基于有序哈希树的 RPKI 资料库数据同步方法①

许圣明[1,2], 马 迪[3], 毛 伟[2,3], 王 伟[2,3]

[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)

摘 要: RPKI(Resource Public Key Infrastructure, 互联网码号资源公钥证书体系) 中的签名对象由 RP(Relying Party, 依赖方) 端同步下载后处理成 IP 地址块与 AS (Autonomous System, 自治域)号的真实授权关系, 用于指导 BGP 路由. 当前的 RP 使用软件 rsync (Remote Sync)来同步, 而 rsync 的同步算法并未考虑 RPKI 中文件(目录)的特点, 导致同步效率并不理想. 通过分析并结合 RPKI 中文件(目录)的特点, 设计并实现了一种基于有序哈希树的 RPKI 资料库同步工具 htsync. 实验结果表明, 与 rsync 相比较, htsync 在同步时的数据传输量较少, 同步时间较短. 在设计的 3 种实验场景下, 同步时间平均加速比分别为 38.70%、30.13%和 3.63%, 有效地减少了同步时的时间和资源的消耗.

关键词: 互联网码号资源公钥证书体系; 有序哈希树; 数据同步

RPKI Repository Synchronization Method Based on Ordered Hash Tree

XU Sheng-Ming[1,2], MA Di[3], MAO Wei[2,3], WANG Wei[2,3]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)
[3](Internet Domain Name System Beijing Engineering Research Center Ltd, Beijing 100190, China)

Abstract: RP(Relying Party) downloads signed objects in RPKI(Resource Public Key Infrastructure) and processes those objects into authorised relations between IP addresses and AS(Autonomous System), which is used to guide the BGP routing. The current RP uses rsync to realize the synchronization, but rsync(Remote Sync) synchronization algorithm does not take the characteristics of the files (directorise) in the RPKI into account. So the synchronization is not efficient. Through the analysis and combining with the characteristics of the files (directories) in the RPKI, this paper designs and realizes a RPKI repository synchronization tool named htsync which is based on ordered hash tree. The experimental results show that, compared with rsync, htsync transmits less data and costs less time during synchronization. In three designed experimental scenario, average speedup ratios of synchronization time are 38.70%, 30.13% and 3.63%, effectively reduce the consumption of time and resources.

Key words: RPKI; ordered hash tree; data synchronization

RPKI 的兴起源自一种互联网安全威胁——域间路由劫持, 确切地说, 是“自治域间路由劫持”[1]. RPKI 通过构建一个 PKI(公钥证书体系)来完成对 INR(Internet Number Resource, 互联网码号资源; 包括 IP 地址前缀和 AS 号)的所有权(分配关系)和使用权(路由起源通告)的认证, 并以此“认证信息”来指导边界路由器, 帮助其检验 BGP 消息的真实性, 从而避免域间路由劫持[2].

RPKI 包括三大基本组件: CA(Certificate Authority, 认证权威)、RP(Relying Party, 依赖方)和 repository(资料库). 这三大组件通过签发、传送、存储、验证各种数字对象来彼此协作, 共同完成 RPKI 的功

① 收稿时间:2015-10-22;收到修改稿时间:2015-11-25 [doi:10.15888/j.cnki.csa.005203]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

能. CA 通过签发 RC(Resource Certificate, 资源证书)来表达 INR 分配关系, 签发 ROA(Route Origin Authorization, 路由起源声明)[3] 来授权某个 ISP 针对自己的一部分 IP 地址前缀发起源路由通告; RPKI 资料库负责存储这些承载了 INR 分配/授权信息的 RC/ROA 等数字对象, 供全球的 RP 下载; RP 负责从 RPKI 资料库中下载同步这些数字对象, 并将其处理成 IP 地址块与 AS 号的真实授权关系, 指导 BGP 路由.

作为连接 RPKI 中记载的 INR 分配/授权关系与实际 BGP 路由之间的桥梁, RP 无疑是 RPKI 体系中重要的角色[4], RP 端与 RPKI 资料库数据同步的效率也决定了 RPKI 体系运行的性能. 目前, RPKI 体系的 RP 端在同步数据时使用开源软件 rsync. 虽然 rsync 能够高效地同步文件和目录, 但是 rsync 并非为 RPKI 体系而设计, 也就是说, rsync 的原理和实现并未考虑 RPKI 体系中文件和目录的特点, 在同步 RPKI 资料库数据时的效率并不理想.

