计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(9):72−80 [doi: 10.15888/j.cnki.csa.007050]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 HTTPS 的 RPKI 缓存更新机制①

耿新杰[1,2], 马　迪[1,2,3], 毛　伟[2,3], 邵　晴[3]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100190)
[3](互联网域名系统北京市工程研究中心, 北京 100190)
通讯作者: 耿新杰, E-mail: gengxinjie@zdns.cn

摘　要: RPKI 作为解决路由劫持等网络安全问题的重要网络架构, 其传输结构主要有两方面构成: 供给侧和依赖方的数据同步, 以及依赖方和需求侧的数据传输. 目前国内外研究内容主要集中在供给侧和依赖方的数据同步环节. 依赖方和需求侧的数据传输仍处于初步探究状态. 本文针对当前 RPKI 理论架构难以适应实际部署需求的缺陷, 设计并利用 JSON 化的 RPKI 缓存数据, 实现了一种基于 HTTPS 的 RPKI 缓存更新架构. 实验结果表明, 该分发架构传输稳定. 与当前 RPKI 理论架构相比能够适应多层传输和大量数据传输的需要.

关键词: RPKI; 数据同步; 数据传输; HTTPS; JSON; 缓存分发架构

引用格式: 耿新杰,马迪,毛伟,邵晴.基于 HTTPS 的 RPKI 缓存更新机制.计算机系统应用,2019,28(9):72–80. http://www.c-s-a.org.cn/1003-3254/7050.html

RPKI Cache Update Mechanism Based on HTTPS

GENG Xin-jie[1,2], MA Di[1,2,3], MAO Wei[2,3], SHAO Qing[3]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)
[3](ZDNS Co. Ltd., Beijing 100190, China)

Abstract: As an important network architecture for solving network security problems such as route hijacking, RPKI has two main components: data synchronization between the supply side and the relying party, and data transmission between the relying party and the demand side. At present, the research content worldwide mainly focuses on the data synchronization link between the supply side and the relying party. The data transmission between the relying party and the demand side is still in the initial state of exploration. In view of the shortcomings of the current RPKI theoretical architecture, we design and use Jsonized RPKI cache data to implement an HTTPS-based RPKI cache update architecture, which can meet the needs of actual deployment. Experimental results show that the distribution architecture is stable. Compared with the current RPKI theoretical architecture, it can adapt to the needs of multi-layer transmission and large-scale data transmission.

Key words: RPKI; data synchronization; data transmission; HTTPS; JSON; cache distribution architecture

1　背景

1.1　现状

BGP (Border Gateway Protocol, 边界网关协议) 缺乏验证真实性和合法性的安全手段, 它容易遭到各种恶意攻击. 为解决 BGP 缺乏安全认证的问题, Kent S 等人于 2002 年提出了 S-BGP[1]. 基于 S-BGP 的设计

① 收稿时间: 2019-02-27; 修改时间: 2019-03-15; 采用时间: 2019-03-25; csa 在线出版时间: 2019-09-05

思想, IETF (Internet Engineering Task Force, 互联网工程任务组), RPKI (Resource Public Key Infrastructure, 资源公钥基础设施) 的技术标准化工作, 旨在提供路由信息交换的认证[2].

RPKI 数据传输结构如图 1 所示, RPKI 依赖方同步 RPKI 认证中心或资料库 (供给侧) 的数据, 并将验证数据分发给 BGP 路由系统 (需求侧) 以供路由器进行验证. RPKI 依赖方需要相当频繁的验证证书和 CRL, 性能将是数据传输首要考虑的问题[3]. 文献[4]实现了一种基于有序哈希树的 RPKI 资料库同步工具, 改进了目前数据同步协议 Rsync 协议[5]未考虑到 RPKI 中文目录的缺陷, 能够大量减少中文数据同步时间和资源消耗. 文献[6]实现了基于 Delta 协议[7]的 RPKI 数据同步, 能够解决目前基于 Rsync 协议进行数据同步所具有的安全隐患. 文献[8]实现了基于哈希表的 RPKI 证书验证优化方法, 通过使用哈希表的方式取代数据库查询进行数据验证, 能够减少数据验证时间消耗. 这些研究逐步完善供给方与依赖方数据同步环节. 但依赖方与需求侧的数据传输研究目前被关注较少, 伴随着 RPKI 在全球进入部署应用阶段, 依赖方与需求侧的数据传输机制是 RPKI 能够发挥作用的关键要素.

1.2 RPKI 体系架构

RPKI 通过建立一整套公钥证书体系对 INR (Internet Number Resource, 互联网码号资源) 的所有权 (分配) 和使用权 (路由起源通告) 进行验证, CA (Certificate Authority, 证书颁发机构) 在进行资源分配的过程中, 同时会签发一系列标识资源从属关系的认证证书并将该证书数据分发到路由处以供验证. 数据传输结构如图 1 所示, 证书数据存储于全球 RPKI 数据库, RP 服务器会从全球 RPKI 数据库同步证书信息并验证, 将验证后的信息缓存于本地数据库中. 路由器通过 RTR (RPKI To Router)[9]协议获取 RP 数据库中的 RPKI 缓存数据, 并将该缓存数据用于验证路由起源通告是否合法.

