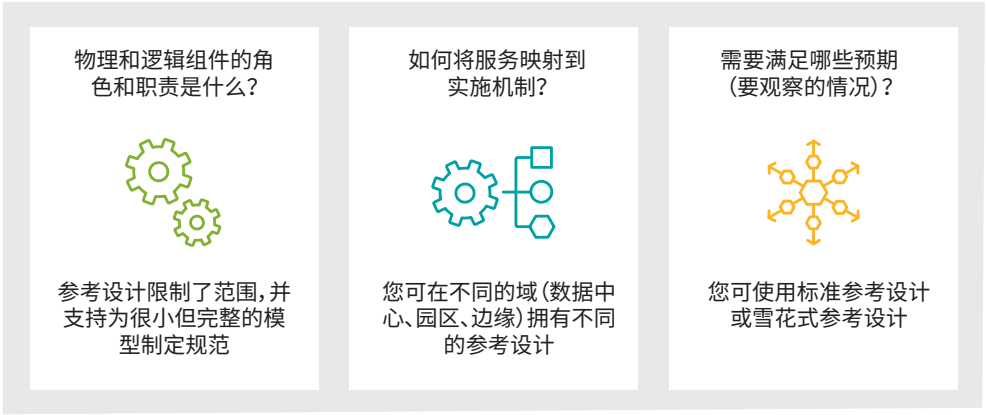

图 1:参考设计是一种决定性质的行为契约。

在不同的参考设计中,相同的意图可能会采用不同的、可能更新的、更具创新性的机制来执行。了解此映射可以识别根本原因,从而向运维人员直接指出对服务造成影响的问题的原因。

参考设计还界定了需经验证的预期。例如,参考设计可以规定,每个叶应该与每个主干有一个 BGP 会话。这样,丢失的和意外的 BGP 会话将很容易被识别。

定义参考设计是一项增值活动,必须由网络专家来完成。有了 Apstra,专业知识将成为系统的一个显式部分,而不是只会存在于专家的脑海中。这样,就可以对专业知识进行建模并将其嵌入到系统中,而不是进行硬编码。

知识就是力量：可靠地处理变化

> *“生存下来的不是最强壮的物种，也不是最聪明的物种，而是对变化最敏感的物种。”*
>
> - 查尔斯 · 达尔文，1809 年

变化是数据中心的常态，因此，可靠地处理变化始终是首先要考虑的问题。变化源自两方面：

1. **运维人员的意图。**运维人员可能希望添加诸如虚拟网络之类的新服务、在基础架构中添加/移除资源或者在一些策略上做出改变。在变化实施时，运维人员可以控制变化。Apstra 会使用**有状态编排**来可靠地实现运维人员的意图。
2. **托管基础架构中的故障。**变化也可能来自基础架构的故障情况（如丢包过多或流量失衡）。运维人员无法控制这种变化，并且操作条件的体量通常是压倒性的。Apstra 可提供基于意图的分析，以帮助运维人员应对操作状态的变化。

能否可靠地处理这两种类型的变化，这取决于您对基础架构状态的了解情况。如下两个条件必须得到满足：

1. 知识必须置于上下文中加以诠释，而这意味着条件和期望之间的关系是可被理解的，正如**上下文模型**部分所描述的那样。
2. 知识必须及时，这意味着它能反映出当前情况，如**实时监控和通知**部分所述。

有状态编排

有状态编排利用意图使源自运维人员的变化可靠。行动的有状态编排流程如图 2 所示。

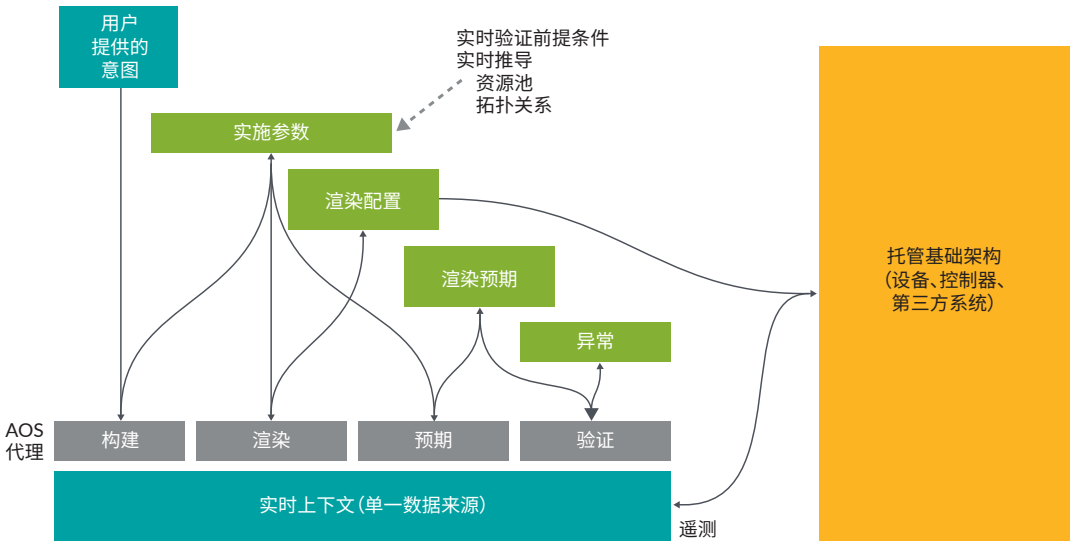

图 2：有状态编排流程

如果您希望在现有全功能系统上做变更，则有状态编排允许用户仅提供变化意图。这是以一种与实施无关的方式完成的，而这使得意图指定变得更简单且更不容易出错。在有状态编排期间所执行的一组完整举措包括：