IETF(Internet Engineering Task Force, 互联网工程任务组)已经意识到当初选择 rsync 的草率和 rsync 在 RPKI 体系中的效率问题, 目前正在讨论制定用于 RPKI 资料库同步的 delta 协议[5]. 2014 年 12 月发布了最新的 delta 工作组草案, 阐述了 delta 协议的设计思想和原理, RPKI 资料库保存其状态的快照, 当有数据更新时, 利用 Notification 文件通知 RP 端更新数据. 但是 delta 协议对 RPKI 体系的运行过程改动比较大, 需要 RPKI 进行协议层面的更改, 仍在工作组讨论之中, 短期内并不能够标准化和被工业界采用.

针对 RP 端和 RPKI 资料库同步时效率不足的问题, 本文提出一种基于有序哈希树的 RPKI 资料库数据同步方法. 根据 RPKI 资料库中文件(目录)的特点, 以文件为粒度, 文件(目录)的相对路径和修改时间戳的哈希值为校验依据, 构建文件(目录)的有序哈希树, 利用有序哈希树管理 RP 端和 RPKI 资料库中的文件(目录), 实现 RP 端和 RPKI 资料库的高效同步. 本文基于上述原理设计实现了同步工具 htsync, 并设计实验对比 htsync 和 rsync 的同步性能, 实验结果表明, 与 rsync 相比, htsync 在同步时的数据传输量较少, 同步时间也较短, 在设计的 3 种实验场景下, 同步时间平均加速比分别为 38.70%、30.13%和 3.63%, 有效地减少了同步时的时间和资源的消耗.

1 背景

1.1 rsync 算法基本原理

RP 端和 RPKI 资料库同步时使用的是开源软件 rsync, rsync 算法由 Andrew Tridgell 于 1996 年发明[6]. 作为 unix/linux 下同步文件的一个高效工具, rsync 能同步两处计算机的文件(目录), 并且只传输两处文件(目录)的不同之处, 进而减少了传输的数据量. 假设在两处计算机上分别存放着文件 A 和 B, 现使用 rsync 来同步这两份文件, 使得 B 与 A 一致, rsync 算法描述如下:

① 将文件 B 分割为不重叠的大小固定为 S 字节的文件块. 最后一个文件块大小可能小于 S 字节.

② 对文件 B 的每一个文件块计算两个校验值: 一个 32 位弱校验值和一个 128 位强校验值.

③ 将文件 B 的这些校验值发送给文件 A 所在的计算机.

④ 使用比对算法, 在文件 A 中校验所有的大小为 S 字节的文件块. 进而确定两个文件的不同之处, 传输不同的文件块.

rsync 将文件划分为固定大小的文件块, 通过比较这些文件块的弱校验值或者强校验值, 来确定文件的哪些部分发生了变化, 最终将变化的部分传输到目的计算机, 达到高效同步的目的.

1.2 rsync 与 RPKI

虽然 rsync 能够高效地同步文件和目录, 也作为一个流行的工具广泛使用在诸多场景中. 但是 rsync 并非专门为 RPKI 体系而设计, rsync 的原理和实现并未考虑 RPKI 体系中文件和目录的特点.

(1) RPKI 资料库文件数量多, 读取时占用较多的 I/O 资源. RPKI 资料库同步过程涉及到的文件种类有: CA 证书(.cer)、ROA(.roa)、资料清单(.mft)、CRL 证书(.crl)和 Ghostbusters 证书(.gbr)等[7]. 截止到 2015 年 6 月 30 日, 全球 RPKI 资料库文件大小为 290M 字节, 共 47763 个文件, 而此时 RPKI 在全球的部署率仅为 6.4%. 可以想象, 如果 RPKI 今后在全球普遍推广部署后, RPKI 资料库将是十分庞大的. 而 rsync 在同步文件时需要将文件读入内存分块和计算校验值, 如果文件数量巨大, 那么读取数量庞大的文件无疑会占用很多的 I/O 资源, 对同步的整体性能会产生不利的影响.

(2) 文件尺寸小, 可以以文件为单位传输, 而非文

© 中国科学院软件研究所 http://www.c-s-a.org.cn

件块. RPKI 体系中的文件的尺寸通常比较小, 经统计, 在上述资料库的 47763 个文件中文件尺寸分布图如图 1 所示.