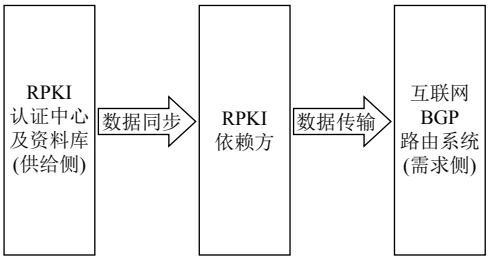

图 1 RPKI 数据传输结构

1.3 RPKI 大规模部署的体系要求

目前 RPKI 体系架构仅在部分 ISP 或研究机构进行小规模部署实验. 根据文献[10]结论, 在大规模部署的条件下, 需要满足以下要求:

1) 负载均衡: 传输架构负载承受能力满足实际运行需要.

2) 传输效率: 传输任务能在给定时间范围内到达指定地址.

3) 可靠性: 传输架构能适用于复杂的网络环境.

但当前现有的 RPKI 缓存分发结构, 在大规模部署的情况下, 难以满足 RPKI 缓存分发的要求, 从以下四点进行分析:

1.3.1 负载均衡问题

RPKI 缓存更新结构存在如下几个原因, 难以满足负载均衡的要求.

1) RP 服务器的负载能力有限: 目前的缓存更新结构中, RPKI 缓存由 RP 服务器直接向所有的路由器进行分发. 在实际部署过程中, RP 服务器的负载能力有限. 一旦大量的路由器同时对 RP 服务器发起数据传输请求, 会给 RP 服务器带来巨大的负载压力, 有可能造成信息阻塞、RP 服务器宕机等问题.

2) 对全球 RPKI 数据库构成压力: 为了降低实际部署情况下 RP 服务器的负载压力, 需要在网络中大量部署 RP 服务器, 这会给全球 RPKI 数据库带来负载压力. 这种改进方案仅仅是将 RP 服务器的负载压力转移到全球 RPKI 数据库处, 同样不能满足运行稳定的要求.

1.3.2 非一致性和本地化控制问题

由于 RP 服务器直接从全球 RPKI 数据库获取证书数据, 其获取数据内容的方式和验证过程完全独立, 即不与其他 RP 服务器相关, 大规模部署 RP 服务器, 会导致同一区域内 (例如一个 ISP(Internet service provider, 互联网服务提供商)) 路由信息的冲突, 产生非一致性问题.

IETF 在 RFC 8416[11]中, 考虑到 ISP 本地化信息控制的需求, 提出了 SLURM (Simplified Local Internet Number Resource Management, 简化本地互联网码号资源管理). RP 服务器会通过本地信任锚点获取 SLURM 文件来对 RPKI 缓存数据进行补充和修改, 以达到本地化信息控制的目的. 在这种情况下, 大量的部署 RP 服务器会增加管理上的难度和本地化控制信息传输的难度.

1.3.3 传输效率问题

在现有的 RPKI 缓存更新结构中, RP 服务器与路由器之间的缓存更新通过 RTR 协议进行. RTR 协议是专门为存储 RPKI 缓存的服务器与路由器之间进行数据传输设计的一种协议结构, 但 RTR 协议仅能通过路由器主动发起传输请求, 且不适用于复杂的网络架构. 因此既难以满足实际在实际运行过程中对 RPKI 缓存分发效率的要求也不能保证分发过程的稳定. 具体原因如下.

1) 无法主动推送及时响应: 服务器不能主动向客户端进行数据传输, 只能依靠客户端周期性的从服务器处获取数据. 如果直接将 RTR 协议用于缓存分发, 会出现如图 2 所示的问题, 即当实际部署中出现多层分发结构时, 周期性的获取数据会导致最下层的路由器获取 RPKI 缓存数据花费的时间是中间多层服务器获取缓存数据的周期之和, 导致重要的 RPKI 缓存数据不能得到及时分发, 无法及时响应.

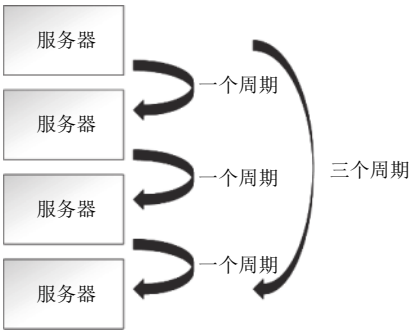

图 2 RTR 协议应用于多层传输结构的问题

2) 数据结构设计不合理: 在目前的 RPKI 缓存分发结构中, RPKI 缓存通过 RFC 6810 规定的数据结构进行描述, 这是一种专为 RTR 协议设计的数据结构. 该数据结构中只能包含一条 RPKI 缓存更新数据, 影响传输效率, 并且该数据结构只能通过 RTR 协议进行解析和编写, 缺少可读性和通用性, 实际部署成本较大.

