**基于分布式数据库的网络控制技术研究**

摘要：

1. **绪论**
2. **研究背景与意义**

近年来，随着科技的发展和社会的进步，互联网在人们的生活中变得越来越重要，随着线上支付、线上教育、线上办公等浪潮的兴起，互联网开始在人们的生活中扮演着不可或缺的角色。由于地面网络受地形条件的影响较大，只靠地面网络难以实现万物互联的世界，因此天基网络应运而生。天基网络作为由上万颗的卫星搭建而成的重要信息基础设施，在此基础上需要设计比较完备的控制系统。

软件定义网络（SDN，Software Defined Network）作为一种新型的网络架构，与传统网络相比，能够实现控制平面和数据转发平面的分离，并提供统一的应用编程接口。在SDN平台上，易于实现网络实时流量信息收集，以及全网络拓扑内的流量调度，因此在网络控制与管理方面极大地优于传统网络。

随着网络规模的日益增长，单台控制器受到自身性能的限制，很难满足大型网络管控的应用需求。因此，常见的做法就是对网络进行拆分，把大型网络划分为多个区域，通过多个控制器的协作实现全网络的管控。为了保证SDN控制器集中式决策带来的负载均衡、路由优化等优势，我们需要完成不同分区的控制器之间的数据同步，实现对卫星设备及业务的统一管控。

1. **国内外研究现状**

近年来，随着SDN和网络功能虚拟化（NFV，Network Function Virtualization）技术在数据中心网[1]、无线接入网[2]和光网络[3]等领域的应用，卫星网络框架设计的思路得到了极大的拓展。在[4]中，作者提出了一种软件定义的卫星网络体系结构，针对目前建造空间信息系统的弊端给出了解决方案，同时介绍了在卫星网络中应用SDN的优点，并指出了一些待解决的问题。在[5]中，作者介绍了一种基于聚类的多级多域架构部署方案，应用分布式路由架构合理地进行负载分配和流量调度，解决了大规模网络负载均衡问题。在[6]中，作者提出一种有效的分层SDN模型，使得根控制器管理域间流量，而从属控制器管理域内流量从而降低了根控制器的负载。

众所周知，卫星带宽资源稀缺，同时由于卫星保持高速移动，具有网络拓扑变化快的特点。针对卫星节点之间的路由问题和资源分配问题，目前已有许多成果。在[7]中，作者提出了一种基于负载均衡的低复杂度路由算法，每个节点都会将其拥塞信息告知其相邻节点，以便数据包可以根据链路状态动态选择下一跳，从而缩短平均队列延迟，降低丢包率。在[8]中，作者提出了一种软件定义路由算法，采用虚拟拓扑方法来生成拓扑快照，以克服空间信息网络中拓扑的动态变化。在[9]中，作者提出了一种基于意图的路由算法和一种基于马尔可夫理论的带宽资源分配算法，结合卫星逻辑位置和数据包内容质量，降低了端到端时延，提高了业务请求成功率。

另外，随着互联网技术的不断升级，云计算、大数据等技术的大规模应用，分布式数据库的建设与应用已成为一种必然趋势。为了保证分布式数据库系统的可用性和一致性，设计者需要采用一致性协议来实现数据同步。在[10]中，作者详细介绍了经典的分布式一致性协议以及在目前常见的几种分布式数据库系统中一致性协议的应用，并从读写操作、节点类型与网络通信等方面进行了对比分析。在[11]中，作者结合Raft算法的思想，设计并实现了基于时间戳的分布式选举模块、自动化的集群切换模块和基于QUORUM策略的日志强同步模块。

1. **本文的研究方法**

SDN集中式控制器在应对大规模网络应用场景时，会受到单个SDN控制器的性能瓶颈限制。通过将大规模的SDN网络划分为多个小区域，采用SDN分布式控制器可以有效减轻每个域的SDN控制器的通信负载，从而提高整个网络的传输效率。因此，分布式控制器之间的协作需要得到设计保障。

SDN分布式控制器之间一般是通过东西向接口进行信息交互的，我们可以通过SDN分布式控制器和分布式数据库(DDB，Distributed Database)的有效结合，利用分布式数据库实现数据同步，从而完成不同区域的SDN分布式控制器之间的信息交互，进而实现对卫星设备及业务的统一管控。

1. **小结**

本章主要介绍了本课题的研究背景与意义，同时简单介绍了本文的研究方法。在接下来的几章中，我们将介绍基于分布式数据库的网络控制技术的设计与实现，并且介绍实验环境如何搭建，以及相关的测试分析。每一章的内容安排如下：

1. 首先介绍SDN的相关框架结构，然后介绍底层的openflow协议，为后续的网络控制技术设计提供理论基础。
2. 首先介绍本课题的设计目标，然后根据需求分析给出相应的方案设计，最后介绍如何实现一个原型系统。
3. 首先介绍如何搭建实验环境，然后进行网络连通性测试以及系统鲁棒性测试。
4. 通过对测试结果的系统分析，总结本文所提出的方案的优点和不足之处，进而对后续的工作方向进行展望。
5. **相关技术介绍**
6. **SDN相关介绍**

SDN（Software Defined Network）即软件定义网络，是近年来兴起的一种新型网络架构，其主要特点就是将控制平面与数据转发平面相分离。SDN框架如图2所示，主要由应用层、北向接口、控制层、南向接口、转发层五部分组成。

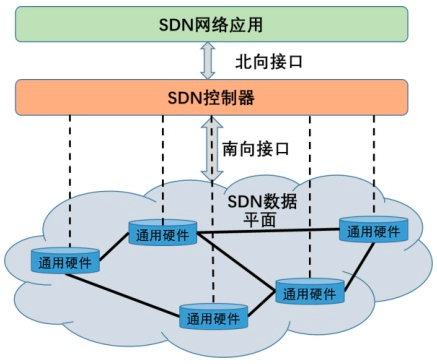


图2 SDN框架图

其中，应用层提供网络管理、网络安全、流量控制等服务和应用。控制层由控制器提供统一管理和控制，包括协议计算、策略下发、链路信息收集等功能。控制器中包含大量业务逻辑，以获取和维护不同类型的网络信息、状态详细信息、拓扑细节、统计详细信息等。转发层提供各种硬件设备进行数据转发，一般是支持openflow协议的硬件交换机，也可以是数据中心的一组网络交换机和路由器、

SDN架构与传统网络架构相比，其最大的优势就是控制平面和数据平面完全分离。在传统的网络中，控制平面和数据平面相互耦合，例如路由器不仅负责数据包的转发，还需要根据路由协议进行控制信息的计算。当网络规模逐渐增大时，分布式的路由协议很难实现全网信息的收敛，导致网络控制的难度大大提升。然而，SDN交换机仅仅具备数据转发功能，所有的链路发现、拓扑管理、路由计算、策略制定等都在SDN控制器上进行，然后通过OpenFlow协议下发流表（flow table）给SDN交换机，SDN交换机则根据流表来执行相应的行动（action），完成数据流的转发。因此，控制层可以实现集中式控制，转发层通过硬件实现高速转发。

北向接口是SDN应用程序和SDN控制器之间的接口，通常提供抽象的网络视图和网络行为的直接表达。通过控制器向上层应用开放的接口，业务应用能够方便地调用底层的网络资源和能力。南向接口是在SDN控制器和SDN数据路径（datapath）之间定义的接口，通常提供转发操作程序控制、统计报告、事件通知等功能。控制器通过南向接口的上行通道对底层交换设备的状态信息进行统一监控和统计，通过南向接口的下行通道进行策略制定、流表下发，实现网络控制。