步骤 1：实时验证前提条件。这项新要求是否会违反某些策略?例如，是否允许您创建这个虚拟网络，或者新要求是否会带来某些安全漏洞?或者，您是否可以在知道其他一些设备已脱机时，将某设备置于维护模式，并且您了解，将该设备脱机会使您的系统变得易受攻击?有了有状态编排，所有这些问题都会被问及，并可根据上下文图表进行实时验证。

步骤 2：实时推导实施参数。为了实现意图，必须在特定的托管要素上提供并激活适当实施机制的特定参数。Apstra 会使用参考设计详述如何实现意图，结合其对当前上下文的理解会使系统能够自动执行实时参数推导。运维人员意外输入错误的命令、接口、IP 地址或 VLAN 的可能性为零。

步骤 3：实时多配置部署。Apstra 可执行实时多配置部署，这可能会对多个要素进行自动化配置部署，这些要素可能来自不同的供应商且采用特定于技术的 API。

步骤 4：实时生成期望。充当行为契约的参考设计允许 Apstra 执行实时期望生成，这会描述为了声明结果已满足意图而需要满足的条件。

步骤 5：实时验证期望。Apstra 之后会触发一系列遥测和基于上下文的操作分析，以执行实时期望验证。

步骤 6：经验证的服务成果。在此过程结束时，用户将可明确观察到经验证的服务成果。意图或预期操作状态的每项变化都能够实时反映在上下文模型中，并且任何需要关注变化的组件也将会获得实时通知。Apstra 可以使用基于意图的分析从原始操作数据中提取相关知识。

对于无状态编排而言，缺失了许多步骤（如图 3 所示）。配置部署成果是无状态编排的典型内容，即：将配置推送到可能的多个系统，并验证配置是否已被托管要素所接受。而这正是无状态编排通常会止步的地方——不存在服务期望的见解，也不存在判断是否满足这些期望的验证。自动化的服务成果验证是缺失的，并且，独立的各系统将为用户提供有关原始操作状态的“单一管理平台”。将由用户来执行视觉发现，以了解成果是否已经实现。这个单一管理平台通常会包含过多信息，并且缺乏特定于实施的上下文，这使得视觉发现极其困难，且存在主观性。

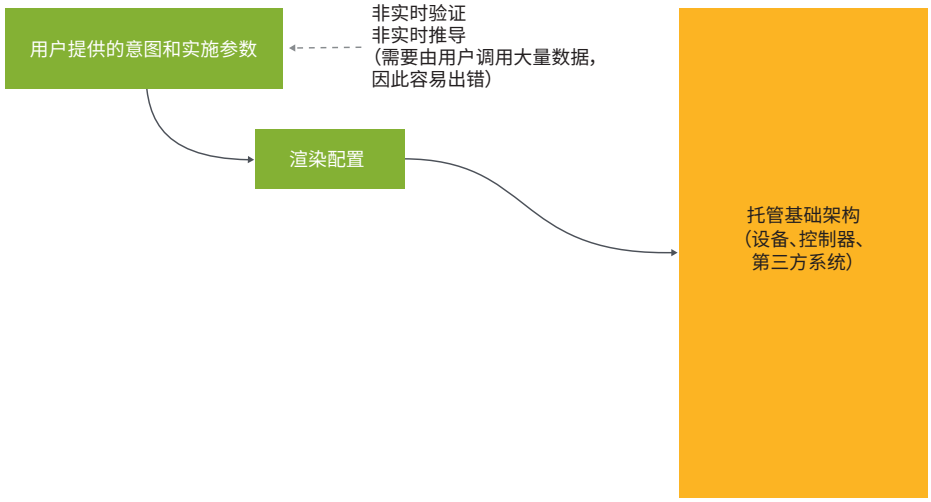

图 3：典型的无状态编排

基于意图的分析

基于意图的分析 (IBA) 会从原始遥测数据中提取知识，以此协助运维人员应对其基础架构中的操作状态变化。正如前面所提到的，必须具备可实时进行查询的上下文，才能使用 IBA。

知识提取包含两个步骤：

1. **检测关注条件**（要观察的情况）。条件检测由 IBA 探针完成。
2. **将关注条件进行自动分类，并推导出这些条件之间的关系**。条件的语义内容不尽相同。换句话说，这些内容中的一些会比另一些更重要。了解条件之间的关系是至关重要的，这样就可以查明重要的可操作条件（根本原因），并了解哪些条件仅是后果而已，当重要的根本原因得到处理时，这些后果会消失。由 IBA 的根本原因识别组件进行条件分类和因果关系推导。

关注的建模条件

IBA 可使用以下四个分类对关注条件进行建模：异常、症状、影响和根本原因。

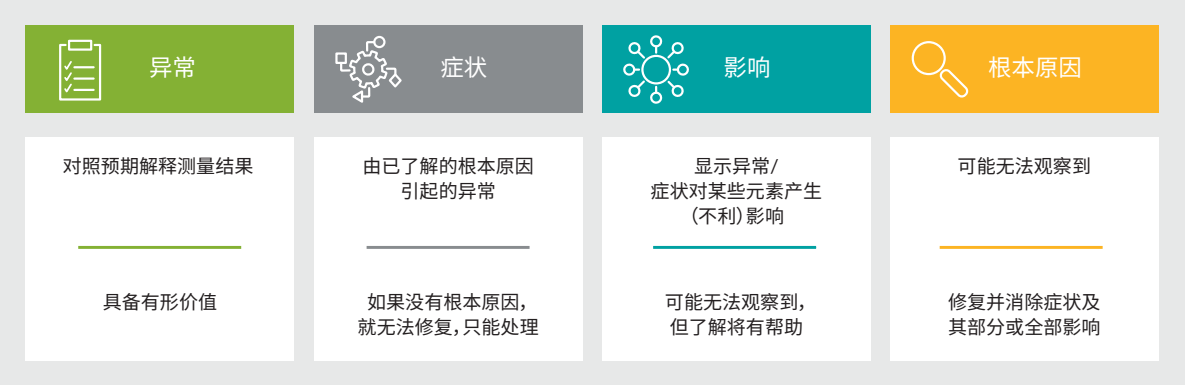

图 4：条件分类

异常

异常本质上代表了对测量结果的解释，或者测量结果的某种聚合，它与某种预期相反，因此比简单的测量具有更大的切实价值。

症状

症状是由得到充分了解的根本原因所引起的异常。它们通常很容易就能观察到，但无法直接修复，只能缓解，这就像给病人服用阿司匹林可以缓解发烧状况，但不能治疗疾病本身一样。如果不治病，那就会继续发烧。重要的是，当根本原因得到解决时，症状就将消失，因此，症状是不可操作的，它们只对于诊断根本原因方面有效。