1.3.4 可靠性问题

IETF 在 RFC 6810 中提到将 RTR 协议不适合用于复杂的拓扑结构, 因为 RTR 协议在复杂拓扑结构中容易出错, 例如: 不能保持无环路条件. 此外基于 TCP 协议的 RTR 协议, 在实际网络情况下不便于进行跨网段传输.

经过上述分析, 当前的 RPKI 缓存分发机制不能满足大规模部署时 RPKI 缓存分发的需求. 本文针对以上四种缺陷设计了一种基于 HTTPS 的缓存更新机制, 该机制能够解决这些缺陷, 满足大规模部署时 RPKI 缓存分发的需求.

2 基于 HTTPS 的 RPKI 缓存分发架构

本文针对实际部署中 RPKI 缓存数据分发的问题, 利用 JSON 化的 RPKI 验证缓存数据实现了一种基于 HTTPS 的缓存分发机制, 如图 3 所示. 在本文设计的机制中, RP 服务器和路由器之间的 RPKI 缓存分发, 通过搭建多级 VC 服务器实现. 在 ISP 的重要管理节点即 RP 服务器的部署地点处, 搭建 VC 服务器. RP 服务器和该 VC 服务器之间通过同一数据库共享 RPKI 缓存. 该 VC 服务器与其它 VC 服务器之间通过 HTTPS 协议构建安全传输信道进行 RPKI 缓存数据的分发. 此外在每个 AS 内均部署有相应的 VC 服务器. AS 内的路由器通过 RTR 协议从 VC 服务器处获取 RPKI 缓存数据.

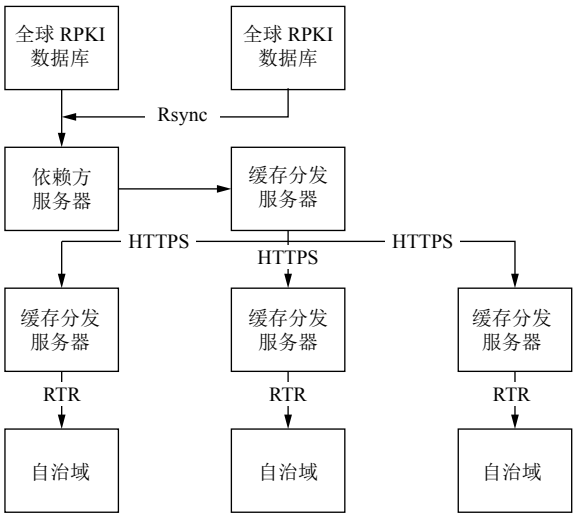

图 3 基于 HTTPS 的缓存分发架构图

2.1 实际应用场景分析

上述设计是大规模部署中 RPKI 缓存分发的基本模型, 在实际应用场景中, 可以根据不同场景下的需求对分发结构进行进一步的优化或修改. 在实际生活场景中, 部分路由系统的搭建由互联网运营商完成. 由于互联网运营商的实际管理模式为层级化管理或扁平化管理. 经过探究, 本文提出两种不同应用场景中 RPKI 缓存分发架构的改进.

2.1.1 互联网运营商扁平化管理场景

在扁平化管理场景下, 基本模型虽然能满足管理

架构的需求, 但是直接由单一 VC 服务器对其它所有管理范围内的 VC 服务器进行 RPKI 缓存数据的分发并处理其它 VC 服务器特殊情况下发来的请求, 会使得服务器负载压力过大. 因此本文在基于基本模型提出一种优化架构, 如图 4 所示.

通过搭建中间多级 VC 服务器, 将单一 VC 服务器对管理范围内其它 VC 服务器的传输优化为多 VC 服务器对其它 VC 服务器的传输. 这种新的架构能减少 VC 服务器的负载压力, 此外新的传输架构也能够更好的契合互联网运营商的扁平化管理.