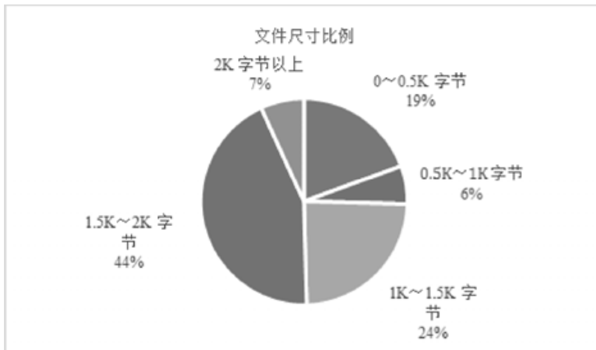

图 1 文件尺寸分布图

从分布比例可以看出, 文件大小在 2K 字节以内的文件数量占总文件数的 93%. 这些文件按照发布点的不同, 组织在不同的目录中. 另外, RPKI 体系中的文件是数字签名对象, 签名对象的内容包含文件的语义内容和该语义内容的哈希值[8]. 由于文件语义内容的哈希值和文件语义内容之间的强关联性, 因此当文件语义内容发生变化时, 即使是细微的改变, 哈希值也将彻底改变, 结果整个签名对象的内容也会彻底的改变, 不存在签名对象局部更新的情况. 考虑到 RPKI 体系中小尺寸文件居多, 并且可以忽略文件内容的局部更新的情况, 所以在 RPKI 资料库同步时可以以文件为传输的单位, 不必将文件再划分为更小的文件块. 这样一方面减少了读取文件划分文件块时的性能消耗, 另一方面并不会传输过多的协议数据.

根据以上所述的 RPKI 资料库中文件的特点, 本文设计实现了一种基于有序哈希树的 RPKI 资料库同步工具 htsync.

2 使用有序哈希树的同步方法

2.1 有序哈希树

本文采用一种有序哈希树来检测 RPKI 资料库文件和 RP 端文件副本的变化. 其主要优势在于不需要逐个对文件浏览比较, 可以快速获取文件变化. 有序哈希树是与 RPKI 资料库目录或 RP 端目录相对应的树, 表示该目录的特征, 其节点可以用四元组<p, hash, firstChild, nextSibling>表示, 其中, p 表示该节点对应的文件或者目录的相对路径; hash 表示该节点存储的哈希值, 在本文中, 有序哈希树中文件节点的哈希值为文件的相对路径和文件的修改时间戳拼接后进行哈希运算的哈希值, 目录节点的哈希值为其子节点(目录内文件对应的的文件节点)的哈希值拼接后进行哈希运算后的哈希值; fristChild 表示该节点的头子节点; nextSibling 表示该节点的后继兄弟节点. 图 2 给出了目录和有序哈希树的对应的示例. 有序哈希树的构建算法 buildHT 如下所示.

buildHT 算法

输入: RPKI 资料库文件(目录)或 RP 端文件(目录)f;

输出: 有序哈希树 HT(f).

(1) 令文件 f 的相对路径为 p, 为文件 f 创建有序哈希树的节点 node = <p, NULL, NULL, NULL>, 若 f 为目录, 则进入(2); 否则, 假设文件的修改时间为 t, 拼接 p 和 t, 形成字符串 l, 文件 f 的哈希值为 h = Hash(l), 令 node.hash = h, 进入(4);

(2) 若 f 是目录, 对于其任意的直属子目录或文件 f’, 创建与 f’对应的有序哈希树 HT(f’) = buildHT(f’), 进行如下处理:

① 令节点 node’ = HT(f’).root, 若 node.firstChild = NULL, 则 node.firstChild = node’, 进入(2); 若 node.firstChild ≠ NULL, 则记 chNode = node.firstChild;

② 设文件 f’的相对路径为 p‘, 若 p’ < chNode.p, 则令 node.firstChild = node’, node’.nextSibling = chNode, 进入(2);

③ 若 chNode.nextSibling = NULL, chNode.nextSibling = node’, 进入(2); 否则令 nextNode = chNode.nextSibling;

④ 若 p’ < chNode.p, 则令 chNode.nextSibling = node’, node’.nextSibling = nextNode; 否则令 chNode = nextNode, 进入○3;

(3) 对于节点 node 的所有子节点 chNode, 按顺序依次拼接 chNode.hash, 形成字符串 s, node.hash = Hash(s);

(4) 令 HT(f).root = node, 退出.