SDN通过标准的南向接口屏蔽了底层物理转发设备的差异，实现了资源的虚拟化，同时开放了灵活的北向接口，供上层业务按需进行网络配置并调用网络资源，使用者可以基于 SDN 控制器进行个性化的开发以满足实际的业务需求，从而相比传统网络，SDN 网络具有良好的可编程与网络控制能力。

1. **OpenFlow相关介绍**

OpenFlow协议[1]是最知名的SDN南向接口协议，用于SDN控制器和底层交换设备之间的交互。作为一种网络通信协议，OpenFlow能够控制网络交换设备的转发平面，通过新增、修改与移除数据包控制规则与行动，来改变数据包的转发路径。SDN控制器可以利用支持OpenFlow协议的南向接口，更改推送到交换设备的流表，使得网络管理员能够实现精确的流量分区、流量控制，以获取网络最佳性能。

在OpenFlow协议下，控制器可以对底层网络交换设备的详细行动等进行综合管理，这种行为表现出了OpenFlow可编程的特性。为了本课题后续内容中需要实现的SDN应用程序开发，我们需要对OpenFlow控制器和OpenFlow交换机之间的报文交互有一定的了解。

1. 流表和流表项

在OpenFlow交换机中，流表负责描述何种数据包要进行何种处理等设置信息，以及执行这些处理的次数等记录。流表中包含的表项称为流表项，分别对各个流的行为进行定义。OpenFlow控制器通过在OpenFlow交换机中设置流表项来实现对网络流的各种控制。

流表项由以下3个要素构成：头字段、计数器、行动。流表项的头字段描述了何种数据包与流表项相匹配，计数器记录了匹配次数，行动则描述了OpenFlow交换机对于匹配的数据包采取的操作。

1. 控制器和交换机的基本行动

OpenFlow 1.0中定义的交换机必备行动有以下2种：转发数据包的Forward行动和丢弃数据包的Drop行动。

Forward行动可以通过指定输出目的地的虚拟端口，快速实现向指定端口发送数据包、向入端口以外的全部端口发送数据包（FLOOD）、将数据包封装并发送至控制器（CONTROLLER）等各种操作。

Drop行动能够丢弃与未指定Forward行动的流表项相匹配的数据包。

1. Packet-In消息

使用Packet-In消息可以将到达OpenFlow交换机的数据包发送至OpenFlow控制器，以下两种情况即可发送Packet-In消息：不存在与流表项一致的项目时（OFPR\_NO\_MATCH）；匹配的流表项中记载的行动为“发送到OpenFlow控制器”时（OFPR\_ACTION）。

Packet-In消息格式如下表1所示：

|  |  |  |
| --- | --- | --- |
| 字段 | 比特数 | 内容 |
| buffer\_id | 32 | 表示OpenFlow交换机中保存的数据包的缓存ID |
| total\_len | 16 | 帧的长度 |
| in\_port | 16 | 接收帧的端口 |
| reason | 8 | 发送Packet-In消息的原因。不存在匹配的流表项时，OFPP\_NO\_MATCH=0；向OpenFlow控制器的发送是通过行动指定的时，OFPP\_ACTION=1 |
| pad | 8 | 用于调整对齐的填充 |
| data | 任意大小 | 包含以太网帧的数据时使用的字段。以太网头的大小为14个8位字节，因此该字段从32比特字的中间位置开始，具有IP头的32位对齐的效果 |

表1 Packet-In消息格式

1. Packet-Out消息

使用Packet-Out消息可以从OpenFlow控制器向OpenFlow交换机发送包含数据包发送的命令。另外，使用Packet-Out消息还可以将OpenFlow控制器创建的数据包发送到OpenFlow交换机（OFPP\_CONTROLLER）。

Packet-Out消息格式如下表2所示：

|  |  |  |
| --- | --- | --- |
| 字段 | 比特数 | 内容 |
| buffer\_id | 32 | 表示OpenFlow交换机中保存的数据包的缓存ID |
| in\_port | 16 | 数据包的输入端口。用于表示OpenFlow交换机接收数据包时，输入端口的物理端口号。该信息未指定时，使用OFPP\_NONE；发送OpenFlow控制器创建的数据包时，使用OFPP\_CONTROLLER |
| actions\_len | 16 | 行动信息的长度，单位为8位字节 |

表2 Packet-Out消息格式

1. **Mininet相关介绍**

Mininet是一个轻量级的SDN研发和测试平台，支持Openflow、OpenvSwitch 等SDN部件，其主要特点如下：

1. 支持创建自定义拓扑。Mininet在单个Linux内核上运行一组终端主机、交换机、路由器和链接，它使用轻量级虚拟化使单个系统看起来像一个完整的网络，并运行相同的内核、系统和用户代码。
2. 支持运行真实的程序。Mininet主机的行为就像一台真实的机器，可以运行包括在底层Linux系统上安装的任何程序，例如：Web服务器、TCP窗口监视工具，以及Wireshark抓包工具。
3. 支持自定义数据包转发。在Mininet上运行的程序可以通过看似真实的以太网接口以给定的链接速度和延迟发送数据包。并且，在Mininet上运行的交换机可以使用OpenFlow协议进行编程。

Mininet作为一个网络仿真器，它的虚拟主机、交换机、链接和控制器都是真实的东西，它们只是使用软件而不是硬件创建的，并且在大多数情况下，它们的行为类似于离散的硬件元素。通常可以创建类似于硬件网络的Mininet网络，或者类似于Mininet网络的硬件网络，并可以在两个平台上运行相同的二进制代码和应用程序。因此，Mininet测试通过的程序拥有比较强的可移植性。

本课题的SDN实验测试平台将利用Mininet进行搭建。首先，在Mininet上创建一系列的OpenFlow交换机、主机和链接，组成数据平面。由于Mininet不支持OpenFlow控制器编程，我们需要另外选取合适的OpenFlow控制器进行开发。然后，通过Mininet实现OpenFlow控制器和数据平面之间的远程连接，进而完成相关的数据转发测试。

1. **路径规划算法介绍**

本课题中的业务路由计算可以通过归约到图论中的最短路问题进行求解。图论中典型的最短路算法有Dijkstra算法和Floyd算法。

Dijkstra算法是典型的单源最短路径算法，时间复杂度为O(N^2)。该算法寻找最短路径的过程可以看作是一个排序过程，不断在未加入最短路的节点中选取距离源节点最近的节点，直到所有节点都加入最短路。每次将一个节点加入最短路后，进行相应的“松弛”操作，更新其相邻节点到源节点的总距离。

Floyd算法是求解任意两点之间最短路径的典型算法，时间复杂度为O(N^3)。该算法的原理是动态规划，对于每一对顶点u和v，看看是否存在一个顶点w使得从u到w再到v比己知的路径更短，如果是则更新u到v之间的路径。

1. **本章小结**

本章首先简单介绍了SDN的框架结构，通过传统网络和SDN的对比分析，展示了SDN将控制平面和数据平面完全分离带来的好处。

其次介绍了SDN南向接口的OpenFlow协议，对流表、行动和相关报文字段加以解释说明，并以SDN控制器和交换机之间的交互为例，展现了OpenFlow协议可编程的特点，有助于后续的SDN控制器应用开发。