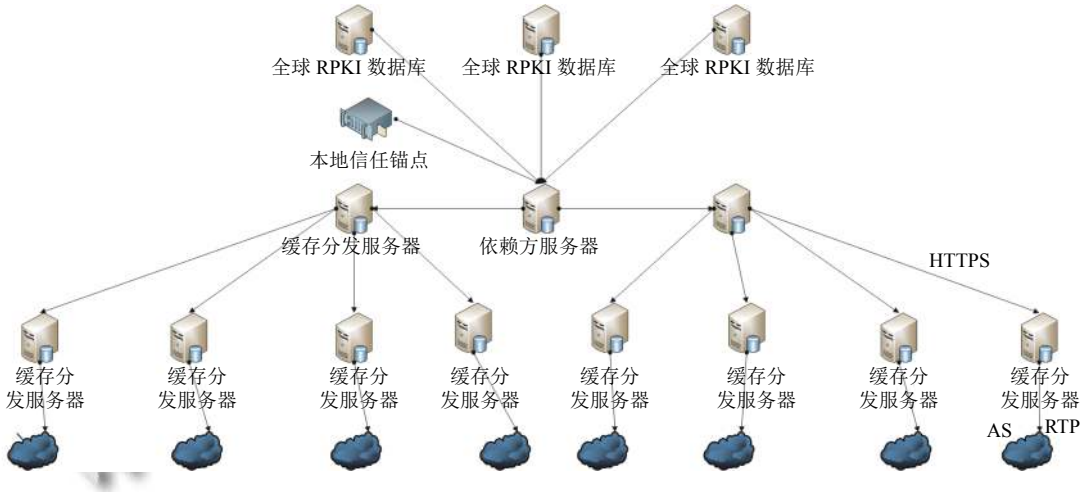

图 4 扁平化管理场景下 RPKI 缓存分发架构的优化

2.1.2 互联网运营商层级化管理场景

在层级化管理的场景下, 由单一 VC 服务器进行对其它所有管理范围的 VC 服务器进行 RPKI 缓存数据分发的机制不能很好的契合互联网运营商的管理模式的需求. 因此可对这种场景下的 RPKI 缓存分发架构进行相应的修改. 在图 5 所示的架构中, 与层级化管理相对应的每一层 VC 服务器都可以接受其本地信任锚点的本地化控制信息文件, 并通过该文件对其子层级进行本地化信息控制. 经过修改后的传输架构与互联网运营商的层级化管理结构契合度更高, 也更易于使用层级化管理的互联网运营商进行本地化信息控制.

2.2 RPKI 缓存分发设计

本文实现的传输架构能够解决实际部署中的负载均衡问题以及非一致性和本地化控制问题. 架构设计出于以下考量.

1) 降低 RP 负载: 通过搭建 VC 服务器, 可以显著降低 RP 服务器的负载压力. AS 内部的路由器通过其所在 AS 部署的 VC 服务器获取 RPKI 缓存数据. RP 服务器只需将通过验证后的 RPKI 缓存数据传输到事先配置好的 VC 服务器处即可.

2) 降低全球 RPKI 数据库压力: VC 服务器仅通过其上级 VC 服务器或者 RP 服务器获取 RPKI 缓存数据, 并不需要访问全球 RPKI 数据库进行证书数据同步, 不会额外增加全球 RPKI 数据库的负载压力.

3) 保证路由信息的一致性: 由 VC 服务器进行 RPKI 缓存的分发, 不需要大量部署 RP 服务器. 能够保证一定区域内路由信息的统一性.

4) 适应本地化控制的需求: VC 服务器从更高一级 VC 服务器或 RP 服务器中获取验证缓存数据, 保证了同一区域内的路由信息通过同一 RP 服务器进行管理和修改, 降低了管理难度, 也减少了本地化控制信息传输的难度.

5) 降低 RP 服务器的功能复杂度: 通过在 RP 服务器处搭建 VC 服务器进行 RPKI 缓存分发, 降低 RP 服务器设计的复杂度. 将同步功能和分发功能分离, 具有更高的松耦合度和灵活度, 有效的降低了实际部署难度.

2.3 HTTPS 分发设计

本文设计方案选用 HTTPS 协议作为 VC 服务器之间 RPKI 缓存分发的传输协议, 能够解决 RTR 协议的周期叠加问题和可靠性问题. RFC8484[12]提出了一种基于 HTTPS 的 DNS 查询机制, 该文档提出, 可以使用 HTTPS 发送 DNS 查询, 并通过 HTTP 获取 DNS 响应. 本文设计方案参考了 RFC 8484, 将该文档提出的场景视作是 RPKI 缓存数据分发的一种相近场景.