影响

影响可表明，由于异常而发生了一些情况。影响可能并不总是可以观察到，但了解影响很有用。例如，您可能希望了解某重要客户是否因设备故障或数据包丢失过多而受到了影响，若如此，他们下一步会致电您进行投诉。

当影响无法被观察到时，可以根据对意图和用于实施意图的实施机制的了解来量化影响。了解影响有助于运维人员优先考虑对运维造成最大影响的根本原因和异常情况。

根本原因

在所有条件中，根本原因是最重要的。根本原因可能并不总是可被观察到，因此，很难加以诊断，但它们会引起许多症状和影响。由于修复根本原因可以消除相关的症状和影响，因此根本原因是可操作的对象。

IBA 探针：清楚您所了解的

有这样一句适用于数据中心网络的名言："让我们陷入困境的不是无知，而是深信为真理的谬论。"意图能够描述并形式化常态，这是我们所了解的。IBA 可确保我们所了解的内容的确符合预期。

> *"让我们陷入困境的不是无知，而是深信为真理的谬论。"*
>
> **- 马克·吐温**

IBA 探针负责检测关注条件，并由作为参考设计一部分的行为契约所驱动。IBA 探针可获取数据，应用某些处理方法，然后将结果与预期进行比较。IBA 探针本质上是可配置的数据处理通道，允许用户设置关注条件（即，要观察的情况）。

数据源和查询

探针的初始阶段通常是数据源，它负责获取原始遥测数据。数据源阶段的配置包含一项查询操作，该查询可以精确地表示哪些数据需要收集，这是一项十分强大的功能。

例如，假设运维人员对分析仅适用于结构接口的等价多路径路由 (ECMP) 的不平衡表示关注，并且有报告表示，特定版本的操作系统在 ECMP 哈希算法中引入了错误。查询可以表示需要在每个架顶式交换机上收集接口计数器，但仅限于交换矩阵接口和运行特定版本 x.y.z. 交换机操作系统的交换机。既然设置了这种要观察的情况，运维人员就不必担心变化了。如果将新的交换矩阵链路添加到符合条件的交换机中，该交换机将自动包含在分析中。如果添加了新交换机，则其会自动被包含在内。如果有人将另一台交换机升级到 x.y.z 版本，它将自动被包含在内。IBA 探针无需维护。

阶段处理器

在许多情形中，运维人员不会关注原始遥测数据的瞬时值，而是会关注聚合或趋势。IBA 包含用来对计算平均值、最小值/最大值、标准偏差等信息进行聚合的阶段处理器。运维人员随后可以将这些聚合与预期进行比较，以便他们能识别聚合指标位于指定范围内还是在指定范围外，若超出限制则会被标记为异常。

然后，运维人员可能希望检查该异常的存续时间是否超过特定阈值的时间段，并且仅当超过阈值时才会标记该异常，以避免标记瞬时或临时状况的异常。运维人员只需配置后续阶段以包含名为"状态时间处理器"的工具即可实现这一点。

超越数字数据

探针不仅能对数字数据进行操作。例如，它们可被用于验证控制措施和数据平面的正确性。比如在给定意图的情况下，运维人员可以使用查询操作来配置数据源处理器，以构建期望的路由或转发表。

在以太网 VPN (EVPN) 环境中，意图将包含各种内容，如所存在的虚拟网络、这些虚拟网络的端点所在的位置以及有关实施机制的信息。然后可以将该预期表与来自遥测的预期表进行比较，并在发现存在不匹配时标记异常。

还可以将探针配置为用于跟踪转发和路由表大小的突然变化，并在趋势与预期不符时发出警告。“预期的”阈值也可以根据意图经动态计算得出，它可以是虚拟网络数量、端点数量、虚拟隧道端点 (VTEP) 数量、具有某些问题的 VTEP 数量等的函数。其可能性是无限的。

通过简单的 REST 调用或通过使用 GUI，可以按声明方式创建、激活并停用探针。探针激活也可作为遥测的触发器发挥作用。换言之，仅当探针关注特定的遥测数据时，才会收集此数据，因此，探针本质上是一种遥测收集配置机制。数据源处理器中的查询使配置能够根据需要进行细化和精确，从而消除了“数据囤积混乱”的现象，在收集了大量数据，而不清楚如何处理这些数据时，这种混乱情况就会发生。这反过来又将存储和处理数据的成本推到了顶峰。

