

Internet Technologien

Master-Arbeit

Entwurf und Analyse einer Methode zur Messung
der Konsistenz von RPKI Cache Servern und deren
Einfluss auf BGP

Samir Al-Sheikh

Matr. 4457630

Betreuer: Prof. Dr. Matthias Wählisch

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, sind als solche gekennzeichnet. Die Zeichnungen oder Abbildungen sind von mir selbst erstellt worden oder mit entsprechenden Quellennachweisen versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner Prüfungsbehörde eingereicht worden.

Berlin, den 23. November 2017

(Samir Al-Sheikh)

Abstract

Die vorliegende Arbeit analysiert die Konsistenz von Cache Servern innerhalb der RPKI-Infrastruktur und untersucht die Auswirkungen von Inkonsistenzen auf BGP. Ausgehend von der Architektur des Border Gateway Protocol (BGP) und der Resource Public Key Infrastructure (RPKI), wird das Problemfeld der Cache-Konsistenz hinsichtlich der genauen Definition der Konsistenz zwischen Cache Servern erläutert und das Verfahren zur Erhebung der Daten verifiziert. Nachfolgend wird die Cache Konsistenz für mehrere Cache Server in Form von Inkonsistenz-Analysen untersucht, die mithilfe der Datensätze der einzelnen Cache Server Inkonsistenzen quantifizieren. Um zu überprüfen, ob RPKI-Inkonsistenzen Einfluss auf die Vertrauenswürdigkeit auf BGP-Router und somit auf den BGP-Verkehr haben, wird die Auswirkung von möglichen Inkonsistenzen auf die BGP-Validierung genauer untersucht. Des Weiteren wird neben dem Entwurf und die Implementierung eines ROA-Dump-Archivs, das die Inkonsistenz-Analysen ermöglicht, auch das Design und die Implementierung einer Bibliothek (ROAFetchlib) zur Analyse der Auswirkungen auf BGP vorgestellt mithilfe der es möglich sein soll historische BGP-Daten mit historischen ROA-Daten zu validieren. Abschließend werden wichtige Aspekte für Problematiken und Lösungsansätze thematisiert.

Based on the architecture of the Border Gateway Protocol (BGP) and the Resource Public Key Infrastructure (RPKI), the problem area of cache consistency is explained with regard to the exact definition of consistency between Cache Servers and the procedure for collecting data is verified. The Cache consistency for multiple Cache Servers is then examined in the form of inconsistency analyses that quantify inconsistencies using the data records of each Cache Server. In order to check whether RPKI inconsistencies influence the reliability of BGP routers and thus the BGP traffic, the impact of possible inconsistencies on the BGP validation is examined in more detail. In addition to the design and implementation of a ROA dump archive that enables inconsistency analysis, the design and implementation of a library (ROAFetchlib) for impact analysis on BGP will also be presented, which will allow to validate historical BGP data with historical ROA data. Finally, important aspects of problems and solutions are discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis	11
Tabellenverzeichnis	13
Quellcodeverzeichnis	15
1 Einleitung	17
2 Hintergrund	19
2.1 Border Gateway Protocol (BGP)	20
2.1.1 Funktionsweise	20
2.1.2 Hijacking	20
2.2 Resource Public Key Infrastructure (RPKI)	21
2.2.1 Zertifizierungshierarchie	22
2.2.2 Globales RPKI Repository	23
2.2.3 RRPD und RSYNC	24
2.2.4 RPKI Cache Server	25
2.2.5 RTR-Protokoll	25
2.2.6 BGP-Router - Route Origin Validation (ROV)	25
3 Problemfeld der Cache-Konsistenz	27
3.1 Definition von Cache-Konsistenz	27
3.2 Mess-Spezifikation für die Untersuchung der Cache-Konsistenz	28
3.3 Verifikation des RPKI-Verhaltens	29
4 Analyse der Cache-Konsistenz	31
4.1 Meta-Analysen	31
4.1.1 Erreichbarkeit	32
4.1.2 Delay	35
4.2 Inkonsistenz-Analysen	41
4.2.1 Beacon Delay	41

4.2.2	Trust Anchor	44
4.2.3	ROA-Unterschiede	46
4.2.4	ROA-Unterschiede pro RIR	50
4.2.5	ROA-Unterschiede pro ASN	52
4.2.6	ROA-Unterschiede pro ASN - konsekutive Segmente	55
4.3	Zusammenfassung	58
5	Analyse der Auswirkungen der Inkonsistenzen auf BGP	61
5.1	Auswirkungen der Inkonsistenzen	61
5.2	Auswirkungen der Inkonsistenzen pro ASN	64
5.3	Zusammensetzung invalider Validierungsergebnisse	67
5.4	Zusammenfassung	69
6	Zwischenfazit: Cache-Inkonsistenz	71
7	Entwurf und Implementierung eines ROA-Dump-Archivs	73
7.1	Messinfrastruktur	73
7.2	Implementierung der ROA-Crawler	74
7.2.1	Konfiguration	74
7.2.2	Web-Sync	75
7.2.3	RTR-Sync	77
7.2.4	(Re-)Scheduling	79
7.2.5	Zeit-Management	82
7.3	Entwurf des ROA-Dump-Archivs	83
8	Entwurf und Implementierung einer Bibliothek zur Analyse der Auswirkungen auf BGP	85
8.1	BGPStream	85
8.2	ROA-Broker	87
8.2.1	Design	87
8.2.2	Funktionsweise und Implementierung	89
8.3	Bibliothek – ROAFetchlib	93
8.3.1	Design	93
8.3.2	API	94
8.3.3	Integration in BGPStream	95
8.3.4	Funktionsweise und Implementierung	97
9	Verwandte Arbeiten	105
9.1	BGP	105
9.2	RPKI	106

9.3 RPKI Cache Server Inkonsistenzen	106
10 Zusammenfassung und Ausblick	107
10.1 Zusammenfassung	107
10.2 Ausblick	108
Literaturverzeichnis	113

Abbildungsverzeichnis

2.1 RPKI-Übersicht	22
3.1 Unterschied zwischen Cache-Konsistenz und -Inkonsistenz	27
3.2 BGP-Auswirkungen einer Cache-Inkonsistenz	28
4.1 Übersicht der Analysen und deren Abhängigkeiten	31
4.2 Fehlerquellen für eine Nicht-Erreichbarkeit innerhalb der Messinfrastruktur	32
4.3 Messergebnisse der Delay-Analyse als Heat-Map in Sekunden [W-Web] . . .	36
4.4 Messergebnisse der Delay-Analyse in Sekunden [W-Web]	37
4.5 Messergebnisse der Delay-Analyse für alle Übertragungswägen [W-Web] . .	37
4.6 Unterschiedliche Möglichkeiten des zeitlichen Verlaufs einer Cache Server Anfrage	39
4.7 Legende für Boxplot	42
4.8 Messergebnisse der Beacon-Analyse in Minuten [R-RTR/ W-Web]	42
4.9 Messergebnisse der Trust-Anchor-Analyse	45
4.10 Messergebnisse der ROA-Unterschiedsanalyse für den Messzeitraum	47
4.11 Messergebnisse der ROA-Unterschiedsanalyse pro RIR [W-Web]	51
4.12 Messergebnisse der ROA-Unterschiedsanalyse pro ASN	53
4.13 Messergebnisse der konsekutiven Segmente innerhalb der ROA-Unterschiedsanalyse pro ASN	56
5.1 Messergebnisse der Analyse der BGP-Auswirkungen durch RPKI-Inkonsistenzen [R-RTR/ W-Web]	62
5.2 Messergebnisse der Analyse der BGP-Auswirkungen durch RPKI-Inkonsistenzen pro ASN	65
5.3 Messergebnisse der Analyse der Zusammensetzung invalider Validierungsergebnisse	68
7.1 Entwurf der Messinfrastruktur	73
7.2 Standorte der konfigurierten Kollektoren	75
7.3 Beispiel für den Ablauf des Next-Own-Timestamp-Scheduling	80
7.4 Beispiel für den Vorgang des Next-Common-Timestamp-Scheduling	81
7.5 Beispiel für Restart Scheduling	81
7.6 Zeit-Management	82
7.7 Entwurf des ROA-Dump-Archivs	83

8.1	BGPStream – Entwurf	86
8.2	Erweiterte Messinfrastruktur mit sequenziellem Broker-Design	87
8.3	Erweiterte Messinfrastruktur mit parallelem Broker-Design	88
8.4	JSON-Format des ROA-Brokers	92
8.5	Gegenüberstellung der Erweiterungsansätze für eine RPKI-Validierung	93
8.6	ROAFetchlib – Design	93
8.7	ROAFetchlib – API	94
8.8	ROAFetchlib – RPKI-Konfiguration-Struct	97
8.9	ROAFetchlib – Input-Konfiguration-Struct	97
8.10	ROAFetchlib – RTR-Konfiguration-Struct	98
8.11	ROAFetchlib – Broker-Konfiguration-Struct	98
8.12	ROAFetchlib – RPKI-Element	99
8.13	ROAFetchlib – Time-Konfiguration	101