其中, 记与文件(目录)f 对应的有序哈希树为 HT(f), 记 HT(f)的根节点为 root. 由 HT 的构建算法可知, 对于文件而言, HT 中对应节点的哈希值即为文件修改时间和路径拼接后的字符串的哈希值(步骤(1)); 对于目录而言, 则首先分别创建其直属子目录或子文

© 中国科学院软件研究所 http://www.c-s-a.org.cn

件的对应节点(步骤(2)); 然后将其直属子节点的哈希值依次进行拼接, 并将拼接结果的哈希值作为其对应节点的哈希值(步骤(3)). 由于有序哈希树中子节点按照相对路径进行了排序(步骤②~④), 因此只要目录的某个子文件内容或者目录结构发生变化, 其对应的 MT 节点的哈希值必然会发生变化. 对于 RP 端来说, 如果 MT 根节点 root 的哈希值发生变化, 也就意味着 RPKI 资料库发生了变化; 反之, 如果哈希值未变化, 那么 RPKI 资料库也没有发生变化. 这也是 htsync 能够快速检测 RPKI 资料库文件变化的原理.

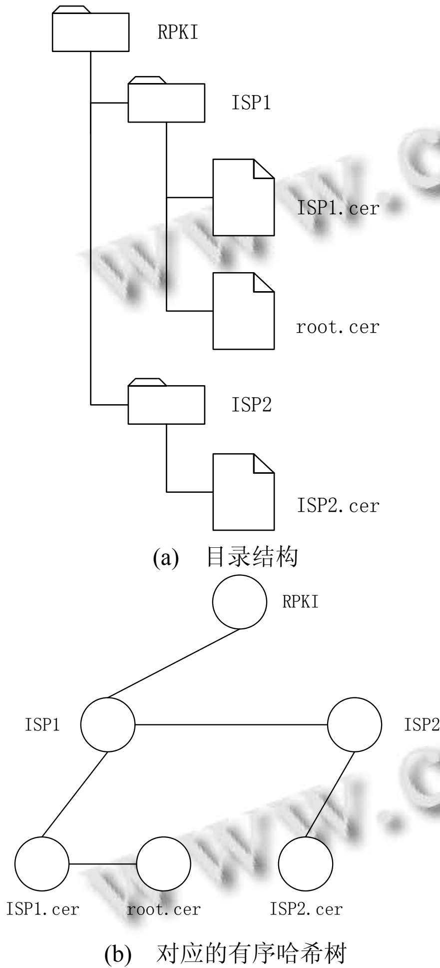

图 2 有序哈希树示例

2.2 RPKI 资料库与 RP 同步原理

htsync 采用 client-server 控制结构, 即 RPKI 资料库为 server, RP 端为 client. 同步算法 findDiff 如下所示.

findDiff 算法 算法输入: RPKI 资料库有序哈希树 HT, RP 端有序哈希树 HT’, 算法输出: 差异文件列表 list.

(1) 令 list 为 HT 和 HT’比较所得的文件变化列表, root 为 HT 的根节点, root’为 HT’的根节点, 若 root.hash = root’.hash, 进入步骤(3);

(2) 令 node = root.firstChild, node’ = root’.firstChild, 对 node 和 node’ 进行步骤① ~ ⑧的处理:

① 若 node、node’均为 NULL, 进入步骤(3);

② 若 node = NULL 并且 node’ ≠ NULL, 则令 subHT 为 HT’中根节点为 node’的子树, 执行子过程 delete(subHT, list), node’ = node’.nextSibling, 进入步骤①;

③ 若 node ≠ NULL 并且 node’ = NULL, 则令 subHT 为 HT 中根节点为 node 的子树, 执行子过程 add(subHT, list), node = node.nextSibling, 进入步骤○1;

④ 若 node.p < node’.p, 则令 subHT 为 HT 中根节点为 node 的子树, 执行子过程 add(subHT, list), node = node.nextSibling, 进入步骤①;

⑤ 若 node.p > node’.p, 则令 subHT 为 HT’中根节点为 node’的子树, 执行子过程 delete(subHT, list), node’ = node’.nextSibling, 进入步骤①;

⑥ 若 node.p = node’.p 并且 node.hash = node’.hash, 则进入步骤○8;

⑦ 若 node.p = node’.p 并且 node.hash ≠ node’.hash, 则令 subHT 为 HT 中根节点为 node 的子树, subHT’为 HT’中根节点为 node’的子树, 执行子过程 update(subHT, subHT’, list), 进入步骤⑧;

⑧ 令 node = node.nextSibling, node’ = node’.nextSibling, 进入步骤①.