然后介绍了Mininet仿真测试平台，并为本课题的实验平台搭建提供了思路。

最后简单介绍了Dijkstra算法和Floyd算法，通过对比两种算法的优缺点，为后续的动态路径规划算法的设计和应用提供了理论基础。

1. **基于分布式数据库的网络控制技术的设计与实现**
2. **设计目标**

本课题的研究背景是天基网络的分布式控制技术，因此需要考虑卫星网络负载重、拓扑变化快等诸多现实因素，通过对SDN控制器、分布式数据库进行功能需求的分析，以确定最终的设计目标。

根据现有天基网络所存在的性能瓶颈，我们可以确定以下几项设计目标：轻量化、同步时延低、可拓展性、方便部署和管理。

1. 轻量化

天基网络拥有多达上万颗的卫星节点，由于网络负载非常重，为了尽可能降低开销，我们需要通过对现有的开源SDN网络控制器进行对比调研，选择使用轻量级的SDN控制器。

1. 同步时延低

为了降低分布式SDN网络控制器的决策冲突，需要向全部SDN控制器提供相同的网络状态信息。因此，我们需要设计SDN控制器之间的东西向接口，以支持分布式SDN网络控制器之间的信息交互。我们打算利用分布式数据库之间的数据同步功能，来实现SDN控制器之间的东西向接口。同时，为了降低同步开销，我们需要通过对现有的分布式数据库进行对比，选择使用同步开销小、同步时延低的轻量化分布式数据库。

1. 可拓展性

卫星节点成千上万，数量较多，导致网络状态变化较快，因此我们需要构建具有一定拓展性的分布式架构，以满足网络节点拓展的需求。

1. 方便部署和管理

相较于传统网络，SDN网络由于控制平面与数据转发平面分离，本身就具备

管理简单的优势。我们只需要关心SDN控制器的控制逻辑即可，位于数据转发平面的交换机并不具备计算路由的职责，只需要依靠SDN控制器设置的流表执行相应行动，从而完成交换机的数据转发功能。

1. **设计方案**

本课题的研究内容主要主要围绕SDN控制器、分布式数据库、逻辑拓扑结构和动态路径规划算法展开。基于以上设计目标，我们可以针对本课题提取出以下四个关键问题：

首先是SDN网络控制器和分布式数据库的选取。为了尽可能降低开销，我们需要选取轻量级的控制器和数据库。另外，数据库横向对比的具体指标包括：不同类型的一致性协议、同步时间、同步开销等方面。一致性协议分为单主协议和多主协议，其中单主协议要求所有写操作都由主节点处理并且同步给其他副本，多主协议要求所有写操作可以由不同节点发起并同步给其他副本。通过对分布式数据库和不同类型的一致性协议的对比，本课题拟选取同步时间短、同步开销小、采用多主协议的分布式数据库。

其次是逻辑拓扑的构建。一方面，基于天基卫星网络的特点，我们分析比较了集中式架构和分布式架构的优缺点。集中式架构不适合大规模网络的应用场景，而分布式架构在经济性、安全自主、灵活性、可伸缩性等方面有明显优势，因此我们选择构建更具拓展性的分布式架构，作为SDN控制器的逻辑拓扑结构。另一方面，SDN控制器互联方式可以分为层次控制方式和扁平控制方式，我们可以通过比较分层互联和同层互联的优缺点，选择合适的SDN控制器互联方式。层次控制方式采用分层互联架构，只有根控制器维护全局网络视图，跨区转发的控制算法相对比较复杂；扁平控制方式采用同层互联架构，每个SDN控制器都维护全局网络视图，独立决策，同步负载相对较大。

然后是数据同步技术。由于控制器是根据数据库的数据进行路由决策的，数据库的一致性就显得尤为重要，在同样的数据之下，才能使控制器把控好整个网络的资源情况和业务分布，进而采取更优的决策。不同分区采用不同的分布式数据库，不同分区之间的SDN控制器通过分布式数据库来实现相关网络状态信息的同步，包括逻辑拓扑结构、业务路由信息等，从而保证分布式SDN控制器决策的一致性，降低各个分布式SDN控制器决策时的产生资源冲突和网络拥塞等问题的概率。

最后是动态路径规划算法。控制器一般需要控制多条业务流的传输，如果多条业务流同时使用同一条链路进行传输，将会导致链路业务繁重、鲁棒性差。因此，为了保证业务服务质量，我们需要根据卫星网络特点和业务状态，研究合适的网络流量分布优化方法，利用动态路径规划算法实现网络流量的均衡分布，避免网络拥塞，提高业务下发成功率。

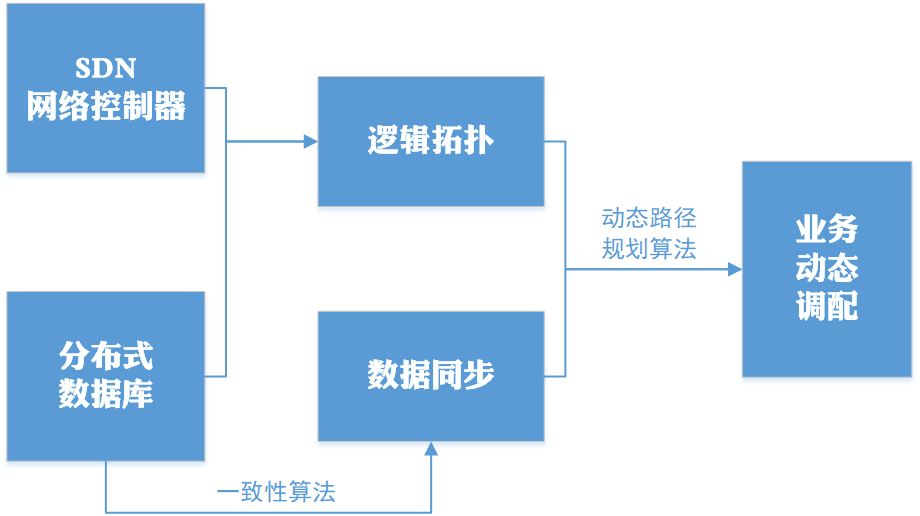


图1 技术路线

本课题的技术路线如图1所示。首先，通过调研对比选取轻量化、开销小的数据库和SDN开源控制器作为项目实体，并且根据天基网络需求，构建合适的分布式网络控制器逻辑拓扑。然后，采用基于分布式数据库的数据同步技术实现网络状态信息的同步。最后，采用基于流量优化的动态路径规划算法，解决卫星网络的路由问题，实现业务动态调配。

1. **具体实现**
2. SDN控制器的选取

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 控制器 | 开发语言 | 多线程 | Rest API | OpenStack | 南向接口协议 |
| NOX | C++/Python | 不支持 | 不提供 | 不支持 | OpenFlow |
| POX | Python | 支持 | 不提供 | 不支持 | OpenFlow |
| Mul | C | 支持 | 提供 | 支持 | OpenFlow/Netconf |
| Ryu | Python | 支持 | 提供 | 支持 | OpenFlow/Netconf/  OF-config/OVSDB |
| Floodlight | Java | 支持 | 提供 | 支持 | OpenFlow |
| OpenDaylight | Java | 支持 | 提供 | 支持 | OpenFlow/Netconf/  BGP/PCEP/OVSDB |
| Beacon | Java | 支持 | 提供 | 支持 | OpenFlow |