Tabellenverzeichnis

3.1 Konfiguration der Mess-Spezifikation	29
4.1 Messergebnisse der Erreichbarkeitsanalyse [%]	33

Quellcodeverzeichnis

7.1	Start des Background-Schedulers des ROA-Crawlers	75
7.2	Fetching der ROA-Daten und Meta-Protokollierung des ROA-Crawlers . . .	76
7.3	Ausführung des entsprechenden (Re-)Schedulings des ROA-Crawlers	76
7.4	Meta-Protokollierung der Erreichbarkeit des RTR-Sync	77
7.5	Ausführung des RTR-Clients des Python-Bindings der RTRlib	78
7.6	Fetching der ROA-Daten aus der Präfix-Tabelle und Konsistenzüberprüfung	78
8.1	API-Aufruf (<code>rpki_set_config</code>) für die Erstellung einer RPKI-Konfiguration	96
8.2	API-Aufruf (<code>rpki_validate</code>) für RPKI-Validierung eines BGP-Elements . .	96
8.3	API-Methode (<code>rpki_set_config</code>) zum Setzen der RPKI-Konfiguration . . .	99
8.4	Befüllung der Hash-Tabelle “ <code>rpki_result</code> “ mit Validierungsergebnissen . . .	100
8.5	Historische Validierung – Validierungsaufruf der RTRlib	102
8.6	Hybrider Modus zum Wechsel des Validierungsmodus	103

KAPITEL 1

Einleitung

Seit seinen Anfängen vor mehr als 35 Jahren hat sich das Internet weltweit zu der erfolgreichsten Kommunikationsinfrastruktur entwickelt. Es verbindet inzwischen mehr als 100 Millionen Menschen miteinander und stellt für viele Länder eine kritische Infrastruktur dar.

Trotz dieser Entwicklung basiert das Internet auf Protokollen aus den Anfängen der Internet-Entwicklung. Damals waren die Dimensionen noch weitaus kleiner und überschaubarer, so dass beim Entwurf der Protokolle von vertrauenswürdigen Netzwerken ausgegangen wurde. Dementsprechend wurden diese Protokolle entwickelt, ohne Sicherheitsaspekte zu beachten. Diese Aspekte wurden mit der Zeit und der starken unübersichtlichen Ausbreitung des Internets ein Defizit dieser Protokolle, da weniger vertrauenswürdige Punkte Verbindungen anderer gewollt oder ungewollt durch beispielsweise Misskonfigurationen beeinflussen konnten.

Eines dieser Protokolle ist das Border Gateway Protocol (BGP) [1], dessen Aufgabe das Austauschen von Erreichbarkeitsinformationen zwischen diesen Netzwerken darstellt. Es gilt somit als Grundbaustein des Internets, dessen gesicherte Funktionsweise von äußerster Wichtigkeit für alle darauf aufbauenden Protokolle ist. Aus diesem Grund wurden über die Jahre neue Versionen entwickelt und versucht, das Protokoll an heutige Dimensionen anzupassen [2]. BGP unterstützt jedoch nicht die Prüfung der Integrität und Autorisierung des Präfixes. Um somit die zuvorgenannten Defizite auszugleichen, wurden weitere Protokolle entwickelt, die auf unterschiedliche Art und Weise versuchen, den Informationsaustausch hinreichend zu sichern.

Eines dieser Protokolle ist das Resource Public Key Infrastructure, kurz RPKI [3]. Mithilfe dieser Standardisierung wird eine Infrastruktur aufgebaut, die es erlaubt, die Autorisierung einer Internet Ressource (ein Netzwerk oder IP-Addressblocks) durch Zertifikate und öffentliche Schlüssel zu attestieren. Des Weiteren werden die Beziehungen dieser Ressourcen untereinander mithilfe von digital signierten Objekten festgehalten. Dadurch wird das bestehende BGP-Protokoll um eine Infrastruktur für eine transparente, für jede Instanz stets validierbare Vertrauenswürdigkeit erweitert.

Eine Schlüsselkomponente dieser Infrastruktur sind RPKI Cache Server, die einen Zwischenspeicher und eine Abstraktionsstufe zwischen der globalen RPKI-Signierungsarchitektur

und den BGP-Routern darstellen. Diese Server werden von Zertifizierungsstellen, wissenschaftlichen Institutionen aber auch von beliebigen Instanzen auf der gesamten Welt betrieben. Um BGP-Router mit einer einheitlichen Vertrauenswürdigkeit durch die RPKI-Signierungsarchitektur zu versorgen, ist die Konsistenz und Integrität dieser Server essentiell, da die Router auf Grundlage der Vertrauenswürdigkeit ihre Funktionsweise anpassen.

Zielsetzung der Arbeit: Diese Arbeit untersucht die Konsistenz bzw. Inkonsistenz zwischen dem Informationsstand von RPKI Cache Servern über einen bestimmten Messzeitraum. Dafür wurde eine Messinfrastruktur entwickelt, die eine periodische Speicherung des aktuellen Stands der Cache Server auf einem Archiv ermöglicht. Auf Basis dessen können folgend die gespeicherten Daten auf Inkonsistenz analysiert und die Ergebnisse ausgewertet werden. Des Weiteren werden in dieser Arbeit die Auswirkungen der Inkonsistenzen auf die BGP-Infrastruktur mithilfe einer entworfenen Bibliothek analysiert und ausgewertet. Die Fragestellung, ob inkonsistente RPKI Cache Server eine Auswirkung auf die Funktionsweise von BGP und somit auf die Sicherheit der darauf aufbauenden Protokolle haben, ist von äußerster Wichtigkeit. Inkonsistenzen zwischen den RPKI Cache Servern würde zu einer heterogenen Vertrauenswürdigkeit führen und somit den Nutzen des gesamten RPKI Protokolls verringern.

Aufbau der Arbeit: Das nachfolgende Kapitel 2 beschreibt die Architektur und Funktionsweise des Border Gateway Protokolls und der Resource Public Key Infrastructure. In Kapitel 3 wird die Cache-Konsistenz zunächst genauer betrachtet und definiert. Das darauffolgende Kapitel 4 beschreibt die dazugehörigen vorgenommenen Analysen und deren Auswertungen. Auswirkungen möglicher Inkonsistenzen auf BGP werden daraufhin im Kapitel 5 untersucht. Im Kapitel 7.7 werden der Entwurf und die Implementierung eines ROA-Dump-Archivs und die Funktionsweise von ROA-Crawler zur Befüllung dieses Archivs vorgestellt. Das Design und die Implementierung einer Bibliothek zur Validierung historischer BGp-Daten mit den entsprechenden ROA-Datensätzen ist Gegenstand des Kapitels 8. Das Kapitel 6 bewertet die Ergebnisse der Arbeit hinsichtlich deren Einfluss auf das BGP-Ökosystem. Mögliche weiterführende Betrachtungen und Lösungsansätze für die Vermeidung von Inkonsistenzen werden im Kapitel 10 beschrieben. Kapitel 10 fasst die Ergebnisse der vorliegenden Arbeit zusammen. Abschließend wird auf Untersuchungen mit thematischem Bezug verwiesen.

KAPITEL 2

Hintergrund

Die Kernaufgabe des Internets ist die Erreichbarkeit von Endgeräten unterschiedlicher Netze sicherzustellen. Einer der wichtigsten Grundbausteine ist dafür die Adressierung und die logische Verbindung von paket-orientierten Kommunikationsnetzwerken durch das Internet Protocol, kurz IP [4]. Die Adressierung der Entitäten innerhalb eines Netzwerks findet durch IP-Adressen statt, die aufgrund ihrer Eindeutigkeit durch zunächst 32-Bit (IPv4) und später 128-Bit (IPv6) Werte dargestellt werden. Ein IP-Adressblock – nachfolgend IP-Präfix genannt – ist eine Zusammenfassung von mehreren IP-Adressen, die sich eine bestimmte Anzahl an Bits (Mask-Length) teilen. Die gemeinsamen Bits adressieren das Netzwerk, und die übrigen Bits die einzelnen Entitäten. Ein IP-Präfix hat folgende Form: 10.11.12.0/24.

Die Vergabe der weltweit öffentlichen IP-Adressbereiche wird von der Internet Assigned Numbers Authority, kurz IANA, durchgeführt und organisiert. Diese delegiert die Adressierung weiter an fünf Regional Internet Registries, kurz RIRs, um anhand von geopolitischen Faktoren die Präfixe zu verteilen. Diese RIRs sind aktuell: RIPE (Europa, Asien), AfriNIC (Afrika), LACNIC (Südamerika), ARIN (Nordamerika), APNIC (Südostasien).

Die globalen IP-Adressen werden durch ein hierarchischen Modell organisiert. Weltweit verteilte Netzwerke können einen Bereich dieser IP-Adressen besitzen und diese innerhalb des Netzwerks weiter vergeben. Solche Netzwerke, Autonome Systeme (AS) [5] genannt, werden von Internet Service Providern, kurz ISPs, von großen internationalen Unternehmen und von wissenschaftlichen Institutionen, wie zum Beispiel Universitäten, betrieben. Für die Erreichbarkeit eines AS muss dieses mit weiteren AS verbunden sein und Erreichbarkeitsinformationen austauschen. Diese Informationen werden aufgrund der Änderungsrate durch ein dynamisches Routingprotokoll verteilt. Das AS besteht aus einer Vielzahl an Routern mit einem einheitlich verwendeten Protokoll zur Weiterleitung von Paketen innerhalb des Netzwerks. Ein solches AS tritt gegenüber anderen AS als einzelne Entität mit einem öffentlichen Router auf. Die AS werden durch eindeutige 16-Bit AS-Nummern (ASN) identifiziert. Für die Organisation dieser eindeutigen Nummern ist die IANA und die RIR zuständig. Um Informationen über die Erreichbarkeit von IP-Adressen zwischen den AS auszutauschen, wurde das Border Gateway Protocol entwickelt.