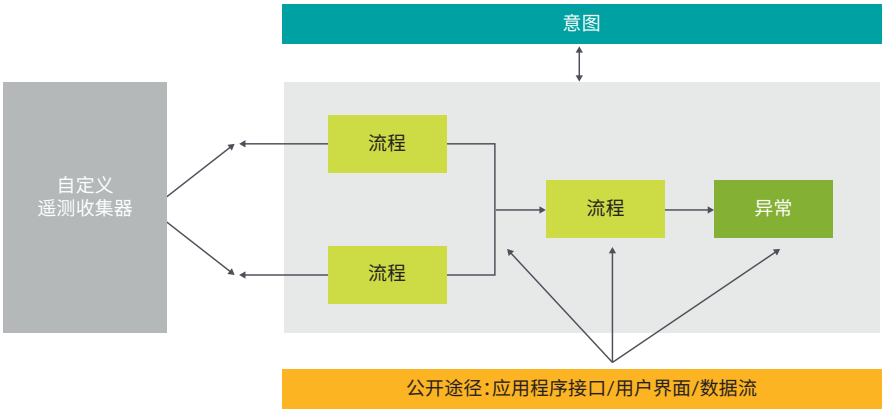

图 5：遥测收集

在配置探针后，每个阶段的输出都可以通过 API、GUI 和流式传输端点获得。Apstra 内置有一套探针，并且 GitHub 上也存在一个开源的探针库。用户还可以从头开始创建新探针。

根本原因识别

根本原因识别 (RCI) 是一种机制，其作用是自动分类和推导在基础架构中所识别的条件之间存在的因果关系。RCI 的主要好处是，从海量不可操作的条件中找出需要运维人员采取行动的根本原因条件，这些条件仅仅是根本原因故障导致的后果。例如，假设运维人员已经对基础架构进行了彻底检测，并在“单一管理平台”控制台上观察有关异常情况的日志（参见图 6）。红点表示分散在基础架构中的不同类型的情况，这些情况发生在许多不同的要素中。

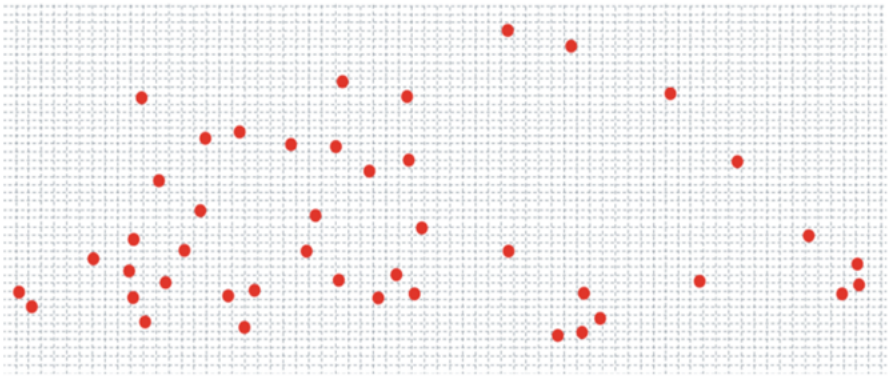

图 6：单一管理平台控制台

这张图片的问题在于，它不具有可操作性，只是一堆“噪音”而已。在没有分类的情况下，每个条件都在试图获得运维人员的注意力，可是，需要对哪一个条件采取行动呢?这就是 RCI 发挥作用的地方了。有了上下文知识（说明服务是什么，如何实施、如何映射到实施机制之中，以及经托管的基础架构中的要素如何产生关联），RCI 就可以识别根本原因以及相关症状和影响并加以分类。如图 7 所示，使用 RCI 进行分析的结果为：

- 根本原因是名为“主干 _1 ”的交换机上的内存泄露（根本原因：大红点）。
- 该内存泄漏情况导致内存不足进程终结程序会作用于并杀掉一些进程，其中一个是路由进程（症状：深蓝色点）。
- 而此情况进而导致该设备和对等设备上的 BGP 会话错误，同时丢失预期的路由表条目（症状：深蓝色点）。
- 因此，属于客户 X 和 Y 的端点遇到了连接问题（影响：浅蓝点）。

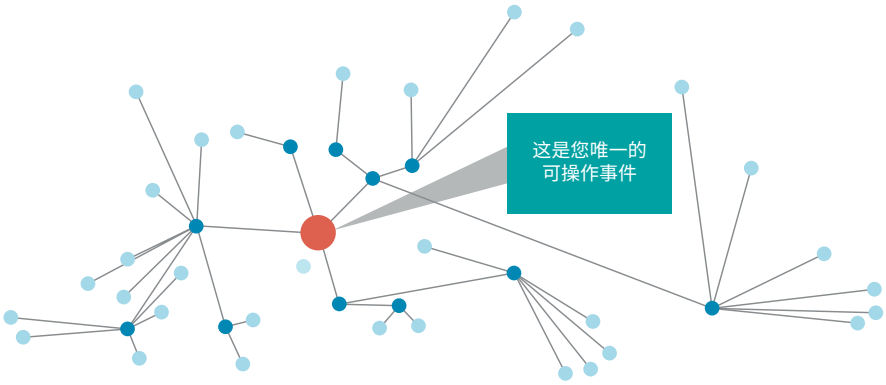

图 7：根本原因识别控制台

RCI 让“单一管理平台”中所涉及的复杂人工分析过程实现自动化，这些过程会使运维人员承受大量信息带来的负担，并将执行视觉发现和进行关联的操作留给了他们。相反，RCI 为运维人员提供了一个简单的管理平台，仅识别需要进行的操作。

上下文模型

拥有所有数据，并不意味着您拥有所有答案。若是收集大量数据而不自动提取知识，这会导致运营支出激增，因为您的专家将会在理解数据上花费宝贵时间。数据本身的价值是有限的，除非它能为您提供知识，或提供相应问题的答案。

原始遥测数据通常存储在键值存储项中，非常适合水平缩放和分片。然而，除了简单的键查找外，它们并不支持其他查询方式。要提取知识，您需要构建对查询的支持，并具有构造这些查询的上下文。

另一方面，SQL 数据存储确实支持复杂查询。存在的问题是，SQL 表是围绕预期查询所设计的，而预期查询会在设计期间创建。如果您希望在运行时询问一些其他问题，您可能要对数据库进行反规范化，若非如此，查询可能最终会涉及大量的连接项，这将导致性能降低到停滞状态。然而，操作知识的提取所关系到的都是运行时出现的查询。这就是基于图表的上下文数据实施的亮眼之处：其支持在运行时带有大量“连接项”的任意复杂查询。通过仅添加更多基本构建块、节点和关系的实例，还可以将图表模型进行轻松扩展。