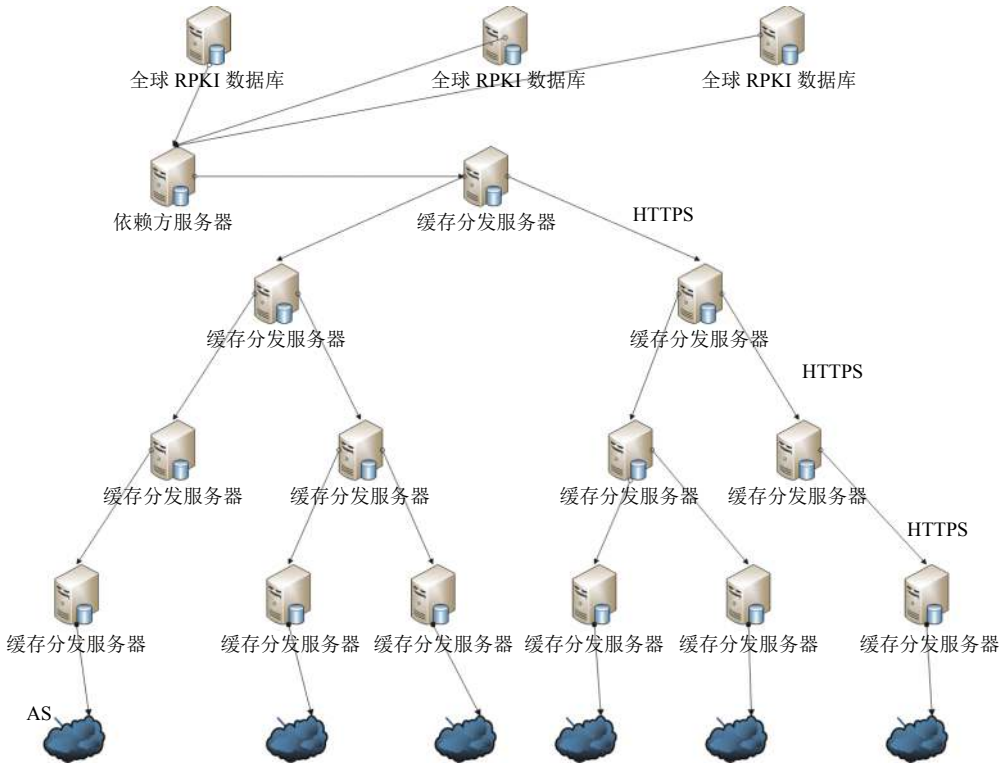

图 5 层级化管理场景下 RPKI 缓存分发架构的优化

本方案采用 HTTPS 协议有以下原因:

1) 允许主动推送, 适用于应急响应: HTTPS 协议允许由客户端主动发起连接请求, 能够迅速分发 SLURM 文件进行本地化信息控制, 避免错误配置长时间影响本地.

2) HTTPS 适用范围广, 能够适应复杂网络拓扑结构: HTTPS 应用于几乎所有的操作系统, 适用范围广.

3) HTTPS 协议是一种面向内容的应用层协议, 它简化了对数据格式的要求和协议兼容设计的细节问题, 可以传输任意的数据对象.

因此采用 HTTPS 协议, 既能够满足及时分发 RPKI 缓存的需要, 又能够适应实际部署中复杂的网络结构. 所以本文提出的设计方案采用 HTTPS 协议作为 VC 服务器之间分发 RPKI 缓存的传输协议.

2.4 通用的数据格式

本文基于 RFC 8416 中用来描述 SLURM 文件的 JSON 格式的数据结构, 参考了 RFC 8427[13](该文档中使用 JSON 格式对 DNS 查询和响应进行了描述). 使用 JSON 用以描述 RPKI 缓存更新, 能够解决 RTR 协议规定的数据结构在实际部署中只能传输单条数据和缺少通用性的问题.

RFC8416 中为描述 SLURM 文件制定了一种 JSON 格式的数据结构. 在实际分发过程中, 通过该结构来描述 RPKI 缓存更新的内容, 可以简化与 SLURM 文件的交互接口. 不需要提供专门的解析和重构模块,. 同时该数据结构可应用于增量更新, 增量更新的方式可以降低全量更新所消耗的资源, 提高分发效率, 能够满足实际运行的需要.

因此, 本文提出的设计方案采用 RFC 8416 中规定的 JSON 格式的数据结构来描述 VC 服务器之间分发的 RPKI 缓存.

3 详细设计

3.1 数据结构完整设计

考虑到网络传输的复杂性, 以及在实际场景中, ISP 对 VC 服务器数据库中 RPKI 缓存数据的管理控制需求, 本方案在 RPKI 缓存更新的基础上添加控制信息, 将 RPKI 缓存与控制信息封装为一个完整的数据包. 控制信息和 JSON 格式的 RPKI 缓存数据完全解耦, 控制信息可用 JSON 和 XML 等结构化语言进行描

述, JSON 和 XML 数据包结构的具体设计如下所示.

JSON 格式如下:

```
{
  "head": {
    "operate": "new",
    "time": "2019-02/22/19 15:31:24",
      "newversion": "2",
    "toversion": "2",
    "sha1":"**********",
    "as": "1"
  },
  "data": {
    "slurmVersion": 1,
    "validationOutputFilters": {
    "prefixFilters": [
      {
        "prefix": "1.10.10.0/24",
        "comment": "test prefix filters"
      }
    ]
  },
  "locallyAddedAssertions": {
  "prefixAssertions": [
    {
    "asn": 199998,
    "prefix": "19.99.98.0/24",
    "comment": "test assertions"
      }
    ]
    }
  }
}
```

XML 格式如下:

```
<?xml version="1.0" encoding="utf-8"?>
<test>
    <head>
        <operate>new</operate>
        <time>2019-02/22/19 15:31:24</time>
        <newversion>2</newversion>
        <toversion>2</toversion>
        <sha1>**********</sha1>
        <as>1</as>
    </head>
    <data>{
        "slurmVersion": 1,
                "validationOutputFilters": {
        "prefixFilters": [
          {
          "prefix": "1.10.10.0/24",
          "comment": "test prefix filters"
          }
        ]
      },
      "locallyAddedAssertions": {
      "prefixAssertions": [
      {
        "asn": 199998,
        "prefix": "19.99.98.0/24",
        "comment": "test assertions"
      }
    ]
  }</data>
</test>
```

在上述所示的数据包结构中, 数据包由头部 (header) 和数据 (data) 两部分构成. 其中头部字段含义如表 1 所示.