表3 主流开源控制器对比

如表3所示，通过对主流开源控制器的对比，我们可以发现以下结论：

①目前控制器的主要编程语言分别是C/C++，Java和Python。其中，基于C/C++的控制器在处理性能上有较好的表现；基于Java的控制器有较为丰富的API，便于业务的扩展；基于Python的控制器在网络编程方面有较好的灵活性，易于开发，但是效率较低。

②除了最初的NOX控制器不支持多线程之外，后来的主流控制器均支持多线程技术，使得控制器的响应速度更快，可以对上层的不同业务进行优先配置。

③早期的NOX控制器、POX控制器不支持OpenStack，其余控制器均支持OpenStack云管理平台。SDN和OpenStack的结合可以更好地对资源进行集中分配调度，为云数据中心降低了维护成本。

④早期的NOX控制器、POX控制器和Floodlight控制器均只支持单一的OpenFlow协议，而对其他的南向接口协议并不适配，造成了实际部署的困难，增加了运营成本。后来的控制器支持OpenFlow/Netconf/BGP/PCEP/OVSDB等多种南向接口协议，组网更加灵活。

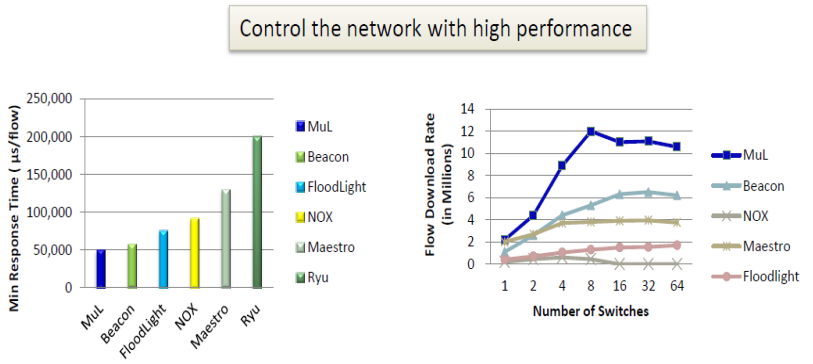


图2 SDN控制器最小响应速度和流下载速率比较

如图2所示，相较于一般主流SDN开源控制器，MuL控制器由于采用C语言编程，在响应速度和流下载速率吞吐量方面都拥有更好的表现，拥有高性能和高可靠性。基于MuL控制器的轻量化和高效性，本项目拟选用MuL控制器。

1. 分布式数据库的选取

相较于传统数据库，分布式数据库的优势主要有以下几点：具有灵活的体系结构以及优越的可拓展性；系统的可靠性高、可用性好；适应分布式的管控机构；局部应用的响应速度快。

本课题拟采用分布式键值数据库进行数据存储和管理，因此选取DB-Engines排名前15的主流键值数据库进行对比，如下图3所示：

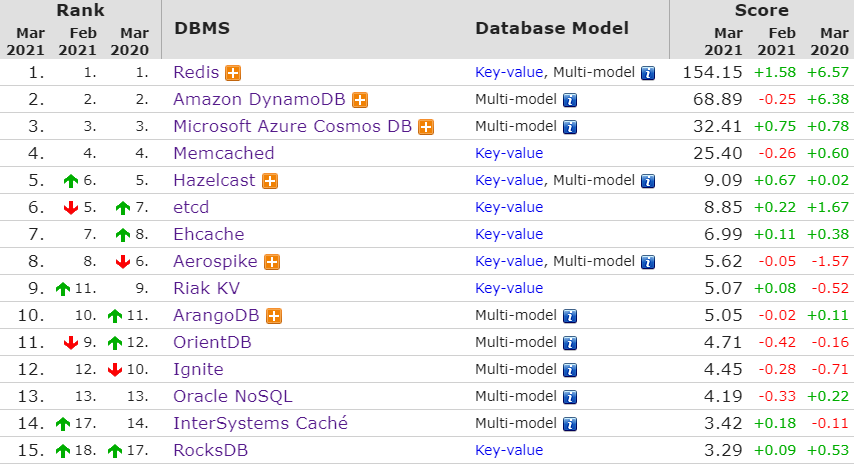


图3 主流键值数据库排名

基于之前对分布式数据库的需求分析，我们关注的主要指标有两个：分别是实现语言和许可证。①实现语言：为了保证数据库底层程序语言的轻量级，实现语言选择C、Go等语言，排除采用Java、Python语言实现的分布式数据库。②许可证：为了便于项目的后续开发，选择开放源码的分布式数据库。

经过初步筛选后，我们选出了Redis、etcd、ArangoDB、Aerospike这四款轻量级开源分布式键值数据库进行详细对比，如下表4所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | Redis | etcd | ArangoDB | Aerospike |
| 描述 | 内存数据结构储存器，用作数据库、缓存和消息代理（专注于性能，大部分决策优先考虑高性能和低延迟） | 可靠的分布式键值存储器 | 原生的多模型数据库管理系统，用于图形、文档、键/值和搜索。全部在一个引擎中，可以用一种查询语言访问 | 闪存优化内存NoSQL数据库 |
| 最新版本 | 6.2.0  2021年2月 | 3.4  2019年8月 | 3.7.9  2021年3月 | 5.5.0.2  2021年2月 |
| 实现语言 | C | Go | C++ | C |
| 许可证 | 开源 | 开源 | 开源 | 开源 |
| 服务器  操作系统 | Linux、OS X、  Windows | Linux、  Windows | Linux、OS X、  Windows | Linux |
| 数据一致性 | 最终一致性 | 强一致性 | 强一致性，对每个集合或每个写操作可配置最终一致性 | 本地集群配置强一致性，跨数据中心配置最终一致性 |
| 数据并发操作 | 支持（数据访问由服务器序列化） | 支持 | 支持 | 支持 |
| 数据持久化 | 支持（通过快照/操作日志持久化） | 支持 | 支持 | 支持 |

表4 轻量级开源分布式键值数据库对比

根据以上相关指标的对比，可以发现以上数据库主要在数据一致性上有所差异。在分布式数据库中，会涉及到CAP原理来保证数据的一致性。CAP分别指：一致性（Consistency）、可用性（Availability）和分区容忍性（Partition tolerance），CAP原理指出：分布式系统中的CAP三要素只能同时满足其中两点，不能同时兼顾三点。由于分区容忍性是分布数数据库最基本的要求，因此分布式数据库的设计必须在一致性和可用性之间做出平衡和取舍。

最终一致性和强一致性都是分布式计算里的一种内存一致性模型。最终一致性指对于已更新的数据的读取，最终都能获取已更新的数据，但不完全保证能立即获取已更新的数据，这种模型通常可以实现较高的可用性。而强一致性则保证对于已更新的数据的所有读取，都能获取已更新的数据，因此带来了一定的写延时，在可用性方面略显不足。

Redis支持最终一致性协议，其余数据库均支持强一致性协议。接下来，我们分别选取两款支持不同的一致性协议的分布式数据库进行对比分析。由于etcd在DB-Engines排名较高，相关测试数据比较丰富，我们选取采用强一致性的etcd数据库和采用最终一致性的Redis数据库进行对比。