接下来让我们更深入地了解一下这个上下文中的内容，因为这将有助于解释查询所实现的效果。

- **设计成果物**：首先，意图图表上下文中包含设计成果物。例如，如果您正在设计某个数据中心网络，它可能包含所需的超额预订因素、所需的服务器数量，以及所需的高可用性 (HA) 配置（单连接、双连接）。
- **资源分配**：这其中包含与资源分配相关的决定。它决定了您是将基础架构作为**“牛群”(cattle) 还是“宠物”(pets)** 来管理。
- **隔离策略**：这也将指定隔离策略。某些资源是否允许重复使用（IP、ASN、VLAN）？
- **分段策略**：其中包含分段策略，其将指定哪些端点和工作负载可以相互通信，以及可以在什么样的条件下这样做。
- **实施信息**：这包含有关在何处（在什么物理或虚拟设备上）以及如何（使用什么机制）实际实施分段，以及可达性策略的实施信息。
- **外部可达性策略**：在数据中心的情形下，这可能还包含外部可达性策略，其将指定在外部可以看到哪些位于内部的端点（反之亦然）。它还含有哪些租户负责哪些分段、向每个租户承诺的服务级别，以及服务级别目标的信息。
- **业务视角**：这会涵盖业务视角上的内容（例如服务级别协议），以及在无法满足这些协议时将会付出哪些代价。

8-D 视图

尽管所有这些成果物都会表示在单个图表中，但该单一真实来源在逻辑上是一个单一的多维区域，其至少容纳了以下八个维度（该数字依此示例而定，其他情况下可能还具有更多维度）：

- 设计
- 资源
- 隔离
- 分段
- 实施
- 外部可达性
- 服务级别
- 业务视角

因此，当出现问题时，您可以在此 8-D 视图中进行独立查询，并获取复杂问题的答案，例如：“在给定某要素上最近的故障条件时，我设计的目标是否仍然有效?” “是否在正确的实施点上使用了正确的资源和所需的隔离策略?” “它是否能以任何方式对分段策略造成影响?” “它是否能满足服务级别目标?” “我会为这次故障付出什么代价?”

这种 8-D 视图的存在并非是“值得拥有”的，它是实现可靠运行的前提条件。8-D 视图内置在了 Apstra 基于意图的系统中。若是没有 Apstra，则您必须使用一个非常复杂的层次来重构此 8-D 视图，该视图将涵盖存在于您的专家脑海中的真实信息来源，以及和部落知识的多个来源。在构建这一层的环境时，每个单独的事实来源都没有在考虑集成的情况下构建，并且它们具有不同的语义和行为，在最好的情形下，这会是一项无差别的繁重工作；而在最坏的情形下，这将是一场难以掌控的恶梦。

实时监控和通知

Apstra 关系到提取有关您的基础架构的意图以及最终操作状态的知识。如果这种知识是及时的（即如果它能反映当前的状况），那么这样的知识就是“力量”。Apstra 的核心是配置实时查询的能力，这能使客户订阅关注条件，并在条件满足时实时收到通知。此上下文中的条件与意图和操作状态相关。这是可靠地应对变化的绝对前提。

正如您将在架构概览部分中看到的，在用户级别所呈现的相同发布/订阅范式由等效的低级别发布/订阅机制提供支持。该机制由分布式数据存储实施，而分布式数据存储会充当 Apstra 系统进程的以数据为中心的逻辑通信通道，这些进程可实现应用逻辑。

自我运维

最后但并非最不重要的是，需要自动对某些事件进行反应，旨在修复问题、记录问题以进行取证分析，或者进行下一级别的深入分析以收集有助于分析根本原因的遥测数据。

自动化反应：

- 能够降低风险，原因是补救参数是从最新单一真实来源自动获得的，且不受错误配置或陈旧数据所影响
- 由于能及时进行补救，因此可改善客户体验
- 降低运维成本，因为这样就无需手动排除故障和手动执行补救方案

由于所需的功能已经存在，自我运维网络的障碍更多源于来自组织或个人的阻力，而非技术上的限制。

最终，自动执行反应可使网络能够进行自我运维和自我修复。由于所需的功能已经存在，自我运维网络的障碍更多源于来自组织或个人的阻力，而非技术上的限制。

可扩展性：让数据中心网络顺应未来

可扩展性包含三个维度：

1. **参考设计可扩展性**。这决定了为实现目标，基础架构的各个部分将如何协同工作。它包含有允许修改图表模型的插件，以及与如何将意图映射到基础架构中的实施机制相关的修改。
2. **分析可扩展性**。这与需要监视的新条件或情况的定义有关，并允许用户定义新的验证，以及如何对它们进行分类和关联。
3. **灵活的服务 API**。它们提供了扩展顶级服务定义的能力。

参考设计

正如此前所提到的，参考设计是一种行为契约，它能够定义如何将意图映射到实施机制，以及必须满足哪些期望才能认为意图已得到实现。本契约的任何方面都可修改，或进行延期。新节点和关系类型均可定义，并且可更改配置模板以及资源分配机制。

分析可扩展性

可通过两种机制引入新分析功能：

1. 可以定义**新 IBA 探针**来检测新关注条件。它们可能包括新的遥测采集器，以及特定条件的数据处理通道。也可以发布探针，随后从公共存储库导入。
2. 可定义**新 RCI 模型**并将其加载到系统之中。RCI 模型本质上是一种新型根本原因与其产生的一组症状之间的映射。在完成定义并加载此模型后，Apstra 系统将根据观察到的症状来自动识别根本原因。