表 1 头部 (header) 字段

字段	字段类型	含义
operate	varchar	new 表示数据库执行更新, back 表示数据库执行回退
time	varchar	RPKI 缓存的分发时间
newversion	int	RPKI 缓存的版本
toversion	int	operate 字段为 new, 取值和 newversion 相同; operate 字段为 back, 取值由管理人员决定, 表示数据库回退的版本号
sha1	varchar	数据包完整性校验的结果
as	varchar	目标 VC 服务器 (ALL 表示目标 VC 服务器为所有的下级 VC 服务器)

Operate 字段包括 new 和 back 两个方式, new 表示直接在 VC 服务器当前状态的数据库进行 RPKI 缓存更新, back 表示需要 VC 服务器回退到之前版本的数据库.

As 字段用以实现面向不同的 AS 的本地化信息控制视图, 管理者可通过修改 as 字段, 构建针对不同 AS 的本地化信息控制视图.

3.2 VC 服务器

在本文的设计方案中, 共有 3 个实体, RP 服务器, VC 服务器和路由器, 由于 RP 服务器与路由器的设计在工业上已有实现, 故本文设计方案仅讨论分发架构中 VC 服务器的设计.

3.2.1 VC 服务器功能模块

为了能够安全高效的进行 RPKI 缓存分发, 结合第 2 节中的设计思路, VC 服务器应具有五种功能模块. (1) 数据包构建模块: 该模块用于将 RPKI 缓存描述为如 2.3 所述格式, 并通过添加头部控制信息, 构建如 3.1 所述的数据包; (2) 传输模块: 根据头部控制信息中的 as 字段将 RPKI 缓存通过 HTTPS 协议分发到对应 as 区域中的 VC 服务器, 并通过多线程实现一次性多目标分发; (3) 数据解析模块: 该模块用于解析收到的 RPKI 缓存数据包, 通过头部控制信息对 RPKI 缓存进行相应的处理; (4) 重传模块: 当 VC 服务器收到不完整数据包或者发现有数据包遗漏时, 该模块会向其上级 VC 服务器请求重新获取 RPKI 缓存更新, 并提供相应的控制信息. 当 VC 服务器接收到下级 VC 服务器的重发请求时, 将根据其提供的控制信息, 调用数据包构建模块构建数据包, 并通过传输模块进行信息的传输. 如图 6 所示.

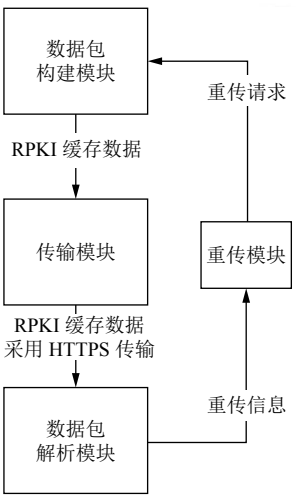

图 6 VC 功能模块示意图

3.2.2 安全保护与可靠性

由于 HTTPS 的安全加密通道只能够保证传输过程中的数据安全. 在本文的设计方案中还建立了 IP 访问控制和数据完整性检查两种机制进行安全性保护, 并通过重传机制增强分发架构的可靠性.

1) IP 访问控制

VC 服务器在接受数据包之前, 首先会对传输数据的 IP 进行检查, 如果该 IP 不合法, VC 服务器会直接抛弃该数据包, 不对其中的数据进行解析. 该机制可以保证 VC 服务器接受的 RPKI 缓存数据的来源是可靠的.

2) 数据完整性检查

VC 服务器在进行数据包解析之前, 首先会根据 sha1 字段中的数据对数据包进行完整性校验, 如果校验结果不相符, 则抛弃该数据包, 并通过重传请求模块向上级 VC 服务器申请数据包的重传.

3) 重传机制

当数据包未能通过数据完整性检测或者数据解析过程中发现 newversion 字段与数据库中的最新版本相差大于 1, VC 服务器就会启用重传机制, 向上级 VC 服务器发出请求重新获取数据包.

4 实验与分析

4.1 RPKI 缓存分发实验

RPKI 缓存分发流程如图 7 所示, VC 服务器会根据需要分发的 RPKI 缓存更新, 利用 3.1 所述数据结构构建分发数据包, 通过 HTTPS 协议将数据包分发到指定 AS 处的 VC 服务器. VC 服务器会对数据包进行检测, 如果验证失败则触发重传请求, 请求重传 RPKI 数据包. 之后, 根据数据包进行相应的处理和更新.

4.1.1 RPKI 缓存分发实验设计与结果分析

实验中使用五台物理机模拟传输架构进行实验, 将五台物理机分别标号为 1–5, 其中 1 号机 7999 端口作为 RP 服务器, 8000 端口作为 VC 服务器, 2–5 号机依次作为不同层级的 VC 服务器.