2.1 Border Gateway Protocol (BGP)

Das Border Gateway Protocol ist ein Pfad-Vektor-Protokoll, das die Kommunikation zwischen Autonomen Systemen standardisiert. Es stellt somit eine der wichtigsten Komponenten des Internets dar. Das heutige BGP liegt in Version BGPv4 vor und wurde im RFC 4271 definiert [2]. Im Gegensatz zu Interior-Gateway-Protokollen (IGP) müssen sich Exterior-Gateway-Protokollen (EGP) neben der Effizienz auch an politischen und wirtschaftlichen Richtlinien orientieren.

2.1.1 Funktionsweise

Um Erreichbarkeitsinformationen untereinander auszutauschen, versendet ein BGP-Router eines Autonomen Systems Update-Nachrichten an BGP-Router seiner benachbarten AS. Dadurch ist bekannt, welche IP-Präfixe über diesen erreichbar sind. Des Weiteren kann ein BGP-Router in Form eines Updates Informationen über IP-Präfixe geben (Announcement), die erreichbar, jedoch nicht Teil des eigenen AS sind. Das Announcement kann auch eine Liste von mehreren Hops (AS-Pfad) enthalten. Diese Liste beschreibt Erreichbarkeitsinformationen zu nicht-benachbarten AS. Das im AS-Pfad am rechten Ende stehende AS wird als Origin-AS bezeichnet. Dieser Pfad ist stets Teil eines Updates und beschreibt explizit den Routing-Weg, um einen IP-Präfix zu erreichen.

Wenn ein BGP-Router ein Update eines anderen AS erhält, kann er dieses, nachdem das eigene AS in den Pfad aufgenommen wurde, an seine unmittelbaren Nachbarn versenden. Dadurch verbreiten sich die Informationen zur Erreichbarkeit durch das gesamte Internet. Aus Effizienzgründen kann eine einzelne Update-Nachricht mehrere Routen beinhalten.

Welche erreichbaren IP-Präfixe ein AS von einem anderen annonciert und welche Route ein BGP-Router zum Erreichen eines Präfixes verwendet, hängt von Richtlinien ab. In der Regel sieht der Standard die kürzeste und somit effizienteste Route vor.

Da es keinen Mechanismus zur Überprüfung der Gültigkeit einer Update-Nachricht gibt, können sich falsche Informationen verbreiten. Dies kann aufgrund von fehlerhaften Konfigurationen des BGP-Routers aber auch mutwillig geschehen. Solch eine mutwillige Falsch-Information wird "Prefix Hijacking" genannt.

2.1.2 Hijacking

Es wird zwischen zwei verschiedenen Hijacking-Arten unterschieden: Prefix Hijacking und Subprefix Hijacking.

Das Prefix Hijacking beschreibt die fehlerhafte Bekanntgabe eines IP-Präfixes. Ein AS versendet in diesem Fall fälschliche Update-Nachrichten, in denen es als Besitzer des Präfixes angegeben ist. Dies kann dazu führen, dass andere AS ihre bestehende gültige Route verwerfen, die ungültige Route speichern und weiter verbreiten. Wenn die falsche Route akzeptiert wird, können Daten fort vom legitimen Besitzer und hin zum fälschlichen AS geleitet werden.

Das Routing innerhalb von BGP funktioniert nach dem *Longest Prefix Match* auf Basis der Zieladresse und den Routing-Einträgen. Dadurch wird stets der Routing-Eintrag gewählt,

der die längste Anzahl an gleichen aufeinanderfolgenden Bits mit der Zieladresse besitzt. Ein Routing-Eintrag eines allgemeinen Präfix-Hijack, der einen bestehenden IP-Präfix annonciert, steht dadurch in direkter Konkurrenz zu den bereits bestehenden Routing-Einträge. Zur Vermeidung dieser Konkurrenz wird ein kleinerer nicht-existierender Präfix (Sub-Präfix) eines größeren bereits bestehenden Präfixes verwendet. Dadurch ist der Routing-Eintrag des Sub-Präfixes der *Longest Prefix Match*.

Unter Subprefix Hijacking versteht man das fälschliche Aktualisieren von Erreichbarkeitsinformationen zu solch einem Sub-Präfix. Da dieser bevorzugt wird, wird diese Veränderung eine weitaus größere Reichweite erlangen und somit von mehr AS verwendet werden. Dies hat zur Folge, dass das AS, das den Hijack ausführt, Daten für diesen IP-Präfix bekommt. Mit diesen Daten kann es anschließend auf drei unterschiedliche Arten umgehen. Die Daten können verworfen werden und der Präfix somit unerreichbar sein. Das AS kann das legitimierte AS imitieren, um an vertrauenswürdige Informationen zu gelangen. Eine andere Möglichkeit ist, dass das AS in einer Man-in-the-middle-Position steht, in dem es die ankommenden Daten an das legitimierte AS weiterleitet, um diese zu manipulieren. Voraussetzung hierfür ist, dass das AS eine gültige Route zum legitimen AS besitzt.

Um solche Hijacking-Methoden zu verhindern und somit die Sicherheit und Vertrauenswürdigkeit des BGP-Austauschs zu erhöhen, wurde Resource Public Key Infrastructure (RPKI) entwickelt.

2.2 Resource Public Key Infrastructure (RPKI)

RPKI ist eine hierarchische Zertifizierungsinfrastruktur, deren Ziel die kryptografisch gesicherte Zuordnung autonomer Systeme und IP-Präfixe zum rechtmäßigen Besitzer ist. Des Weiteren beschreibt es welche autonomen Systeme autorisiert sind, bestimmte IP-Präfixe zu annoncieren. Dadurch können Prefix-Hijacks verhindert werden. Liegen diese Informationen auf den BGP-Routern vor, können diese falsche Update-Nachrichten erkennen und gegebenenfalls verwerfen.

Dafür besteht die RPKI aus vier Teilen (Abbildung 2.1):

- der Zertifizierungshierarchie, die zur Zertifikatsverteilung dient,
- dem globalen RPKI Repository, das alle Zertifikate und signierten Objekte enthält,
- den RPKI Cache Server, die die signierten Objekte validieren und deren Ergebnisse über legitime AS-Präfix-Kombinationen an die BGP-Router geschickt werden und
- den BGP-Routern, die mithilfe dieser Ergebnisse die Routing-Operationen anpassen.

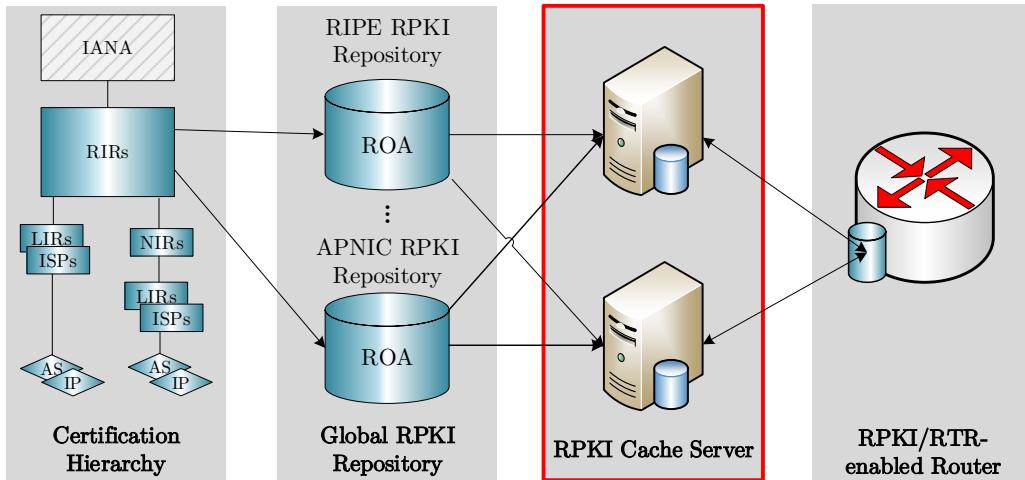


Abbildung 2.1: RPKI-Übersicht

2.2.1 Zertifizierungshierarchie

Die Zertifizierungshierarchie besteht aus zwei Kern-Datenstrukturen:

- RPKI-erweiterte Zertifikate (CA certificates),
- RPKI-signierte Objekte (RPKI signed objects).

RPKI-erweiterte Zertifikate

Ein RPKI-erweitertes Zertifikat dient dem Nachweis des Besitzes einer Internet Ressource. Es kann mithilfe eines öffentlichen Schlüssels stets von anderen auf Gültigkeit validiert werden. Solch ein Zertifikat basiert auf dem X.509-Standard [6] und wird durch die zwei RPKI-spezifischen Felder “IP Resources“ und “AS Resources“ erweitert. Die Zusammenführung beider Ressourcen mithilfe einer Public Key Infrastruktur ist nur möglich, da beide Ressourcen eine ähnliche Hierarchie haben.