(3) 算法运行结束, 返回 list.

add(subHT, list)

① 令 root 为 subHT 的根节点, 记文件 f = root.p, list.push(f), 令 node = root.firstChild;

② 若 node = NULL, 则子过程结束, 退出; 若 node ≠ NULL, 则令 subHT’为 subHT 中根节点为 node 的子树, 执行子过程 add(subHT’, list);

③ 令 node = node.nextSibling, 重复执行步骤②.

delete(subHT, list)

① 令 root 为 subHT 的根节点, 令节点 node = root.firstChild;

② 若 node = NULL, 则进入步骤④; 若 node ≠ NULL, 则令 subHT’为 subHT 中根节点为 node 的子树, 执行子过程 delete(subHT’, list);

③ 令 node = node.nextSibling, 重复执行步骤②;

④ 记文件 f = root.p, list.puth(f), 子过程执行完毕.

update(subHT, subHT’, list)

① 令 root 为 subHT 的根节点, root’为 subHT’的根节点, 令 node = root.firstChild, node’ = root’.firstChild, 若 node = NULL 并且 node’ = NULL, 则令 nodeHT 为

© 中国科学院软件研究所 http://www.c-s-a.org.cn

subHT 中根节点为 node 的子树, 执行子过程 add(nodeHT, list), 令 nodeHT’为 subHT’中根节点为 node’的子树, 执行子过程 delete(nodeHT’, list), 子过程执行完毕, 否则进入步骤②;

② 令 list’ = findDiff(subHT, subHT’), list. pushAll(list’), 子过程执行完毕.

有序哈希树HT的比对算法findDiff由1个算法主控流程和3个子过程组成, 3个子过程分别对应文件的创建、删除和修改, 而修改由删除旧文件和创建新文件两个操作组成, 算法的主控流程和子过程之间通过递归调用完成了 RPKI 资料库有序哈希树和 RP 端有序哈希树的遍历和比对, 两棵哈希树最多遍历一次就可以完成比对过程.

总体上, 比对流程分为两个步骤. 第一步, 比对两个节点的相对路径, 如果路径不同, 说明 RPKI 资料库和 RP 端的文件(目录)结构不同, 则在 RP 端执行相应地创建或者删除文件(目录)操作. 同理, 如果比对时发现有 NULL 节点, 也说明了文件(目录)结构不一致, 同样在 RP 端执行文件(目录)的创建或删除; 如果两个节点的相对路径一致, 那么进行第二步的比对, 比较两者的哈希值. 如果哈希值一致, 说明两个节点代表的文件(目录)相同, 算法进行下一组节点比对, 反之如果哈希值不一致, 则找到了 RPKI 资料库和 RP 端文件(目录)的不同, 如果该节点表示文件, 那么在 RP 端将旧文件删除然后创建新文件并同步, 而如果该节点表示目录, 则递归调用findDiff算法, 进一步确定变化的节点. 由于有序哈希树的节点包含了 RPKI 文件(目录)的特征信息, 即文件(目录)的修改时间戳和路径信息, 因此通过哈希值的比较就可以快速找到变化的节点, 并且跳过没有变化的节点, 从而提高了比对速度.

假设 RPKI 资料库和 RP 端的目录层次为H, 每个目录平均包含的文件(子目录)数目为N. 如果 RPKI 资料库中的一个文件发生变化, 那么使用 findDiff 算法发现该变化的文件的时间复杂度为 O(H * N), 由于是多项式时间复杂度, 因此在 RPKI 资料库文件数目增大时, 该算法的优势更为明显, 具有较好的适应性.

3 实验分析

3.1 实验设计

RP 端与 RPKI 资料库同步数据时, 可能的情况分为三种: 初始同步、同步更新和数据未变化.