实验使用 JSON 格式的数据包进行传输测试. 通过输出时间戳 (毫秒精度) 计算发起连接到通信完成之间的时间差, 从而得到实际过程中的时间效率. 本实验通过 3 个端口 (8000–8002) 得到每个层级分发用时的平均值 (同时测试多线程分发) 并计算总分发用时, 如图 8 所示.

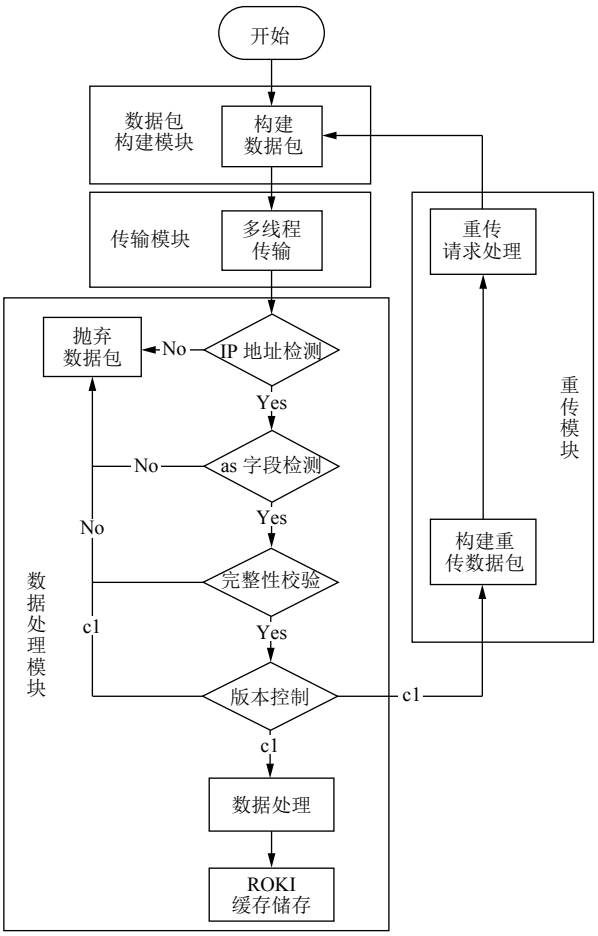

图 7 RPKI 缓存分发流程图

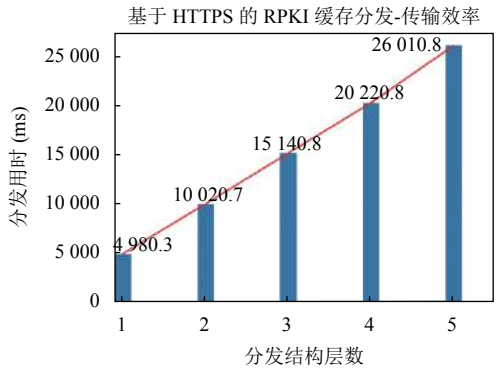

图 8 RPKI 缓存分发用时图

图中的 X 轴表示分发结构中 VC 服务器的传输层数, Y 轴表示总分发用时平均值 (单位: ms), 实际测量得到, 5 层分发结构仅用时 26 010.8 ms(26.0108 s). 在实际部署场景中, 该架构能够迅速稳定的分发 RPKI 缓存, 可以满足实际运行的需要.

本文还对核心模块进行单独测试, 以下给出运行结果.

4.1.2 数据包构建模块

数据包构建模块通过用户输入或者根据重传请求构建数据包, 以 JSON 格式为例.

运行结果如图 9 所示.

图 9 数据包构建模块测试结果

4.1.3 传输模块

传输模块根据数据包头部 AS 字段获取目标 AS 的 IP 地址, 并通过多线程的方式分发 RPKI 缓存, 服务器端运行结果如图 10 所示.

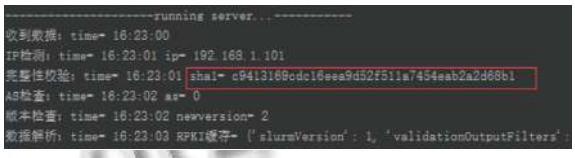

图 10 服务器端运行结果

4.1.4 数据处理模块

数据处理模块功能主要分为以下两部分:

1) IP 访问控制, 完整性检测与版本控制

2) 数据处理与储存

运行结果如图 11 所示.

newversion	operate	asn	prefix	prefixlength
2	Filters		1.10.10.0	24
2	Filters	999996	1.9.6.0	24
2	Filters	1	10.89.56.0	24
2	Filters	483	144.96.69.(	24
2	Assertion	199998	19.99.98.0	24

图 11 数据处理模块运行结果

4.2 性能测试

为了准确评估本方案的实际运行效果, 还设计了

对比试验. 第一组为使用 RTR 协议进行 RPKI 缓存的分发. 第二组为使用 HTTPS 协议进行 RPKI 缓存的分发. 由于架构和数据处理操作相同, 本实验只对 RTR 协议和 HTTPS 协议的传输时间进行测试 (不计算数据处理用时), 本实验根据 RTR 协议逐条传输的特点, 测试在五种不同数目的 RPKI 更新缓存数据 (分别为 10 条, 20 条, 50 条, 100 条, 200 条) 的情况下, RTR 协议与 HTTPS 协议的传输时间对比. 实验结果如图 12 所示.