Die in RPKI verwendete Zertifikatshierarchie folgt der Vergabe von IP-Präfixe, so dass die höchste Instanz dieser Hierarchie die IANA bildet, gefolgt von den fünf Regional- (RIRs), National- (NIRs) und den Local Internet Registries (LIRs), die jeweils öffentliche Zertifikate und Schlüssel besitzen. Anschließend folgen Internet Server Provider (ISPs), die selbst oder in Kooperation miteinander ein oder mehrere AS oder IP-Präfixe besitzen. Diese erhalten von den öffentlichen Instanzen ähnliche Zertifikate, die für die Ausstellung weiterer Zertifikate in ihrem AS bzw. IP-Präfix gedacht sind.

Die Voraussetzung ausgestellter Zertifikate und analog deren Ressourcen ist, dass diese stets eine Teilmenge des Zertifikats bzw. der Ressource des Zertifikat-Ausstellers sind. Es entsteht somit eine Zertifikatkette, die explizit den Verlauf der Zertifikat-Hierarchie für ein beliebiges Zertifikat attestiert. Im Gegensatz zu anderen Public Key Infrastrukturen wird hierbei ausschließlich die gültige Beziehung zwischen Besitzer und Ressource - Autorisierung (authorization) - sichergestellt, jedoch nicht wie üblich die Identität des Besitzers (authentication).

Im Umkehrschluss werden somit die Public-Keys der höheren Instanzen benötigt, um ein beliebiges Zertifikat auf dessen Richtigkeit zu prüfen. Die Zertifikate bzw. Public-Keys der höchsten Instanzen werden als Trust Anchor bezeichnet, da deren Gültigkeit stets angenommen wird. Der Speicherort dieser Zertifikate wird für darunterliegende Instanzen mithilfe des Trust Anchor Locator (TAL) angegeben. Diese beinhalten die URI des Zertifikates der Instanz und den Public Key.

Die beschriebene Infrastruktur ist notwendig, um die Gültigkeit einzelner Ressourcen zu attestieren. Es fehlen jedoch die Beziehungen von AS zu IP-Präfixen, um unterscheiden zu können, ob ein AS die Berechtigung besitzt, einen bestimmten IP-Präfix zu annoncieren. Diese Beziehung kann einer 1-zu-1- und 1-zu-N-Beziehung folgen, so dass mehrere AS einen Präfix annoncieren dürfen. Realisiert wird die Zuordnung dieser Beziehungen durch das Route Origin Authorization (ROA).

RPKI-signierte Objekte (RPKI signed objects)

RPKI-signierte Objekte sind digital signierte, nicht auf dem X.509-Standard basierende Datenstrukturen, die jedoch einer für RPKI standardisierten Vorlage entsprechen [7]. Ein Teil des Objektes ist laut Vorlage ein eindeutiges Zertifikat, das RPKI End-Entität Zertifikat, kurz EE, das benutzt wurde, um dieses Objekt auszustellen, und stets die Gültigkeit des Objekts attestierte. Dies ermöglicht das Zurückziehen von Zertifikaten und Objekten, wodurch deren Gültigkeit endet.

Ein ROA ist ein RPKI-signiertes Objekt und beinhaltet neben dem eindeutigen EE-Zertifikat den öffentlichen Schlüssel, die Signatur, den Aussteller und die Zuordnung von ASN und IP-Präfix. Solch eine Zuordnung wird durch drei Felder definiert: ASN, IP-Präfix und die maximale Länge des Präfixes. Eine mögliche ASN-IP-Präfix-Zuordnung könnte somit die folgende sein:

10111, 10.11.12.0/16, 24.

Ein valides ROA ist somit ein der Vorlage entsprechendes RPKI-signiertes Objekt, das ein valides (nicht zurückgezogenes) Zertifikat enthält und dessen Präfixe Teil des Zertifikates sind. Mithilfe dieses ROA kann somit entschieden werden, ob ein AS autorisiert ist, einen bestimmten IP-Präfix zu annoncieren.

2.2.2 Globales RPKI Repository

RPKI-erweiterte Zertifikate und signierte Objekte werden in frei verfügbaren verteilten globalen RPKI Repositories gespeichert, da sie stets benötigt werden, um die Gültigkeit von Beziehungen und den Ressourcen-Besitz zu prüfen. Diese Repositories können von RIRs, NIRs, LIRs, großen internationalen Unternehmen oder ISPs geführt werden, die in ihrer Gesamtheit das globale RPKI Repository erzeugen.

Da die kryptografische Validierung der ROA und Zertifikate sehr speicher- und rechenintensiv ist, können BGP-Router diese Aufgabe momentan nicht direkt umsetzen. Um die Router zu entlasten, wird eine weitere Zwischenstufe zwischen ihnen und dem globalen RPKI Repository benötigt, den RPKI Cache Servern. Damit diese Cache Server die für die Validierung notwendigen Daten erhalten, gibt es unterschiedliche Protokolle, die zur Synchronisierung

verwendet werden.

2.2.3 RRPD und RSYNC

Es gibt zwei verschiedene Arten der Synchronisierung zwischen dem globalen RPKI Repository und den einzelnen Cache Servern: RSYNC und RRDP.

Das RSYNC-Protokoll [8] ist ein herkömmliches Synchronisierungsprotokoll, das Dateien zwischen zwei Entitäten abgleicht. Dafür wird ein für das Protokoll entworfenes URI-Schema verwendet, das das Erreichen externer Datenquellen ermöglicht. Mithilfe verschiedener Optimierungsalgorithmen werden so ausschließlich sich in Größe und Änderungszeit veränderte Dateien (Quick-Check Algorithmus) und darauf aufbauend ausschließlich sich zwischen zwei Iterationen verändernde Teile dieser Dateien (Delta-Transfer Algorithmus) abgeglichen und ausgetauscht. Dieses Verfahren ermöglicht, jedem Cache Server, in einem bestimmten Intervall eine vollständige Synchronisierung der im globalen Repository befindlichen geänderten Daten bzw. Datenteile vorzunehmen.

Das RPKI Retrieval Delta Protocol (RRDP) [9] ist ein Protokoll, das ausschließlich für die Synchronisierung von RPKI-Validierungsdaten entworfen wurde. Es gibt den Cache Servern, einen Mechanismus zum gesicherten (HTTPS) Anfordern von inkrementellen Updates vom globalen Repository.

Des Weiteren ermöglicht das RRDP im Gegensatz zum RSYNC einen State-orientierten Ansatz, so dass der Zustand jedes Cache Servers auf einem entsprechenden RRDP-Deamon des globalen Repositories gespeichert wird. Um dies umzusetzen, gibt es für jeden Cache Server ein "Notification-File", das eine global einzigartige Identifikation (Session ID) und eine Serial Number enthält. Damit ist es dem Cache Server möglich zu bestimmen, ob dieser mit dem globalen Repository synchronisiert und somit auf dem aktuellen Stand ist.

Des Weiteren enthält es einen Link zum letzten Snapshot, der einen absolut vollständigen Datensatz beschreibt, und eine Liste an geänderten Dateien (Delta Files), die sich seit der letzten Synchronisierung verändert haben. Anhand der Serial Number kann das globale Repository somit den letzten Stand des Cache Servers ausfindig machen. Somit gilt das Notification-File als datei-basierte Übersicht über die Veränderungen seit der letzten Synchronisierung. Diese Datei kann durch ein Fetching in regelmäßigen periodisierten Abständen vom globalen Repository geladen werden.

Im Falle einer geänderten Serial Number oder einer nicht kontinuierlich fortlaufenden Kette an Delta Files kann der Cache Server den letzten Snapshot des globalen Repository laden und somit eine vollständige Re-Synchronisierung einleiten.

Nachdem eine vollständige Synchronisierung durch einen Snapshot stattgefunden hat, kann der jeweilige Cache Server mit der Validierung der ROA-Daten beginnen. Im Falle von mehreren Notification-Files wartet der Cache Server bis sämtliche Notification-Files verarbeitet wurden, bevor es zu einer Validierung kommen kann. Aufgrund der Architektur des Protokolls ist es möglich, aus Performance- und Lastgründen die Snapshots und Delta-Files auf dem Weg vom globalen Repository zum Cache Server zwischenspeichern [9].

2.2.4 RPKI Cache Server

Das globale RPKI Repository trägt sämtliche RPKI-relevanten Objekte und Zertifikate in sich, wobei diese nicht auf Gültigkeit bzw. Rückzug validiert werden. Die Aufgabe der RPKI Cache Server ist das Fetchen und Validieren der RPKI-signierten Objekte und deren Zertifikate mithilfe der vom TAL zur Verfügung gestellten öffentlichen Zertifikate des RIR.

Anhand der Validierung können zu diesem Zeitpunkt valide ROA, die für BGP-Router als vereinfachte Form (ASCII) dargestellt sind, zusammengetragen werden. Somit bilden die RPKI Cache Server einen Zwischenspeicher und eine weitere Abstraktionsstufe, um den BGP-Routern auf möglichst rechen- und speicherarme Weise die Validierung von BGP-Announcements zu ermöglichen. Es ergeben sich somit zwei verschiedene Validierungsschritte: Zum einen die Validierung des Cache Servers, um die validen ROA zu identifizieren und zum anderen die Validierung der BGP-Announcements mithilfe dieser validen ROA auf dem BGP-Router.