服务 API

Apstra 以基于组的策略的形式提供服务级别 API，它将提供以与实施无关的方式为广泛服务和策略提供支持的灵活性。

在使用基于组的策略时，意图将以一个图表的形式展示，其会表示放置在组（成员关系）中的端点，旨在表达某些常见行为的意图。策略会被实例化并与组或单个端点相关联，以定义该行为。

策略能够以定向（“从/到”关系）或非定向（应用到）的方式与组相关联。策略是规则的集合。当规则之间的排序很重要时，规则可以与下一个规则具有关系。组可以由其他组构成（层次结构）。组还可以与其他组具有关系，以表达某些约束条件（例如：“这些端点/组位于这些端口组之后”）。可以将端点、组、策略和规则视为用于表达连接意图的构建块。

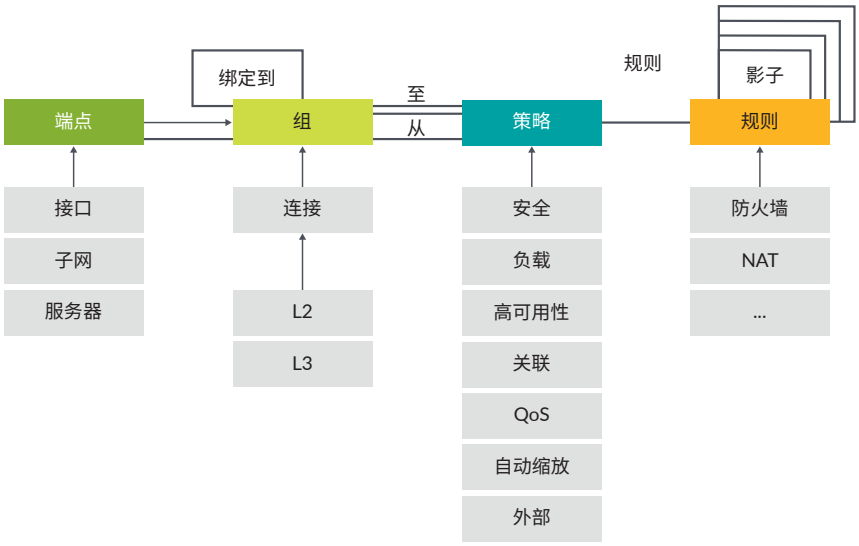

图 8：基于组的策略

端点是您的基础架构中受策略所约束的各项要素，因此，这在其构造中十分常见。它们可以表示接口（物理或虚拟的）、服务器、虚拟机 (VM)、容器或应用端点。图 8 中显示的蓝色箭头表示“逻辑”继承。端点包含更精确地定义它们的参数（例如，接口名称、端口号、序列号和主机名、VM UUID、容器 IP、应用协议、UDP/TCP 端口号）。端点还可以表示不由 Apstra 管理的要素（外部端点），并且因此可用于表达需要与 Apstra 需要交互的外部系统（例如，外部路由器 IP/ASN）的约束。

端点会被放置到组中。任何给定的端点都可以是多个组的成员，每个组会表示预期行为的不同方面。例如，组可以表示：

- “东部”地区的一组服务器
- 用于负载平衡的一组服务器
- 构成冗余组的一组服务器

组也可以具有过载的语义。例如，作为 L2 域组成员的端点是 L2 广播域（子网）的成员。与之相似，作为 L3 域的成员的端点具有 L3 可达性。

组也可以由其他组构成。组的端点成员可以被显式地实例化，或者可以存在作为组定义一部分的动态成员资格规范，按照组定义，组中存在的规范中隐含端点。

例如，L3 域可以具有无类域间路由 (CIDR) 或子网作为其属性；因此，IP 位于该范围/子网中的所有端点都是该组的隐式成员。在此情形下，“L3 到服务器”参考设计将指定许多外部/内部子网。即使在 Apstra 中没有明确指定和管理端点（容器），也意味着容器确实属于指定的 L3 域。然而，为了对粒度分段进行建模，我们还将引入容器端点，及其托管的服务器的显式规范。

策略可定义通用行为，并且可以包含用于精确定义该行为的参数。策略可以按非定向方式应用到组，也可以在需要时指示方向（例如在安全策略中实现）。特定策略的示例包括安全性、负载平衡、HA、关联性（例如，对于共置端点或分布端点的期望）以及服务质量 (QoS)。

策略可在有需要时包含规则。规则通常要遵循“操作遵循的条件”的模式。例如，在安全策略中，“match”（匹配）是一个条件语句，其行动是“allow/deny/log”（允许/拒绝/记录）。

可扩展性：无痛发展

Apstra 支持网络透明的分布式状态访问和管理，而并行执行则由单独的进程提供支持。实时执行由事件驱动的异步执行模型和实时执行调度共同提供支持。通过 C++ 作为中间语言进行编译，以实现机器级别的效率，从而为效率和可预测性提供支持。扩展有三个维度。

扩展状态

首个维度是扩展状态。Apstra 数据存储通过添加更多高可用性服务器来进行水平扩展。意图和遥测数据存储是相互独立的，并且可以根据需要进行独立扩展。同时也存在一项针对分层数据存储的规定，其中，顶层数据存储仅订阅（与之同步）下一级数据存储中所需的状态子集，而这些状态子集是由设计人员指定的，这样的订阅是为了在不同数据存储之间实现状态关联。

扩展处理

第二个维度是扩展处理。Apstra 可以在有要求时启动处理代理的多个副本（按代理类型），以分担处理负载。可以通过添加更多服务器来托管代理，从而添加更多代理。这样还能够管理代理的生命周期。

系统的基于状态的发布/订阅架构能使代理对定义良好的状态子集作出反应（提供应用逻辑）。完整意图的覆盖范围是通过被委托处理不同状态子集的独立代理来完成的。而这意味着当意图或操作状态发生变化时，代理的反应是“增量变化”，并且与整个状态的体量无关。