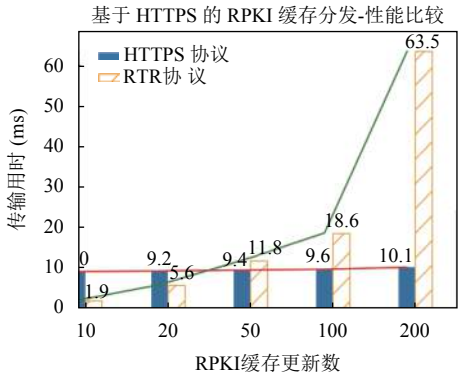

图 12 性能比较结果

从图 11 可以看出, 当 RPKI 缓存更新数较少的时候, RTR 协议传输效率比 HTTPS 协议高, 但是随着缓存更新数的增加, RTR 协议传输时间也显著增加, 而 HTTPS 协议并无明显变化. 在 RPKI 缓存更新数达到 200 时, RTR 协议用时为 HTTPS 协议的 6.28 倍. 因此在实际部署环境中, 使用 HTTPS 协议可以使得传输效率更加稳定.

此外, 文献[9]提到在 RTR 协议中客户端每间隔一小时通过服务器端获取一次数据. 在多层分发架构中, 客户端获取数据的周期叠加会使得分发时间长达数小时. 而本文提出的基于 HTTPS 协议的分发架构能够使得分发时间不会受到周期叠加的影响, 可以高效分发 RPKI 缓存.

5 结束语

随着互联网时代的进一步发展, RPKI 的部署势在必行, 也获得了国内外许多组织的大力推行. 但是在实际部署过程中还有许多问题需要进行研究. 在这种背景下, 本文利用 JSON 化的 RPKI 验证缓存数据实现了一种基于 HTTPS 的缓存更新机制. 本文首先说明了现有 RPKI 架构的难以应用于实际部署的缺陷, 并给出了本方案的设计考量, 然后给出了设计的完整内容和具体细节. 在此基础上本文用 python 对该设计进行了实现并与 RTR 协议进行对比.

RPKI 作为路由安全领域的热点话题, 既是为互联网运行保驾护航的重要举措, 也是互联网进一步发展的基石. 因此如何将 RPKI 更好的部署在实际场景中也是一个重要的探索方向和研究内容.

参考文献

1 Kent S, Lynn C, Seo K. Secure border gateway protocol (S-BGP). IEEE Journal on Selected Areas in Communications, 2000, 18(4): 582–592. [doi: 10.1109/49.839934]

2 Lepinski M, Kent S. An infrastructure to support secure internet routing. RFC 6480. 2012. [doi: 10.17487/RFC6480]

3 Reynolds MC, Kent S. A high performance software architecture for a secure internet routing PKI. Proceedings of 2009 Cybersecurity Applications & Technology Conference for Homeland Security. Washington, DC, WA, USA. 2009. 49–53.

4 许圣明, 马迪, 毛伟, 等. 基于有序哈希树的 RPKI 资料库数据同步方法. 计算机系统应用, 2016, 25(6): 141–146.

5 Weiler S, Ward D, Housley R. The rsync URI scheme. RFC 5781. 2010.

6 司昊林, 马迪, 毛伟, 等. RPKI 增量同步 Delta 协议的形式化检测与实现. 计算机系统应用, 2018, 27(11): 1–8. [doi: 10.15888/j.cnki.csa.006562]

7 Mandelberg D, Bruijnzeels T, Muravskiy O, *et al*. The RPKI repository delta protocol. RFC 8182. 2017. [doi: 10.17487/RFC8182]

8 安春林, 马迪, 王伟, 等. 基于哈希表的 RPKI 证书验证优化方法. 计算机系统应用, 2018, 27(2): 132–137. [doi: 10.3969/j.issn.1003-3254.2018.02.022]

9 Bush R, Austein R. The resource public key infrastructure (RPKI) to router protocol, RFC 6810. 2013. [doi: 10.17487/RFC6810]

10 胡军. 复杂网络下多服务注册中心部署策略研究[硕士学位论文]. 重庆: 重庆大学, 2011.

11 Ma D, Mandelberg D, Bruijnzeels T. Simplified local internet number resource management with the RPKI (SLURM), RFC 8416. 2018. [doi: 10.17487/RFC8416]

12 Hoffman P, McManus P. DNS queries over HTTPS (DoH), RFC 8484. 2018. [doi: 10.17487/RFC8484]

13 Hoffman P. Representing DNS messages in JSON, RFC 8427. 2018. [doi: 10.17487/RFC8427]