Die vereinfachte ASCII-Form der ROA wird ebenfalls als ROA bezeichnet. Diese enthält ausschließlich die für die Zuordnung relevanten Daten: ASN, Präfix und maximale Länge. Ähnlich wie bei den Peers in BGP ist die Wahl des Cache Servers eine Vertrauensfrage. Diese vereinfachte Form enthält alle atomaren Einträge aller ROA. Ein Eintrag wird nachfolgend als ROA-Eintrag oder kurz ROA bezeichnet.

Der Austausch der ROA-Daten zwischen Cache Server und Router erfolgt über das RTR-Protokoll.

2.2.5 RTR-Protokoll

Das RTR-Protokoll [10] ermöglicht einen effizienten Zugang zu kryptografisch validierten ROA eines Cache Servers. Dazu verbindet sich der BGP-Router mit einem für ihn bevorzugten oder mehreren bevorzugten Cache Servern. Das RTR-Protokoll erlaubt inkrementelle Updates zwischen Cache Server und Router.

Mit einem End-of-Data PDU wird dem Router signalisiert, dass dieser bereits auf dem aktuellen Stand ist, und setzt anschließend seine Serial Number auf die des PDUs. Ein Reset Query verwirft den aktuellen Stand des Routers, um eine neue Verbindung zu einem Cache Server aufzubauen oder die aktuelle Beziehung zurückzusetzen.

Sobald der Cache Server seine ROA-Datenbank erneuert hat, in dem erneut ein Fetching und die Validierung durchgeführt wird, schickt dieser ein Notify, um alle verbundenen Router zu benachrichtigen. Je nach Konfiguration erfragt der Router daraufhin erneut neue Daten von dem Cache Server oder wartet bis zum periodischen Abfragezeitpunkt. Durch dieses inkrementelle Update ist ein effizienter Austausch von validen ROA möglich [10].

Da die aktuelle Version der Cache Server Software (RIPE Validator), dieses inkrementelle Update nicht unterstützt, wird im Folgenden ausschließlich von periodischen Abfragen ausgegangen.

2.2.6 BGP-Router - Route Origin Validation (ROV)

Nachdem der BGP-Router durch das RTR-Protokoll die neuesten ROA-Datensätze bekommen hat, kann dieser die Validierung der BGP-Announcements vornehmen. Die ROA-Datensätze spiegeln dabei die aktuell erlaubten Präfix-ASN-Beziehungen wider. Der BGP-

Router muss nun prüfen, ob die in einem BGP-Announcement vorkommende Kombination von Origin ASN (nachfolgend nur ASN genannt) und Präfix enthalten ist. Die Route Origin Validation kann zu folgenden drei Zuständen führen: autorisiert (und somit valide) oder nicht autorisiert (und somit invalide) oder nicht gefunden.

Eine Präfix-ASN-Kombination ist valide, falls es im aktuellen ROA-Datensatz einen Eintrag gibt, der exakt dieses ASN trägt, und der Präfix Teil der erlaubten Präfix-Range dieses Eintrags ist. Dabei ist zu beachten, dass jeder Eintrag, der ebenfalls dieses ASN und ein Covering-Präfix des zu validierenden Präfixes trägt, ebenfalls den validen Zustand erhält. Ein Covering-Präfix ist dabei ein Routing-Eintrag der mit dem Präfix übereinstimmt, jedoch nicht der *Longest Prefix Match* ist. In diesem Fall ist das ASN autorisiert, diesen Präfix zu annoncieren.

ROA-Datensatz:	10111,10.11.12.0/16,24
BGP-Announcement:	10111,10.11.12.0/16
Validierungszustand:	valide

Eine Präfix-ASN-Kombination gilt als invalide, falls es explizit einen Eintrag im gesamten ROA-Datensatz gibt, der zu dem zu validierenden Präfix ein anderes ASN inne trägt. Ist dies der Fall, so ist das ASN aus dem BGP-Announcement nicht autorisiert, diesen Präfix zu annoncieren.

ROA-Datensatz:	10111,10.11.12.0/16,24
BGP-Announcement #1:	10110,10.11.12.0/16
BGP-Announcement #2:	10111,10.11.12.0/32
Validierungszustand:	invalid

Ein Präfix und somit ebenfalls eine Präfix-ASN-Kombination gilt als “notfound“, falls der zu validierende Präfix nicht Teil des ROA-Datensatzes ist. In diesem Fall gibt es zu diesem Zeitpunkt für diesen Präfix keine Auskunft über die Autorisierung bekannter ASN dieses Präfixes.

ROA-Datensatz:	10111,10.11.12.0/16,24
BGP-Announcement:	10111,20.11.12.0/16
Validierungszustand:	notfound

Die Korrektheit, Integrität und Konsistenz der ROA-Datensätze aller Cache Server ist für die Validierung von BGP-Announcements von größter Bedeutung.

Nachfolgend wird näher erläutert, welche Eigenschaften zur Überprüfung der Cache-Konsistenz herangezogen werden können und wie diese definiert sind.

KAPITEL 3

Problemfeld der Cache-Konsistenz

Für die Untersuchung der Cache-(In)konsistenz ist eine genaue Definition notwendig. Nachfolgend wird beschrieben in welchen Fällen es sich um ein konsistentes Cache-Server-Verhalten handelt und in welchen Fällen Inkonsistenzen vorliegen.

3.1 Definition von Cache-Konsistenz

Um eine global einheitliche Validierung der BGP-Router zu ermöglichen, müssen die ROA-Datensätze der einzelnen Cache Server stets gleich sein. Zwei Cache Server sind konsistent zueinander, wenn sie zu einem bestimmten Zeitpunkt eine identische Menge an ROA-Datensätzen zur Verfügung stellen. Der Zustand eines Cache Servers ist dabei die aktuelle Menge an ROA-Datensätzen. Server, die zu einem bestimmten Zeitpunkt konsistent sind, müssen dies nicht über einen längeren Zeitraum sein.

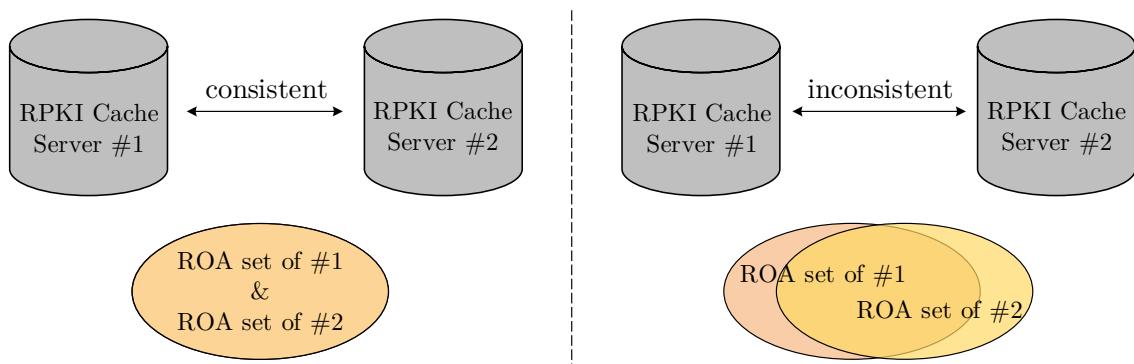


Abbildung 3.1: Unterschied zwischen Cache-Konsistenz und -Inkonsistenz

Zwei Cache Server gelten als inkonsistent, wenn sich die ROA-Datensätze innerhalb des Zeitraums unterscheiden. Diese Inkonsistenzen haben in den meisten Fällen direkte Auswirkungen auf die BGP-Validierung (siehe Abbildung 3.2).

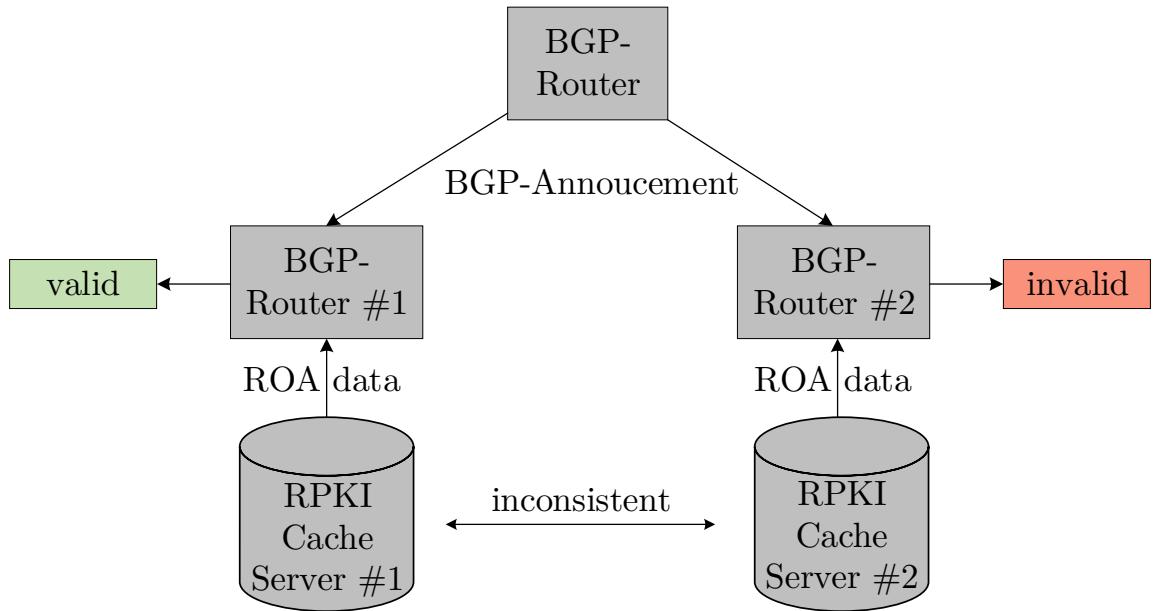


Abbildung 3.2: BGP-Auswirkungen einer Cache-Inkonsistenz

Dennoch können BGP-Router, die jeweils zu den inkonsistenten Cache Servern verbunden sind, gleiche Validierungsergebnisse liefern, wenn (i) die Validierung nicht von den Inkonsistenzen betroffen ist oder (ii) wenn zwei Inkonsistenzen zum gleichen Validierungsergebnis führen. Dadurch wird deutlich, dass die Anzahl an Inkonsistenzen unerheblich sein kann, da die einzelnen ROA-Datensätze und deren ASN-Präfix-Kombinationen unterschiedliche Auswirkungen haben können. So können viele Inkonsistenzen eines Präfixes durch (i) und (ii) weniger invalide Zustände hervorrufen als eine Inkonsistenz eines anderen Präfixes mit. Das heißt, der Grad der Inkonsistenz steht nicht in direkter Relation zur Menge der Inkonsistenzen.