初始同步: RP 端第一次和 RPKI 资料库发生同步, RP 端的数据从无到与 RPKI 资料库数据保持一致;

同步更新: RP端已经和 RPKI 资料库同步过, 并且此次同步时RP端发现 RPKI 资料库数据发生更新, 即数据的新增、删除和修改(RP 端同步时可处理为删除后新增), RP端同步此更新;

数据未变化: RP 端已经和 RPKI 资料库同步过, 并且此次同步时 RPKI 资料库数据未发生变化, RP 端数据保持不变.

当RP端第一次启动时, 会发生“初始同步”, 此时 RP 端没有数据, 需要将全部的 RPKI 数据从资料库同步过来; 当RPKI资料库中数据变化, 即有新的证书签发或者旧证书状态变化时, 发生“同步更新”, RP 端将变化的数据和状态同步过来, 随着今后 RPKI 的广泛部署, “同步更新”在一段时间内会频繁发生; 而当 RPKI 体系趋于稳定, 即资料库中数据变化不频繁时, 发生“数据未变化”. 本文的同步算法能够适应这三种同步情况, 根据RP 端和RPKI资料库端有序哈希树的具体状态, 执行对应的同步操作. 其中, “初始同步”和“同步更新”需要传输数据, 消耗的时间较“数据未变化”多.

表 1 实验环境

CPU	内存	操作系统
Intel Core i5-4590 3.30GHz	8GB	Ubuntu 12.04 LTS

根据以上三种情况设计实验对比 htsync 和 rsync 同步时的同步时间、读写数据量大小. 实验环境如表 1 所示, 算法参数为 RP 端数据存储目录 path/to/RP/cache 和 RPKI 资料库目录 path/to/repository. 实验使用数据来自全球 RPKI 体系资料库, 数据截止到2015年6月30日.

3.2 实验结果

3.2.1 初始同步

设定 RPKI 资料库大小分别为 95M、190M、285M、380M、475M和570M时, 假设Rhtsync和Rrsync分别为 htsync 和 rsync 读的数据量(字节), Whtsync 和 Wrsync 分别为 htsync 和 rsync 写的数据量(字节), Thtsync 和 Trsync 分别为 htsync 和 rsync 的同步时间(秒). htsync 和 rsync 初始同步性能对比如表 2 所示.

表 2 同步性能对比

RPKI 资料库大小	95M	190M	285M	380M	475M	570M
Rhtsync	1080	1080	1080	1080	1080	1080
Rrsync	249.5K	498.9K	748.5K	997.9K	1247.4K	1496.9K
Whtsync	22.8M	45.7M	68.6M	91.4M	114.3M	137.1M
Wrsync	24.0M	47.9M	71.9M	95.9M	119.9M	143.8M
Thtsync	0.74	1.34	7.77	15.68	24.70	29.18
Trsync	0.86	5.17	16.04	24.47	34.73	40.45
加速比(%)	13.95	74.08	51.56	35.92	28.88	27.86

可以看出, htsync 在同步时的读、写数据量上明显少于 rsync, rsync 在同步时需要传输文件块的校验值等协议数据, 所以传输的协议数据量相对较多. 在同步时间上, htsync 要优于 rsync. 随着 RPKI 资料库的增

© 中国科学院软件研究所 http://www.c-s-a.org.cn

大, 同步数据的增多, htsync 和 rsync 的同步时间都随之增多. 假设 Thtsync 为 htsync 的同步时间, Trsync 为 rsync 的同步时间, 同步时间加速比为, $加速比 = \frac{T_{rsync} - T_{htsync}}{T_{rsync}}$, 初始同步时, 平均加速比为 38.70%.

3.2.2 同步更新

设定 RP 端在同步前数据大小分别为 0M、95M、190M、285M、380M 和 475M 时, RPKI 资料库发生更新, 为了测试对比同步时传输的数据量和同步时间, 以新增数据为例. RPKI 资料库新增 95M 的数据, 同步更新后 RP 端数据大小对应分别为 95M、190M、285M、380M、475M 和 570M. 假设 Rhtsync 和 Rrsync 分别为 htsync 和 rsync 读的数据量(字节), Whtsync 和 Wrsync 分别为 htsync 和 rsync 写的数据量(字节), Thtsync 和 Trsync 分别为 htsync 和 rsync 的同步时间(秒). htsync 和 rsync 同步性能对比如表 3 所示.