基于本课题的需求分析，我们从读写性能和一致性协议两方面进行对比。

①读写性能方面：如图4所示，Redis性能高，读写速度快；如图5所示，etcd性能略差，读写速度至少比Redis要差一个数量级左右。

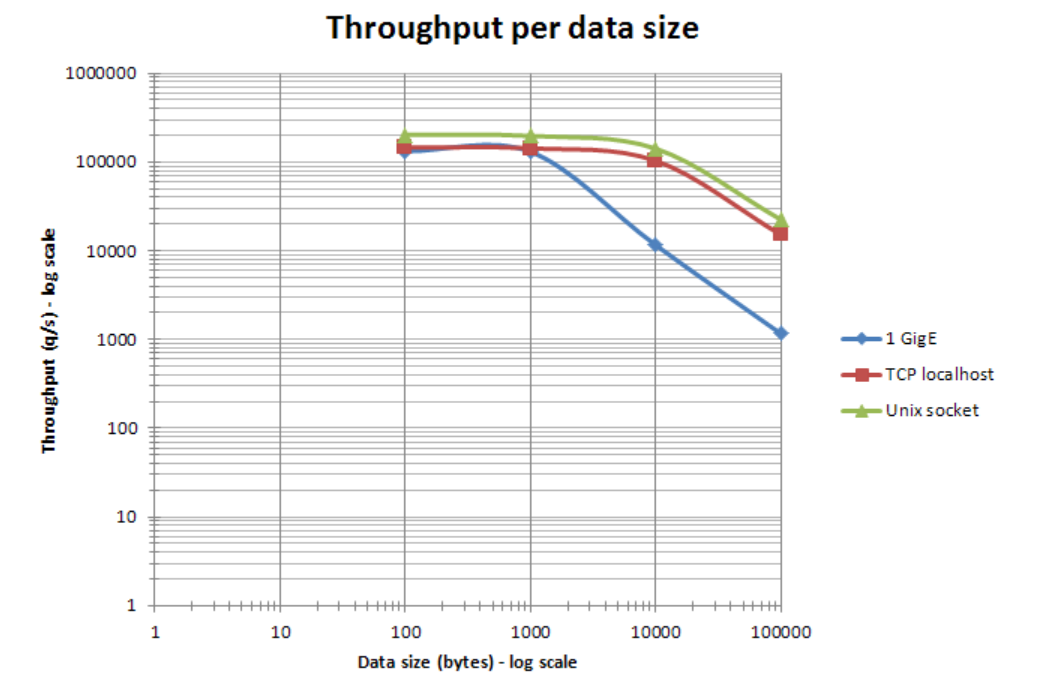


图4 Redis吞吐量

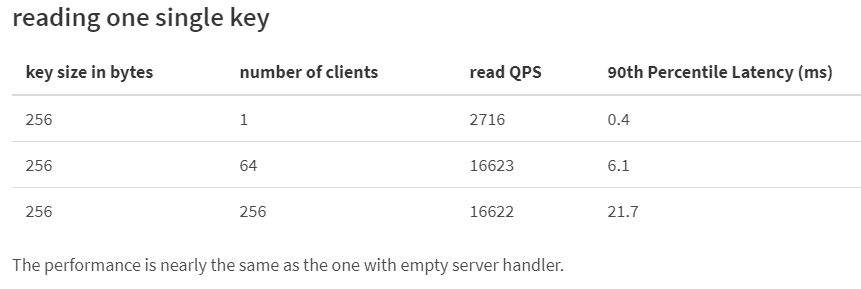


图5 etcd读写性能

②一致性协议方面：我们考虑不同的一致性协议对跨域路由带来的影响。Redis采用最终一致性协议，当SDN控制器收到业务请求时，从数据库中读取拓扑等信息，计算得到从起点到终点的路由P之后，为了便于后续的控制器直接查看（减少冗余计算），将路由P写入数据库，写入成功即可下发流表，不必等待分布式数据库之间数据同步完成。在保证高性能的同时，会对业务路由造成以下影响。

1. 路由变差：当路由P下发之后，由于网络拓扑变化导致最优路由不再是原先计算得到的路由P，业务按照路由P进行转发将产生资源浪费。对于这种情况，可以通过动态路径规划算法对路由进行调整。
2. 路由失败：当路由P下发之后，由于网络拓扑变化导致路由P无法成功到达目的节点，业务将被丢弃。对于这种情况，等待数据库同步之后重新计算路由即可，类似于强一致性协议的效果。

而etcd采用Raft强一致性协议，对于控制器计算出的每个路由流表项（包括域内路由），都必须先写入数据库，等待其余全部数据库同步完成之后，才能下发流表。因此，尽管etcd能够保证域间路由最优，但是为了等待分布式数据库间的数据同步，将会带来额外的写延迟。

在本课题中，不同的一致性协议对业务路由的影响差异不大。因此，我们倾向于采用高性能的Redis分布式数据库。

1. 逻辑拓扑的构建

天基网络卫星节点成千上万，数量较多，规模较大。并且，由于卫星节点质量体积受限，单个SDN控制器处理能力是有限的，难以实现全网的流量管控。因此，本课题需要设计具有一定拓展性的分布式架构，构建可拓展性好的逻辑拓扑，以适应天基网络的应用需求。

随着网络规模的增大，传统的SDN单控制器由于存在CPU、内存等方面的性能瓶颈，在网络管控的性能方面存在明显的上限。为了突破单控制器的性能瓶颈，SDN分布式多控制器部署应运而生，目前比较常见的多控制架构主要有2种：扁平分布式架构和层次分布式架构。

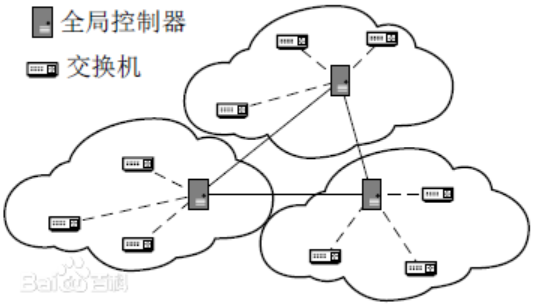


图6 扁平式架构

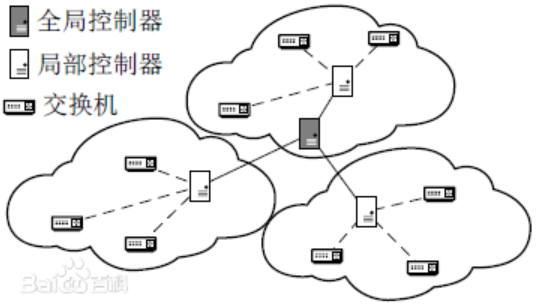


图7 层次式架构

扁平分布式架构如图6所示，网络中的所有SDN控制器都处于相同地位，各个SDN控制器分别部署在不同的区域，分别管控不同的网络设备。各个SDN控制器之间通过东西向接口实现控制信令的交互，在逻辑上都掌控着全局的网络状态信息。当网络状态发生变化时，所有控制器将同步完成信息更新。因此，为了使得每个SDN控制器都掌握全局的网络状态信息，扁平式架构的信息同步负载较大，但是在路由决策方面拥有较为简单的流程。

层次分布式架构如图7所示，网络中的SDN控制器分为2种角色：一种是局部SDN控制器，负责管理本区域内的网络设备，仅仅掌握本区域的网络状态信息；另一种是全局SDN控制器，负责管理各个局部SDN控制器，完成局部SDN控制器之间的交互，同时掌握着全局的网络状态信息。因此，层次式架构的信息同步负载较小，但是由于只有全局SDN控制器能够汇总得到全局的网络状态信息，跨域路由的控制流程将会稍显复杂。