Nachfolgend wird näher erläutert, welche Gegebenheiten für die Analyse der Inkonsistenzen erfüllt sein müssen.

3.2 Mess-Spezifikation für die Untersuchung der Cache-Konsistenz

Für die Identifizierung von Cache-Inkonsistenzen ist es wichtig, dass ausschließlich diese analysiert werden, unabhängig von anderen Nebeneffekten. Dazu dürfen nur die ROA-Datensätze der Cache Server herangezogen werden.

Folgende Spezifikationen müssen erfüllt sein, um ausschließlich Inkonsistenzen des RPKI-Verhaltens zu ermitteln:

1. Die Analyse jedes Cache Servers erstreckt sich über einen einheitlichen Zeitraum.
2. Die Verfügbarkeit eines Cache Servers sollte oberhalb einer Toleranzgrenze liegen, um zuverlässige Messergebnisse zu erhalten.
3. Die Zeitdifferenz zwischen zwei Zuständen ist für den gesamten Messzeitraum einheitlich und konstant.

4. Zur Untersuchung der Konsistenz wird ausschließlich eine bestimmte Menge an Trust Anchor verwendet, so dass die zu validierenden ROA des globalen RPKI Repository uniform sind.
5. Aufgrund des Fehlens eines standardmäßigen allzeit konsistenten Cache Servers, wird dieser mithilfe einer der konfigurierten Server für den gesamten Zeitraum simuliert.

Konfiguration

1. Zeitraum	2. Verfügbarkeit	3. Intervall	4. Trustanchor	5. Konsistenz-Kollektor
01.08 - 31.08.2017	90%	180 Sekunden	Trustanchor der RIR	CC01(Web)

Tabelle 3.1: Konfiguration der Mess-Spezifikation

Nachfolgend wird die Methode zur Verifikation des RPKI-Verhaltens vorgestellt.

3.3 Verifikation des RPKI-Verhaltens

Für die Aussagekraft der Analysen und die Interpretation der Messergebnisse ist es von äußerster Wichtigkeit, dass die erhobenen Daten auch die Realität der aktuell aktiven RPKI widerspiegeln. Um dies zu überprüfen, wird eine Methode verwendet, die eigens erstellte *Beacons* in die aktive RPKI einspeist. So ist es möglich zu prüfen, ob eigene Änderungen dieser Beacons in der RPKI zu dem per Standard definierten Verhalten führen und dies bei den einzelnen Cache Servern sichtbar ist. Dafür werden Zertifikate und ROA für bestimmte ASN-Präfix-Kombinationen ausschließlich für diesen Zweck verwendet, so dass keine Nebeneffekte interferieren. Diese ROA-Beacons werden im Gegensatz zu herkömmlichen ROA eines RIR nicht vom RIR selbst verwaltet, sondern werden als externes Repository in das des RIR eingespeist. Um nun Änderungen dieser ROA zu simulieren, werden die Zertifikate der ROA-Beacons in regelmäßigen Abständen periodisch zurückgezogen und daraufhin erneut ausgestellt.

Da dieser Fall eintritt, spiegeln die erhobenen Daten der Cache Server die Realität der RPKI wider. So ist verifiziert, dass sich die Messergebnisse der Analysen auf die aktuelle RPKI beziehen.

Nachfolgend werden die einzelnen Analysen, deren Messmethodik, Durchführung und Auswertung vorgestellt und näher erläutert.

KAPITEL 4

Analyse der Cache-Konsistenz

In diesem Kapitel werden die einzelnen Analysen zur Cache-Konsistenz vorgestellt. Dafür wird jede Analyse hinsichtlich Messmethodik und Umsetzung erläutert und die Messergebnisse in der Auswertung interpretiert. Die Abbildung 4.1 zeigt eine Übersicht aller Analysen und deren Abhängigkeiten untereinander.

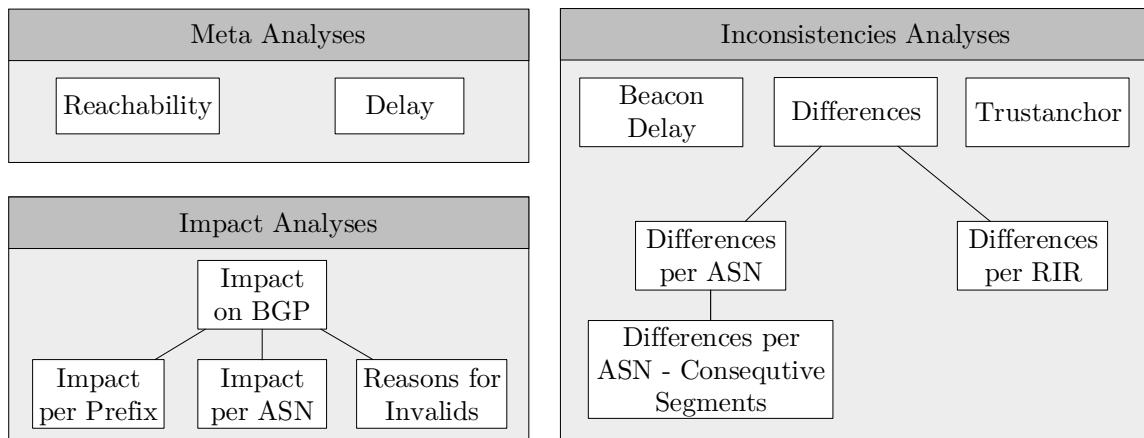


Abbildung 4.1: Übersicht der Analysen und deren Abhängigkeiten

Nachfolgend werden die einzelnen Meta-Analysen gefolgt von den Inkonsistenz-Analysen vorgestellt.

4.1 Meta-Analysen

Die Meta-Analysen versuchen mithilfe von Meta-Informationen wie Erreichbarkeit, Verzögerung (Delay) und Vollständigkeit der Daten mögliche Nebeneffekte für nachfolgende Analysen näher zu beleuchten und diese anschließend bestmöglich auszuschließen. Die Meta-Analysen ermöglichen neben der Abgrenzung weitergehend eine Untersuchung der möglichen indirekten RPKI-Inkonsistenzen, die durch einen Ausfall oder eine Verzögerung ausgelöst werden können.

4.1.1 Erreichbarkeit

Die Meta-Analyse der Erreichbarkeit untersucht, ob und inwiefern ein Cache Server und dessen ROA-Datensätze über den gesamten Messzeitraum erreichbar sind. Dabei wird hinsichtlich der Nicht-Erreichbarkeit unterschieden in:

- Server ist nicht erreichbar,
- Client ist nicht erreichbar,
- ROA-Datensatz ist teilweise unvollständig.

Für die RPKI ergeben sich die in Abbildung 4.2 beschriebenen Nicht-Erreichbarkeiten.

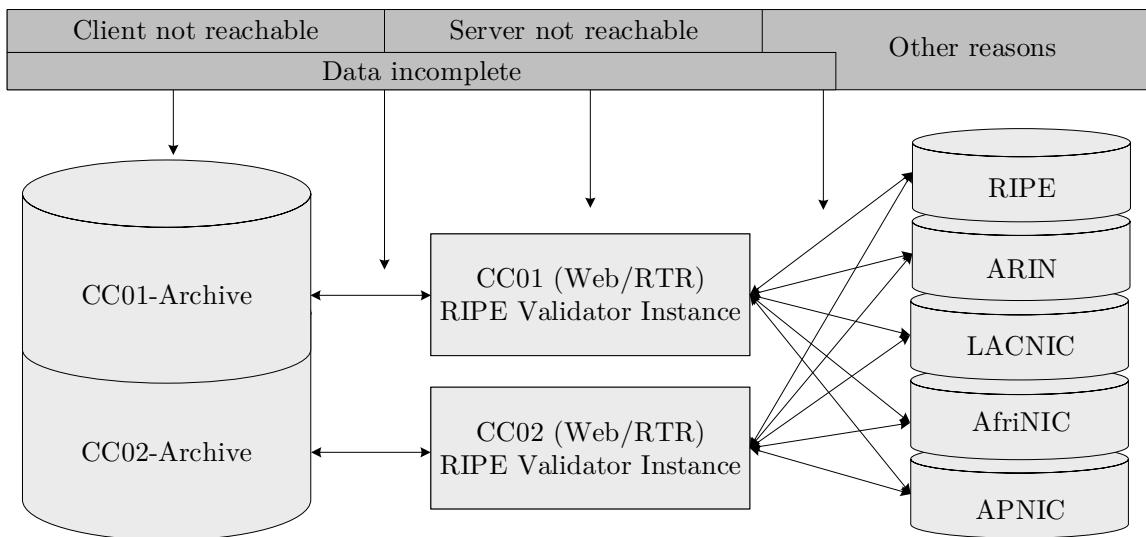


Abbildung 4.2: Fehlerquellen für eine Nicht-Erreichbarkeit innerhalb der Messinfrastruktur

Anhand der erhobenen Daten können Rückschlüsse auf indirekte RPKI-Inkonsistenzen durch den Ausfall eines Cache Servers gezogen werden.