Apstra 会采用传统方法来处理规模和与之相关的复杂性，那就是：分解。“每个人都知道一切”的方法是无法扩展的。您必须分发有关所需状态的知识，并让每个代理确定该如何达到该状态，以避免进行集中化决策的需要。Apstra 对实时图标查询的支持意味着，客户端（如 UI）可以准确地请求他们想要的内容，并准确地获得他们需要的内容（仅此而已），从而允许对从后端获取的数据量进行粒度控制。

扩展网络流量

第三个维度是扩展网络流量。代理和数据存储之间的通信会使用经优化的二进制通道，因此，相较于基于文本的协议，这能显著减少通信量。

容错是通过将 Apstra 应用作为多个进程执行来实现的，这些进程可能在通过网络连接的单独硬件设备上运行，并将状态与处理分离，以支持复制状态和状态的快速恢复。

Apstra 架构概览

在本文的第一部分，我们讨论了数据中心网络架构所面临的一些挑战，以及 Apstra 将如何解决这些问题。我们在这里将深入研究 Apstra 架构的细节。

Apstra 是以分布式状态管理基础设施为基础的，而该基础设施可以描述为：具有水平可扩展和容错内存数据存储的、以数据为中心的通信结构。特定参考设计应用的所有功能都通过一组无状态代理实现。代理通过基于发布 - 订阅的逻辑通信通道进行相互通信，并在本质上实现应用逻辑。

如上所述，每个 Apstra 参考设计应用只是无状态代理的集合。从广义上讲，存在三类代理：

1. **交互 (Web) 代理**负责与用户交互，即，获取用户输入，并向用户提供来自数据存储的相关上下文。
2. **应用代理**负责通过订阅输入实体和生成输出实体来执行特定于应用域的数据转换。
3. **设备代理**驻留在受托管的物理或虚拟系统（如交换机、服务器、防火墙、负载平衡器或者控制器）上（或作为其代理）。它们用于通过特定于设备的本机接口写入配置和收集遥测数据。

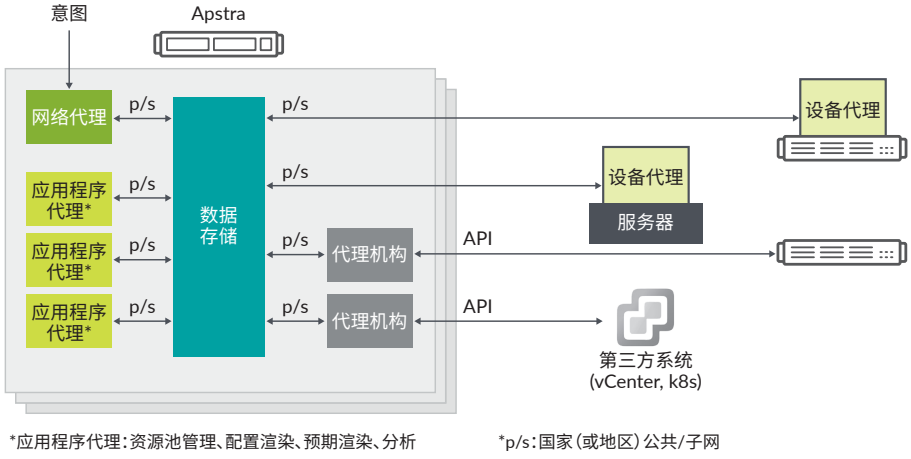

图 9：Apstra 代理

这种交互可以通过一个示例来说明，该示例描述 Apstra 的数据中心网络参考设计应用的一部分。

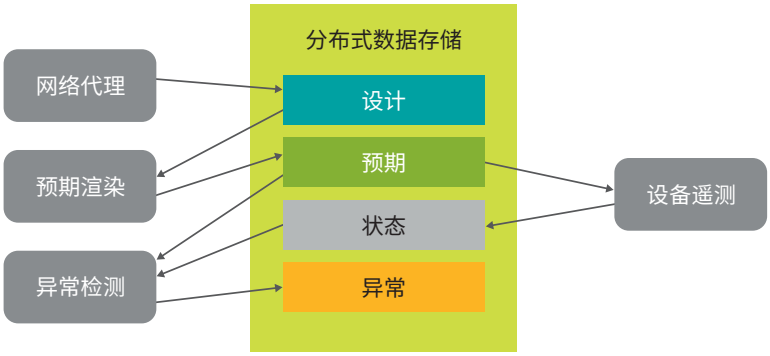

图 10：Apstra 基于意图的系统数据中心网络参考设计应用

Web 代理可接受用户输入，而在本例中，是指 L3 Clos 交换矩阵的设计，其中包含主干、叶和主干与叶之间的链路的数量，以及用于交换矩阵 IP 和抽象语法标记 (ASN) 编号的资源池。Web 代理将此意图作为一组图表节点和关系及其各自的属性发布到数据存储中。

构建代理会订阅此意图，并且：

- 实施正确性和完整性验证
- 从资源池分配资源

然后，假设验证通过，构建代理将该意图与资源分配一起发布到数据存储中：

1. 配置将呈现代理订阅构建代理的输出。
2. 对于每个节点而言，配置代理将获取相关数据（包括资源在内），并将其与配置模板合并。
3. 预期代理还会订阅构建代理的输出，并生成为了验证结果而需要满足的预期。
4. 设备遥测代理会订阅预期代理的输出，并开始收集相关遥测数据。
5. IBA 探针负责处理原始遥测数据，并将其与预期进行比较，然后发布异常。
6. RCI 代理将分析异常，并将其分类为症状、影响和已确认的根本原因。

代理通过发布实体和订阅实体中的更改，通过基于属性的接口（因此称为以数据为中心）进行通信。以数据为中心还意味着，数据定义属于是框架的一部分，并通过定义实体来实现（例如与基于消息的系统相反）。