基于以上两种SDN多控制器架构的对比分析，我们利用分布式数据库合成新的混合式架构。一方面，为了简化路由决策的控制流程，采用扁平结构实现SDN控制器的同层互联；另一方面，为了降低网络状态信息同步的负载，引入分布式数据库来储存网络状态信息，并实现全局网络状态信息的同步。



图8 单个域的逻辑拓扑示意图（待修改）

逻辑拓扑结构如上图8所示，本课题的分布式框架共分为三层。最上层是分布式数据库，中间层是SDN控制器，最下层是卫星交换设备。一个分布式数据库连接多个SDN控制器，一个SDN控制器连接多个卫星交换设备。每个SDN控制器负责管控本区域内的网络设备，并将网络状态信息写入相应的分布式数据库。当分布式数据库完成数据同步后，每个SDN控制器都可以通过访问本地连接的分布式数据库读取全局的网络状态信息，计算出最优的路由决策。

1. 实体之间的接口设计

本课题中的主要实体是SDN控制器和分布式数据库，基于以上介绍的逻辑拓扑架构，不同的SDN控制器之间的信息交互，是通过访问分布式数据库实现的。因此，为了实现SDN控制器之间的信息交互，我们需要设计实现SDN控制器和分布式数据库之间的接口。

本课题选取的SDN控制器（Mul控制器）和分布式数据库（Redis）都是采用C语言编程实现的，因此我们可以将Redis的C语言接口库hiredis集成到SDN应用程序中，实现SDN控制器对Redis的远程访问和数据读写。

hiredis是一个非常全面的C语言版redis接口库，支持所有命令、管道与脚本，API接口函数主要有以下三部分：

①连接数据库

redisContext \*redisConnect(const char \*ip, int port);

输入参数分别是Redis数据库的IP地址和默认监听的端口号（6379），返回值是一个指向redisContext结构体的指针。

②发送命令请求

void \*redisCommand(redisContext \*c, const char \*format, ...);

输入参数分别是redisConnect()函数返回的指向redisContext结构体的指针，以及待执行的Redis命令。返回值是一个void类型的指针，实际为一个指向redisReply类型的指针。返回结果的类型reply->type有以下六种情况，如下表所示：

|  |  |
| --- | --- |
| reply->type字段 | 内容 |
| REDIS\_REPLY\_STRING=1 | 返回值是字符串，字符串存储在reply->str中，字符串长度为reply->len |
| REDIS\_REPLY\_ARRAY=2 | 返回值是数组，数组大小存储在reply->elements中，数组里面存储的都是指向redisReply类型的指针，可以通过reply->element[i]->str访问 |
| REDIS\_REPLY\_INTEGER=3 | 返回值是证书，类型为long long |
| REDIS\_REPLY\_NIL=4 | 返回值为空，表示命令执行结果为空 |
| REDIS\_REPLY\_STATUS=5 | 返回值是命令执行的状态，命令执行成功返回“OK”，存储在reply->str中 |
| REDIS\_REPLY\_ERROR=6 | 返回值是命令执行的错误信息，存储在reply->str中 |

表5 redisCommand()函数返回结果类型

③释放资源

void freeReplyObject(void \*reply);

void redisFree(redisContext \*c);

freeReplyObject()函数的输入参数是指向redisReply结构体的指针，redisFree()函数的输入参数是指向redisContext结构体的指针，会清理连接资源并释放连接。

1. 数据同步技术研究

Redis分布式数据库作为分布式数据库系统，本身拥有一套成熟的数据复制方式，可以提供数据冗余、故障恢复、负载均衡和高可用等服务。

Redis的主从复制侧重于解决数据的多机热备，是指将一台Redis服务器的数据复制到其他的Redis服务器，源Redis服务器作为主节点，目的Redis服务器作为从节点。数据复制是单向的，只能从主节点复制到从节点。默认情况下，每台Redis服务器都是主节点，一个主节点可以有多个从节点，但一个从节点只能有一个主节点。

在本课题中，每个SDN控制器负责管控多个SDN交换机，并通过南向接口和SDN交换机交互，获得网络拓扑、链路时延估计等网络状态信息，然后访问本地部署的分布式数据库并写入网络状态信息。每一个分布式数据库一开始写入的都是局部的网络状态信息，当不同的分布式数据库之间完成数据同步之后，每一个分布式数据库都将拥有全局的网络状态信息，进而可以为SDN控制器的路由决策提供更加强大的数据支持。因此，本课题中的分布式数据库的同步场景属于多主同步，每一个分布式数据库都需要把自己独有的网络状态信息，同步给其余的分布式数据库。

在Redis主从复制过程中，一个Redis从节点只有一个主节点，不支持多个主节点同时进行数据复制。所以，我们需要重新调研选取合适的数据同步方案。基于天基网络的轻量级需求，我们找到了以下几款C语言实现的Redis数据同步方案：分别是Redis-shake数据同步迁移工具、Redis-Migrate-Tool集群迁移工具、Dynomite同步方案。

①Redis-shake数据同步迁移工具

Redis-shake支持源Redis服务器和目的Redis服务器之间的数据单向同步，支持全量和增量数据的迁移，支持单节点、主从版、集群版之间的互相同步。其基本原理如下图9所示，Redis-shake模拟一个从节点加入源Redis节点，首先进行全量拉取（sync），然后进行增量拉取（psync），把从源Redis节点收到的数据写入目的Redis节点。

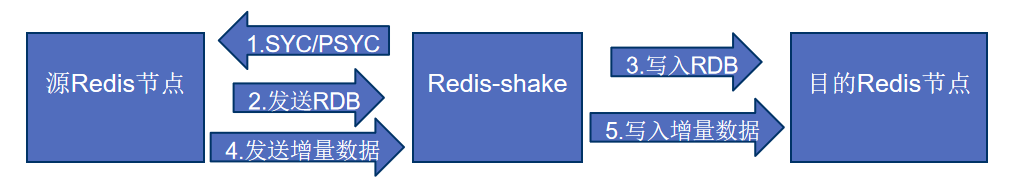


图9 Redis-shake原理图

Redis-shake是Redis主从复制的拓展，本质上仍然只支持数据单向复制，不适合多主复制的应用场景。

②Redis-Migrate-Tool集群迁移工具

Redis-Migrate-Tool具有快速、多线程，实时迁移的特点，迁移过程中，源集群不影响对外提供服务，并且可以显示迁移状态。支持Twemproxy集群，Redis cluster集群，Redis单节点，RDB快照文件和 AOF日志文件的数据迁移。

尽管Redis-Migrate-Tool可以实现双向数据复制，但是由于没有对时间戳进行处理，存在键值覆盖的问题，不能保证最终数据库中保留的数据是最新修改的数据。因此，Redis-Migrate-Tool的应用场景一般是Redis主备数据库之间的单向复制。

③Dynomite同步方案

Dynomite是Netflix实现的一个Redis数据库同步复制方案，支持Redis数据库之间的数据复制，提供读请求高可用服务。拥有对等式网络（P2P，Peer to Peer）和线性可扩展性，每个Dynomite集群中的节点都有相同的职责，所以不存在单点故障。