Nachfolgend wird die zur Feststellung der Erreichbarkeit angewandte Messmethodik vorgestellt.

Messmethodik und Umsetzung

Die für die Analyse notwendigen Daten werden für jeden Cache Server während der Beschaffung der ROA-Daten protokolliert. Dabei werden Cache Server beider Interfaces (Web/RTR) für die Messung herangezogen. Beide Interfaces werden einzeln hinsichtlich der Nicht-Erreichbarkeit geprüft.

Im Falle des Web-Interface wird der Grund für die fehlende Erreichbarkeit des Servers oder des Clients anhand des HTTP- oder URL-Fehlercodes identifiziert. Falls die Adresse nicht aufgelöst werden kann, handelt es sich vorübergehend um einen Fehler seitens des Clients in Form einer fehlenden ausgehenden Verbindung. Bei Unerreichbarkeit des Servers über einen URL-Fehlercode wird dies entsprechend übernommen. Diese gilt ebenfalls bei einem

HTTP-Fehler, da jede mögliche Fehlerquelle durch den Server verursacht wird. Während der Speicherung der ROA-Daten wird die Vollständigkeit der Daten überprüft, wobei auch ein leerer ROA-Datensatz protokolliert wird.

Im Falle des RTR-Interface werden die jeweiligen Fehlercodes der RTRlib verwendet, um die Ursachen der fehlenden Erreichbarkeit zu identifizieren. Bei einem Transportfehler wird von einer client-seitigen Unerreichbarkeit ausgegangen. Falls die RTRlib sich nicht zum Server verbinden kann, wird eine fehlende Erreichbarkeit des Servers protokolliert. Da die erhobenen Daten via RTRlib je nach IP-Version betrachtet werden, kann die Vollständigkeit der Daten analog untersucht werden. Bei Fehlen sämtlicher Daten einer oder beider IP-Versionen wird der aktuelle ROA-Datensatz als fehlerhaft eingestuft.

Die protokollierten Daten werden aufgearbeitet und auf den Messzeitraum beschränkt. Die einzelnen Ursachen jedes Cache Servers für die Nicht-Erreichbarkeit werden gezählt und ins Verhältnis zu der Anzahl aller Messungen gesetzt.

Nachfolgend werden diese Daten vorgestellt, erläutert und hinsichtlich indirekter RPKI-Inkonsistenzen analysiert.

Messergebnisse

Die Messergebnisse (Tabelle 4.1) spiegeln das prozentuale Verhältnis der jeweiligen Nicht-Erreichbarkeit des Cache Servers (Cache Collector – CC) über den gesamten Messzeitraum wider.

Kollektor	Server unerreichbar	Client unerreichbar	Daten unvollständig
CC01(RTR)	0,01	—	0,07
CC01(Web)	0,01	—	—
CC02(RTR)	0,01	0,03	0,78
CC02(Web)	0,03	—	0,02
CC03(RTR)	0,01	0,05	—
CC03(Web)	0,02	—	—
CC04(RTR)	9,26	—	0,74
CC04(Web)	9,26	—	—
CC07(Web)	0,01	—	—

Tabelle 4.1: Messergebnisse der Erreichbarkeitsanalyse [%]

Die Nicht-Erreichbarkeit schwankt zwischen Werten von 0,01 - 9,26%. Die Unvollständigkeit der jeweiligen ROA-Datensätze liegt in einem Bereich von 0,02 - 0,78%. Fehlende Einträge markieren eine vollständige Erreichbarkeit bzw. die Vollständigkeit aller Datensätze. Die Messdaten zeigen, dass kein Cache Server eine vollständige Erreichbarkeit während des Messzeitraums aufweisen konnte.

Auswertung

Die Web-Version des Kollektor-Paars CC01 war während der gesamten Messung äußerst stabil erreichbar, der Client verfügbar und die übertragenen Daten stets vollständig. Da es sich hierbei um den Referenz-Kollektor handelt, sind die Daten für folgenden Messungen zuverlässig. Ähnlich stabil war die Web-Version des Kollektors CC07, die gleiche Messergebnisse aufweist.

Die RTR-Version des Kollektors CC01 hat eine Unvollständigkeit der Daten von 0,07%. Dies entspricht einem Ausfall von rund 10 von 14400 möglichen ROA-Datensätzen. Die Unvollständigkeit der Daten ist dabei – durch Nachforschungen während der Messung – auf den bereits erwähnten Fehler des RIPE Validators (Version 2.23) zurückzuführen. Dieser Fehler verhindert die Unterstützung inkrementeller Updates, wodurch der Kollektor nicht in der Lage ist, ausschließlich geänderten ROA-Daten an den BGP-Router zu senden. Der Cache Server versendet in diesem Fall an alle verbundenen BGP-Router ein Cache Reset. Die BGP-Router löschen nach Standard daraufhin den aktuellen ROA-Datensatz. Nach einem weiteren Aktualisierungsversuch wird der ROA-Datensatz des BGP-Routers erneut schrittweise aufgebaut. Wenn ein Mitschnitt während dieses Vorgangs erstellt wird, kommt es zu Unvollständigkeiten der ROA-Daten im Archiv. Durch diesen Fehler scheint es fälschlicherweise zu RPKI-Inkonsistenzen zwischen den Cache Servern zu kommen, da zu einem bestimmten Zeitpunkt ein Cache Server einen vollständigen ROA-Datensatz besitzt, obwohl zeitgleich ein anderer Cache Server einen unvollständigen ROA-Datensatz hat. Dies ist eine Auswirkung des fehlerhaften Verhaltens des RTR-Servers innerhalb des RIPE Validators. Aufgrund dieses Messfehlers werden hier und im Folgenden ROA-Datensätze, die aufgrund des Fehlers unvollständig sind, als Messartefakte definiert und für nachfolgende Analysen ausgeschlossen.

Die Kollektor-Paare CC02 und CC03 weisen während der Messung eine minimal erhöhte Nicht-Erreichbarkeit von 0,01 - 0,03% des Servers und 0,03% - 0,05% des Clients auf. Da sich die fehlende Konnektivität der beiden Kollektor-Paare auf die jeweiligen RTR-Versionen beschränkt, wird von einem fehlerhaften Verhalten des RTR-Clients ausgegangen.

Auffällig ist im Falle von CC02 und CC04 eine erhöhte Unvollständigkeit der Daten von 0,78 % und 0,74%. Dies entspricht einer absoluten Menge von rund 112 bzw. 107 unvollständigen ROA-Datensätzen innerhalb des Messzeitraums. Diese Unvollständigkeit lässt sich ebenfalls auf den Fehler des RIPE Validators zurückführen. Die Auswirkung des Fehlers ist auf allen RTR-Versionen aller Kollektor-Paare bis auf CC03 zu erkennen, da sich dieser Fehler der Protokoll-Umsetzung erst in Verbindung mit der Version 2.23 des RPKI-Validators äußert. Da sich das Kollektor-Paar zum Zeitpunkt der Arbeit auf Version 2.17 befindet, äußert sich dieser Fehler dort nicht.

Neben der erhöhten Unvollständigkeit der RTR-Version des Kollektor-Paars CC04 ist außerdem dessen stark auftretende fehlende Konnektivität des Servers von 9,26% auffällig.

Dies entspricht einer Ausfallzeit von rund 2,8 von 31 Tagen und 1334 fehlenden von 14400 möglichen ROA-Datensätzen innerhalb des Messzeitraums. Je nach Fragmentierung der Nicht-Erreichbarkeit kann dies zu einem langanhaltenden Ausfall oder zu mehreren kurzzeitigen Ausfällen kommen. Ein gravierender Ausfall tritt auf, wenn viele Aktualisierungsversuche des BGP-Routers übersprungen werden. In diesem Fall würde ein BGP-Router, der ausschließlich diesen Server als Quelle konfiguriert hat, während des Ausfalls einen leeren ROA-Datensatz besitzen. Dies hätte zur Folge, dass alle BGP-Announcements als "not-found" validiert werden. Dieser BGP-Router hätte folglich ein anderes Validierungsergebnis als einer, der mit einem aktiven Cache Server mit gleichem ROA-Datensatz verbunden ist.

Dabei handelt es sich um eine indirekte Inkonsistenz, die jedoch nicht aufgrund unterschiedlicher Zustände der ROA-Daten der Server verursacht wird, sondern ausschließlich durch inkorrekte Warten des Cache Servers. Diese Inkonsistenzen werden daher in der Arbeit nicht weiter thematisiert.