表 3　同步性能对比

RP 端初始大小	0M	95M	190M	285M	380M	475M
Rhtsync	1080	2160	3240	4320	5400	6480
Rrsync	249.5K	249.5K	249.5K	249.5K	249.5K	249.5K
Whtsync	22.8M	22.8M	22.8M	22.8M	22.8M	22.8M
Wrsync	24.0M	24.8M	25.5M	26.2M	27.0M	27.7M
Thtsync	0.74	0.98	1.48	1.79	1.95	2.29
Trsync	0.86	1.53	2.63	2.91	2.84	2.77
加速比(%)	13.95	35.95	43.72	38.49	31.34	17.33

可以看出, htsync 在同步时的读、写数据量和同步时间上都优于 rsync. 随着 RP 端初始数据量的增大, htsync 的同步时间随之增多, 原因是数据量越大, 构建有序哈希树 buildHT 所需时间就越多. 在上述数据量下, 平均加速比为 30.13%.

3.2.3 数据未变化

设定 RP 端数据大小分别为 95M、190M、285M、380M、475M 和 570M 时, RPKI 资料库数据未发生变化, RP 端和 RPKI 资料库同步, 数据未变化. 假设 Rhtsync 和 Rrsync 分别为 htsync 和 rsync 读的数据量(字节), Whtsync 和 Wrsync 分别为 htsync 和 rsync 写的数据量(字节), Thtsync 和 Trsync 分别为 htsync 和 rsync 的同步时间(秒). htsync 和 rsync 同步性能对比如表 4 所示.

表 4　同步性能对比

RP 端大小	95M	190M	285M	380M	475M	570M
Rhtsync	2160	3240	4320	5400	6480	7560
Rrsync	16	16	16	16	16	16
Whtsync	8	12	16	20	24	28
Wrsync	0.8M	1.5M	2.3M	3.0M	3.8M	4.5M
Thtsync	0.31	0.63	0.98	1.29	1.62	1.94
Trsync	0.32	0.65	1.00	1.33	1.70	2.06
加速比(%)	3.13	3.08	2.00	3.01	4.71	5.83

htsync 能够在传输较少的数据的情况下判断 RP 端和 RPKI 资料库数据的异同. 此种情况下, htsync 和 rsync 在同步时间上相差不多, 都能够比较迅速地完成同步. 随着 RP 端数据量的增大, htsync 同步时构建有序哈希树 buildHT 所需时间随之增多. 平均加速比为 3.63%.

4 结语

针对 RPKI 中 RP 端使用 rsync 进行数据同步的效率低下的问题, 本文结合 RPKI 资料库中文件(目录)的特点, 设计实现了一种基于有序哈希树的 RPKI 资料库同步工具 htsync, 改善了 RP 端与 RPKI 资料库的同步性能. 实验结果表明, 与 rsync 相比, htsync 在同步时的数据传输量较少, 同步时间也较短, 在设计的 3 种实验场景下, 同步时间平均加速比分别为 38.70%、30.13%和 3.63%, 有效地减少了同步时的时间和资源的消耗.

参考文献

1 马迪.RPKI 概览.电信网技术,2012,(9):30−33.

2 Lepinski M, Kent S. RFC 6480: An infrastructure to support secure internet routing. IETF, February 2012.

3 Lepinski M, Kent S. RFC 6482: An profile for route origin authorizations(ROAs). IETF, February 2012.

4 Huston G, Michaelson G. RFC 6483: Validation of route origination using the resource certificate public key infrastructure(PKI) and route origin authorizations (ROAs). IETF, February 2012.

5 Bruijnzeels T, Muravskiy O, Weber B, Austein R, Mandelberg D. draft-tbruijnzeels-sidr-delta-protocol, December 2014.

6 Tridgell A, Mackerras P. The rsync algorithm, TR-CS-96-05. Australia(Canberra). A. Tridgell. June 1996.

7 Huston G, Loomans R, Michaelson G. RFC 6481: A profile for resource certificate pepository structure. IETF, February 2012.

8 Lynn C, Kent S, Seo K. RFC 3779: X.509 extensions for IP addresses and AS identifiers. IETF, June 2004.

© 中国科学院软件研究所 http://www.c-s-a.org.cn