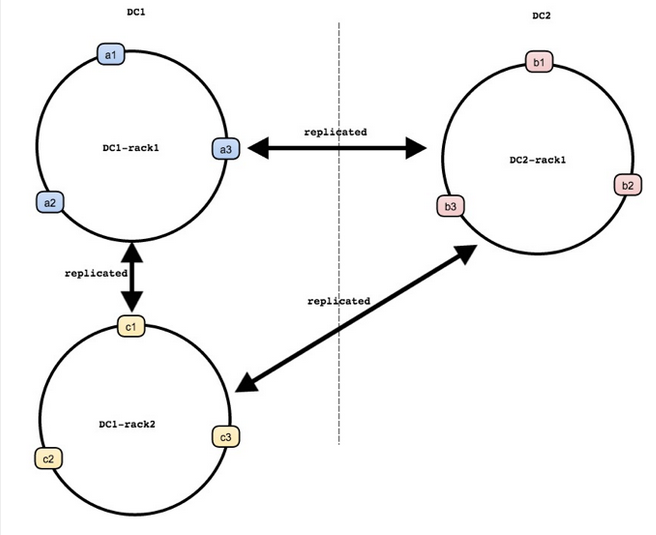


图10 Dynomite拓扑结构图

Dynomite拓扑结构如上图10所示：一个Dynomite集群包含多个数据中心（dc），每个数据中心包含多个机架（rack），每个机架包含多个节点，每一个节点都拥有一个特殊的令牌（token）号，这个令牌标记用来识别出每一个节点所拥有的数据集。每个机架都包含了完整的数据集，这些数据集被分配在机架的多个数据节点上。因此，多个机架就能够提供数据服务的高可用性。

在数据的一致性保证方面，Dynomite同步方案中采用版本向量（节点标识、逻辑修改时间）记录各个节点对数据的修改历史，从而比较数据新旧、发现并修改冲突。如下图11所示，D1->D2，D2->D3，D2->D4都满足第一个时钟上的计数器小于或等于第二个时钟上的所有节点的计数器，因此可以判断他们是因果关系，可以将第一个版本更新为第二个版本。D3和D4则不满足上述条件，因此可以判断他们属于并行分支，需要保留两个版本的数据并通过语义协调得到新的数据版本D5，在保证数据一致性的前提下完成数据更新。

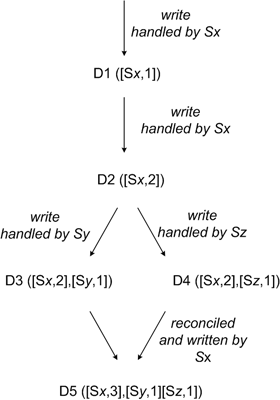


图11 数据版本演变示意图

Dynomite数据同步方案如下图12所示，共有三个组件：Dynomite提供高可用性和线性可扩展集群，可以跨越多个数据中心；Redis提供了高性能的后端，并且是Dynomite支持的主要API；Client是使用Redis协议与Dynomite进行通信的客户端，本课题中采用C语言客户端hiredis。

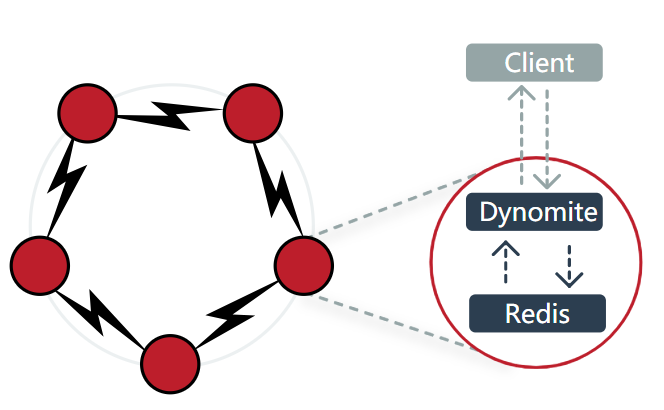


图12 Dynomite集群视图

根据官方测试，通过将1k个键/值对写入一个区域中的Dynomite，然后随机轮询另一个区域以获取20个键，来测量键/值对在另一区域副本上可用的时间。每个键的值只是写操作开始时的时间戳。然后，其他区域中的客户端将读取这些时间戳并计算持续时间。重复几次得到，Dynomite数据复制的平均持续时间约为85毫秒。

总而言之，作为P2P的分布式数据同步方案，Dynomite支持多主集群架构，配置使用相对较为简单直观，同时对Redis的支持度相当高，满足开发使用需求。在本课题中，我们可以把全部分布式数据库划分为同一个数据中心下的不同机架，利用Dynomite同步方案可以保证各个机架的分布式数据库都掌握有全局的网络状态信息。

1. 动态路径规划算法设计

本课题的预期目标是实现并应用分布式网络的动态路径规划算法，由SDN控制器下发流表，完成业务的路由。算法设计主要分为3个部分：网络状态信息收集、网络状态信息存储、路径规划算法应用。

①网络状态信息收集

网络状态信息收集是SDN控制器通过和SDN交换机进行Packet-In和Packet-Out报文交互，从而获取链路拓扑、链路时延估计等信息的过程。根据网络通信的流程，我们可以把网络状态信息收集划分为以下3个模块：拓扑发现、链路时延估计、用户PC注册。

1. 拓扑发现模块

传统网络中的交换机采用链路层发现协议（LLDP，Link Layer Discovery Protocol）完成邻居发现和网络拓扑构建，而SDN网络中的交换机不具备控制平面的功能，因此需要SDN控制器通过链路层发现协议（LLDP，Link Layer Discovery Protocol）和南向接口OpenFlow协议来实现拓扑发现功能。工作流程如下：

①SDN交换机主动连接到SDN控制器，通过新增交换机接入事件触发SDN控制器的相应回调函数，此时SDN控制器可以获取交换机的网桥ID，从而通过OpenFlow协议和交换机进行报文交互。

②SDN控制器构建LLDP报文，封装到Packet-Out报文中，并指定交换机行动为向入端口以外的全部端口发送LLDP数据包，然后将Packet-Out报文发送给SDN交换机。

③SDN交换机按照Packet-Out报文指定的行动将LLDP报文洪泛出去。

④当LLDP报文通过SDN交换机之间的链路到达相邻的SDN交换机之后，SDN交换机由于没有匹配的流表项，将会通过Packet-In报文将收到的LLDP数据包发送给SDN控制器。

⑤SDN控制器收到Packet-In报文之后，不仅可以获取LLDP接收端SDN交换机的入端口，还可以从LLDP数据包中获取LLDP发送端SDN交换机的出端口。通过拓扑发现模块，SDN控制器可以获得SDN交换机之间的全部拓扑连接关系。

1. 链路时延估计模块

链路时延可以基于以上介绍的拓扑发现模块进行估计。当SDN控制器发送LLDP数据包之前，在LLDP报文中记录下报文发送时间的时间戳t1；当SDN控制器接收到Packet-In报文之后，记录下报文接收时间的时间戳t2。二者之差即为总时延，共包含3部分：SDN控制器到SDN交换机SW1的时延，交换机SW1和交换机SW2之间的链路时延，SDN交换机SW2到SDN控制器的时延。

然后我们可以单独测试并估计SDN控制器到SDN交换机之间的控制通道时延t\_sw，工作流程如下所示：

①SDN控制器构建LLDP报文，在LLDP报文中记录下报文发送时间的时间戳t1’，封装到Packet-Out报文中，并指定交换机行动为向入端口发送LLDP数据包，然后将Packet-Out报文发送给SDN交换机。