以数据为中心的发布 - 订阅系统部受基于消息的系统的问题所影响。在基于消息的系统中，消息的数量早晚会超过系统存储或使用这些消息的能力。处理此问题很困难，原因是必须重放消息的历史记录才能达到一致的状态。

以数据为中心的系统对状态变化的激增具有弹性，因为它基本上只依赖于最后的状态。这种状态将获取重要的上下文，并抽象出导致它的所有可能的（和不相关的）事件序列。使用状态机范例编写的代码更易于阅读、维护和调试。

困难问题（例如，弹性和容错）代表所有代理，并可一次性解决。然后，典型的体系结构将由许多无状态代理组成，这些代理可以在发生故障时重新启动，代理只需从系统数据库 (SysDB) 中重新读取它们所订阅的状态，即可从中断的位置恢复。

Apstra 架构的优势

正如我们在本文中所描述的，Apstra 架构提供了显著的优势，解决了一些数据中心网络所面临的最困难的问题。架构能够：

- **帮助运维人员可靠地处理变化。**这可以通过实时可查询的意图和操作上下文来实现。
- **简化网络服务生命周期的所有方面，**其中包括第 0 天、第 1 天和第 2 天的操作。并且简单性还能降低操作员出错的可能性。
- **通过有状态编排降低操作风险，**该编排会利用前置条件验证、后置条件验证、自动配置呈现和自动预期验证。

根本原因识别和基于意图的分析的特殊优势

Apstra 的运营分析组件（根本原因识别和基于意图的分析）使您能够：

- 通过简化的操作分析**减少平均维修时间和主题专家 (SME) 工作负载，**从而让技能最娴熟的资源可腾出时间来进行改进和创新，而非救火。
- 通过收集和存储更少的数据来**提取更多知识。**在参考设计行为契约的支持下，Apstra 知道它在寻找什么，并只会收集这些信息。这种即时处理可以使存储需求和非关注数据的后处理工作减少五到六个数量级。这使您能够更高效地运行基础架构，并在管控成本的同时保持竞争力。
- **快速、轻松地了解问题。**Apstra 在“简单管理平台”中可识别重要的可操作事件，并消除某些症状的噪音，这些症状仅仅是已确定的根本原因的假象。
- **将复杂的工作流程自动化。**Apstra 系统允许您自动执行上下文充足的故障排除工作流程（若非如此，这些工作流程将非常繁琐、效率低下、耗时且成本高昂）。
- **实现零接触维护。**由于它与意图始终保持同步，因此 Apstra 能够在发生变化时实现零接触和零成本的维护。因此，它富有弹性，可自动响应变化，并节省了与维护数据处理通道相关的巨大成本。
- **消除昂贵的自行 (DIY) 开发工作。**数据处理通道集成工作的 DIY 开发成本高昂且脆弱，将占用您核心业务的时间和资源，并且需要大量的 SME 繁重工作。
- 与机器学习/人工智能方法相比，可**提供高度准确的结果。**
- **采用与供应商无关的方法，**让您在最佳供应商和功能中进行自由选择。
- 在您的公共/私有/混合云基础架构中**使用通用 API。**

从第 1 天开始扩展：成功案例

Apstra 从第 1 天起就考虑到了扩展。请考虑一下这家客户——其使用 Apstra 进行了广泛的验证，并取得了令人印象深刻的结果：

- **物理基础设施：**6,000 次接口状态验证；1,000 次布线验证；36,000 个错误计数器；10,000 次功率、温度、电压指标验证
- **L2/L2 数据平面：**12,000 个队列丢弃计数器，数百个 MLAG 的健全性，3,000 次生成树协议 (STP) 验证；状态更改通知
- **控制平面：**1,500 个 BGP 会话运行状况，约 500 个预期的下一跳/默认路由
- **容量计划：**约 500 个趋势分析，为路由表的使用配置了阈值；地址解析协议 (ARP) 表；每个虚拟路由和转发 (VRF) 的组播表；6,000 次链路使用验证
- **符合性：**确保在所有 100 台左右的交换机上运行预期的操作系统版本
- **组播：**预期物理接口模块 (PIM) 邻居的 2,500 次验证；会合点检查 300 次；关于在会合点上的源、组、源-组对的计数中检测异常模式的 25 个验证

从本质上讲，向客户提供的不是包含 82,000 个条目的单一平台，而是一个仅显示异常并根据客户的规范归为仪表板类的简单平台，其中：

- 100% 准确（非统计推断）
- 它们始终与客户的拓扑和意图保持同步，因此无需持续维护
- 提供相关的可操作上下文（偏离的内容、偏离的性质、期望的状态）
- 在存储和处理需求上实现节省（仅需 9 GB RAM，96 GB 磁盘）
- 通过编写复杂而精细的数据处理通道，消除了内部锁定和遗留问题所带来的危险

总结

无法在 IT 基础架构中进行可靠的更改是增长和创新的主要障碍。Apstra 消除了这种担忧，并使永远根除可怕的“遗留基础架构”成为可能，使您能够可靠地进行变化，从而使您能够可靠地创新，并保持竞争力。

关于瞻博网络

瞻博网络将简单性融入到全球互联的产品、解决方案和服务之中。通过工程创新，我们消除了云时代网络的限制和复杂性，可应对我们的客户和合作伙伴日常面临的严苛挑战。在瞻博网络，我们坚信，网络是分享知识和实现人类进步的资源，它将改变这个世界。我们致力于开创具有突破性的方式，提供自动化、可扩展且安全的网络，以满足业务发展的需求。

公司和销售总部
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
电话：888.JUNIPER (888.586.4737)
或 +1.408.745.2000
www.juniper.net

亚太地区及欧洲、中东和非洲地区总部
Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
电话：+31.0.207.125.700

JUNIPER NETWORKS | Engineering Simplicity