②SDN交换机按照Packet-Out报文指定的行动将LLDP报文重新发送给SDN控制器。

③SDN控制器收到LLDP报文之后，记录下报文接收时间的时间戳t2’，由此即可估计出SDN控制器到SDN交换机之间的时延t\_sw=(t2’-t1’)/2。

最后，利用以上过程得到的时延进行计算，即可估计出SDN交换机SW1和SW2之间的链路时延t12=t2-t1-t\_sw1-t\_sw2。

1. 用户主机注册模块

网络中的业务流都有各自的源和目的，从实际情况出发，我们可以认为每个业务都是从一个用户发往另一个用户的。因此，为了在以上介绍的拓扑发现模块构建的网络拓扑中，标记每一个用户所在的位置，我们需要实现用户注册功能。

传统网络通过地址解析协议（ARP，Address Resolution Protocol），将网络层地址解析到数据链路层地址，以便于通过IP地址定位MAC地址。在我们构建的网络拓扑中，每个节点代表SDN交换机的某个端口，因此只需要将用户PC的IP地址绑定到PC连接的SDN交换机端口上，即可实现用户在网络拓扑中的节点映射。

因此，我们可以在SDN控制器上利用ARP协议和OpenFlow协议实现用户注册模块，工作流程如下：

①源主机已知目的主机的IP地址，因此在发送数据包之前，会按照ARP协议向SDN交换机发送ARP Request报文，查询目的IP对应的MAC地址。

②SDN交换机收到源主机发送的ARP Request报文，将会把ARP Request报文以及交换机入端口信息封装进Packet-In报文中，发送给SDN控制器。

③SDN控制器收到ARP Request报文后，将会进行源学习，将源主机IP和对应的交换机端口绑定注册。然后判断目的主机IP是否注册，如果目的主机IP已经注册，则返回固定的MAC地址，以便跳过ARP协议进行IP路由；否则SDN控制器将ARP Request报文的源MAC地址修改为本地SDN控制器MAC地址后，封装进Packet-Out报文，并指定交换机行动为向除入端口以外的全部端口转发，然后发送给该控制器管控的全部SDN交换机。（为了避免形成广播风暴，需要提前在SDN交换机上设置相关流表项，直接丢弃源MAC地址为本地SDN控制器MAC地址的数据包）

④SDN交换机按照Packet-Out报文指定的行动，将ARP Request报文洪泛出去。

⑤目的主机收到ARP Request报文后，将会向SDN交换机发送ARP Reply报文。

⑥SDN交换机收到目的主机发送的ARP Reply报文，将会把ARP Reply报文以及交换机入端口信息封装进Packet-In报文中，发送给SDN控制器。

⑦SDN控制器收到ARP Reply报文后，将会进行源学习，将目的主机IP和对应的交换机端口绑定注册。

根据以上工作流程，SDN控制器可以实现用户主机IP和SDN交换机端口的绑定，从而为后续的路由计算提供源节点和目的节点的相关参数。为了实现跨域路由，我们还需要对上述流程加以补充说明：

①SDN控制器只对本区域内的用户主机进行注册。SDN控制器收到ARP报文进行源学习之前，需要对源MAC地址的有效性进行判断，确保源MAC地址不是控制器MAC地址，从而保证该报文不是由其他区域的SDN控制器修改并转发过来的。

②SDN控制器需要提前向本区域内的SDN交换机下发相应的流表项，丢弃源MAC地址为本地控制器MAC地址的数据包，防止ARP Request报文在本区域内SDN交换机之间的广播风暴。

②网络状态信息存储

为了使得每个SDN控制器都能拥有全局的网络状态信息，需要将单个SDN控制器获得的局部的网络状态信息写入分布式数据库。当分布式数据库之间完成数据同步之后，每个SDN控制器就可以通过访问分布式数据库，读取全局的网络状态信息。

本课题拟选用的Redis数据库支持字符串、散列、列表、集合等丰富的数据结构，适用于解决各种场景下的应用问题。作为分布式键值数据库，Redis数据库中按照Key-Value模式进行数据存储，我们需要对上述网络状态信息进行合适的数据存储结构设计，以便于进行Redis数据库的读写和查询操作。

①链路时延估计信息

链路时延估计信息采用散列结构存储，各字段定义如下表6所示。我们可以通过“HSET Key Field Value”命令将数据写入数据库，通过“HGET Key Field”命令或“HGETALL Key”命令从数据库中读取相应数据。

|  |  |  |
| --- | --- | --- |
| Key | link\_delay | 该散列结构存储链路时延估计信息 |
| Field | port1 | 32比特，表示链路连接的交换机端口1，高16位表示SDN交换机连接的SDN控制器ID，中8位表示SDN交换机ID，低8位表示SDN交换机的端口ID |
| port2 | 32比特，表示链路连接的交换机端口2 |
| Value | delay | 64比特，表示两个SDN交换机之间的链路时延，单位us |

表6 链路时延数据存储结构设计

②SDN控制器到SDN交换机的控制通道时延估计信息

控制通道时延估计信息采用散列结构存储，各字段定义如下表7所示：

|  |  |  |
| --- | --- | --- |
| Key | sw | 该散列结构存储SDN控制器到交换机的时延估计信息 |
| Field | cid | 16比特，表示SDN交换机连接的SDN控制器ID |
| sid | 8比特，表示SDN交换机ID |
| Value | delay | 64比特，表示SDN控制器和交换机之间的链路时延 |

表7 控制通道时延数据存储结构设计

③用户主机注册信息

用户主机注册信息采用散列结构存储，各字段定义如下表8所示：

|  |  |  |
| --- | --- | --- |
| Key | pc | 该散列结构存储用户主机IP地址在逻辑拓扑中的节点映射 |
| Field | ip | 32比特，表示用户主机的IP地址 |
| Value | port | 32比特，表示用户主机连接的交换机端口 |

表8 用户主机注册数据存储结构设计

④业务路由信息

业务路由信息采用列表结构存储，各字段定义如下表9所示。我们可以通过“LPUSH Key Value”命令将数据写入数据库的相应列表，通过“LRANGE Key 0 -1”命令从数据库中读取相应列表的全部数据，通过“LTRIM Key 1 0”命令清空相应列表的全部数据。

|  |  |  |
| --- | --- | --- |
| Key | ip\_src | 32比特，表示源主机的IP地址 |
| ip\_dst | 32比特，表示目的主机的IP地址 |
| Value | out\_sw\_port | 32比特，表示SDN交换机出端口 |

表9 业务路由数据存储结构设计

③路径规划算法应用

根据第二章的简单介绍，我们可以发现Dijkstra算法可用于计算单个源节点到其余全部节点的最短路径，当找到通往目的节点的最短路径时即可返回结果；而Floyd算法进行了更多的冗余计算，以得到任意两个节点之间的最短路径。

由于本课题的业务路由问题主要是计算一个源节点到一个目的节点的最短路径，因此通过Dijkstra算法和Floyd算法的对比，我们选择采用Dijkstra算法进行业务路由计算。

1. **实验环境搭建与测试**
2. **实验环境搭建**
3. **连通性测试**
4. **鲁棒性测试**
5. **实验结果分析**
6. **全文总结与展望**
7. **本文工作总结**
8. **后续工作展望**

参考文献：

1. McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks[J]. ACM SIGCOMM computer communication review, 2008, 38(2): 69-74.