4.1.2 Delay

Die Meta-Analyse der Übertragungsverzögerung (Delay) untersucht die Übertragungsdauer der ROA-Datensätze für konfigurierte Cache Server über den gesamten Messzeitraum. Die Eigenschaften des RTR-Protokolls verhindern eine Messung der Übertragungsdauer für einen bestimmten ROA-Datensatz. Daher beschränkt sich diese Analyse ausschließlich auf die Web-Versionen der Kollektor-Paare.

Neben der Gesamt-Verzögerung werden die Verzögerungen der einzelnen Übertragungswege untersucht:

- Archiv → Cache Server (Access-Delay)
- Server-Datenverarbeitung (Server-Delay)
- Cache Server → Archiv (Propagation-Delay)

Diese Unterteilung verdeutlicht nachfolgend die entstehende Zeitproblematik innerhalb der RPKI hinsichtlich der Konsistenz. Die Analyse grenzt Messartefakte durch Verzögerungen gegenüber RPKI-Inkonsistenzen ab. Diese Messartefakte werden für nachfolgende Analysen ausgeschlossen, um die Messmethodik zu verfeinern.

Messmethodik und Umsetzung

Die Werte der Gesamtverzögerung werden für die Kollektoren während der Speicherung im Mikrosekunden-Bereich protokolliert. Die Gesamtverzögerung wird mithilfe der Differenz zweier Messzeitpunkte berechnet: der Zeitpunkt der Anfrage des Archivs und der Zeitpunkt des Erhalts des vollständigen ROA-Datensatzes.

Für die Berechnung der Verzögerungen der Übertragungs- und Bearbeitungswege werden die Log-Dateien der Cache Server verwendet. Diese enthalten Informationen über den Anfragezeitpunkt, die IP des Clients und die Dauer der Server-Datenverarbeitung. Diese Log-Dateien wurden in Kooperation mit den jeweiligen Betreibern der Cache Server für Analysezwecke übergeben. Ein Parsing der Log-Dateien extrahiert den Zeitstempel der empfangenen Anfrage und die Dauer der Server-Datenverarbeitung. Daraufhin wird die Differenz

zwischen dem Zeitpunkt des Versendens der Anfrage und dem Zeitpunkt des Empfangs der Anfrage berechnet (Access-Delay). Der Propagation-Delay ergibt sich aus Differenz zwischen dem Zeitstempel der empfangenen Anfrage und der Gesamtverzögerung. Für jeden ROA-Datensatz liegen letztendlich vier Werte vor: die Gesamtverzögerung, Access- und Propagation-Delay und die Dauer der Server-Datenverarbeitung.

Nachfolgend werden die einzelnen Messergebnisse vorgestellt, erläutert und hinsichtlich der entstehenden Zeitproblematik näher beleuchtet.

Messergebnisse

Die Messergebnisse der Delay-Analyse werden in drei verschiedene Abbildungen unterteilt. Die ersten beiden Abbildungen dienen der Analyse der Gesamtverzögerung. Diese ermöglichen zwei unterschiedliche Perspektiven auf die Eigenschaften der Gesamtverzögerung. Die dritte Abbildung dient der Analyse aller Verzögerungen der Übertragungs- und Bearbeitungswege.

Die Abbildung 4.3 beschreibt die Dauer der Gesamtverzögerung der Kollektor-Paare über den gesamten Messzeitraum. Ein Datenpunkt entspricht der Gesamtverzögerung eines ROA-Datensatz. Die Werte der Gesamtverzögerung werden in Sekunden mithilfe eines weiß nach schwarz verlaufenden Farbschemas dargestellt. Eine längere Verzögerungen wird durch einen dunkleren Datenpunkt repräsentiert. Ausfälle durch Nicht-Erreichbarkeit werden wie im Beispiel von CC04(Web) durch ein gestreiftes Feld markiert.

Die Messergebnisse zeigen, dass der Referenz-Kollektor CC01(Web) über den gesamten Messzeitraum die geringste Gesamtverzögerung aufweist. Daraufhin folgt CC07(Web), wobei beide Kollektoren kurzzeitige Anstiege verzeichnen. Die anderen Kollektoren weisen einen höheren Wertebereich der Gesamtverzögerung auf. CC02(Web) besitzt dabei einen gleichmäßig leicht erhöhten Verlauf mit einigen leichten Anstiegen. CC03(Web) hat neben einem ähnlich erhöhten Verlauf einige deutliche Ausschläge, die den Maximalwert der Abbildung von 30 Sekunden überschreiten. Die Gesamtverzögerung von CC04(Web) weist bis zum Zeitpunkt des Ausfalls einen leicht erhöhten Verlauf mit einigen deutlichen Ausschlägen oberhalb des Maximalwerts auf. Nachdem Ausfall ist eine deutliche Absenkung der Verzögerung sichtbar.

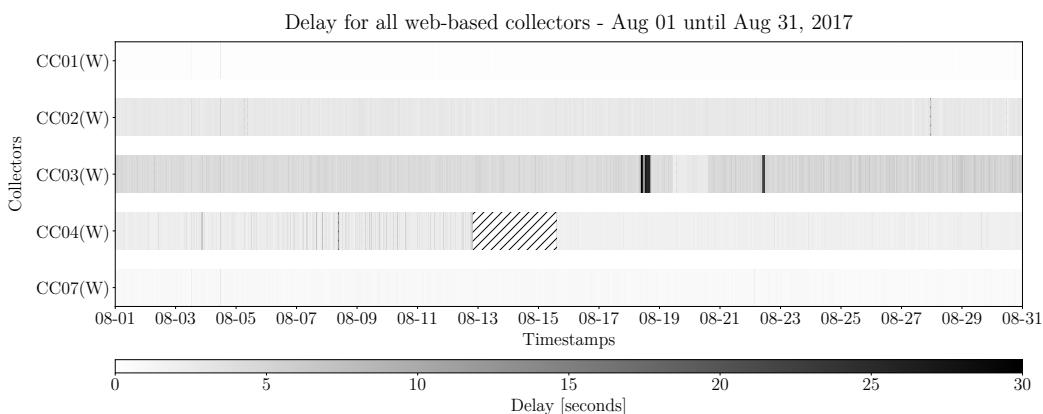


Abbildung 4.3: Messergebnisse der Delay-Analyse als Heat-Map in Sekunden [W-Web]

Eine weitere Visualisierung der Gesamtverzögerung wird in Abbildung 4.4 dargestellt. Der Fokus liegt hierbei mehr auf einzelnen Anstiegen und Ausprägungen als auf dem Verlauf.

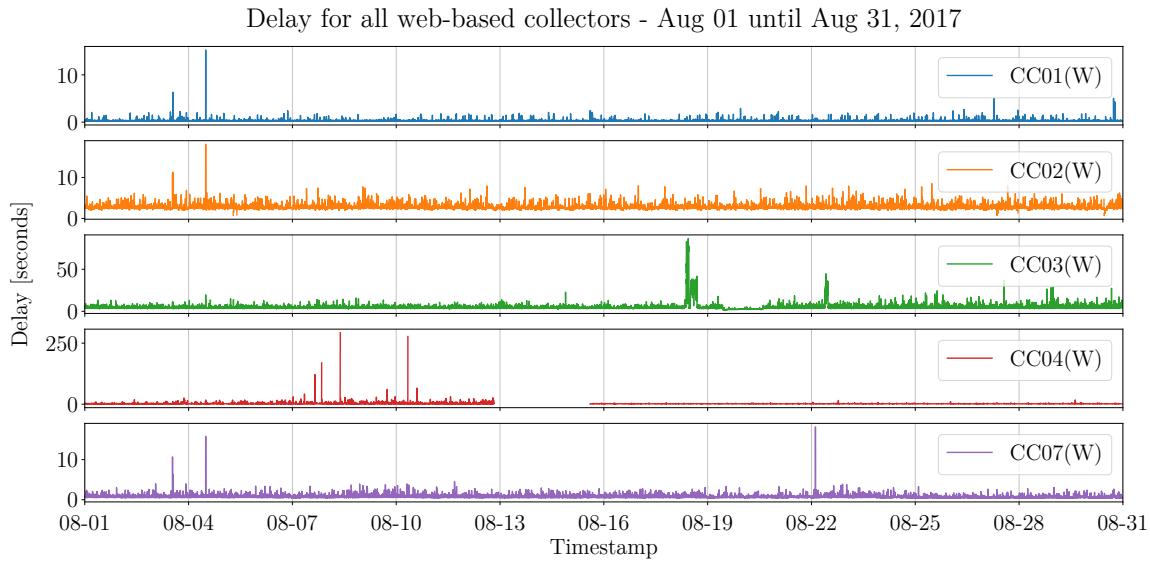


Abbildung 4.4: Messergebnisse der Delay-Analyse in Sekunden [W-Web]

Die Abbildung zeigt die Gesamtverzögerung in einer vertikalen Ausprägung, wobei die einzelnen Graphen teilweise unterschiedliche Achsenbereiche besitzen. So wird ein zeitgleicher Anstieg bei allen Kollektoren am 04.08 im Wert von rund 15 Sekunden sichtbar. Des Weiteren werden die tatsächlichen Maximalwerte von CC03(Web) von rund 75 Sekunden und CC04(Web) von rund 300 Sekunden deutlich.

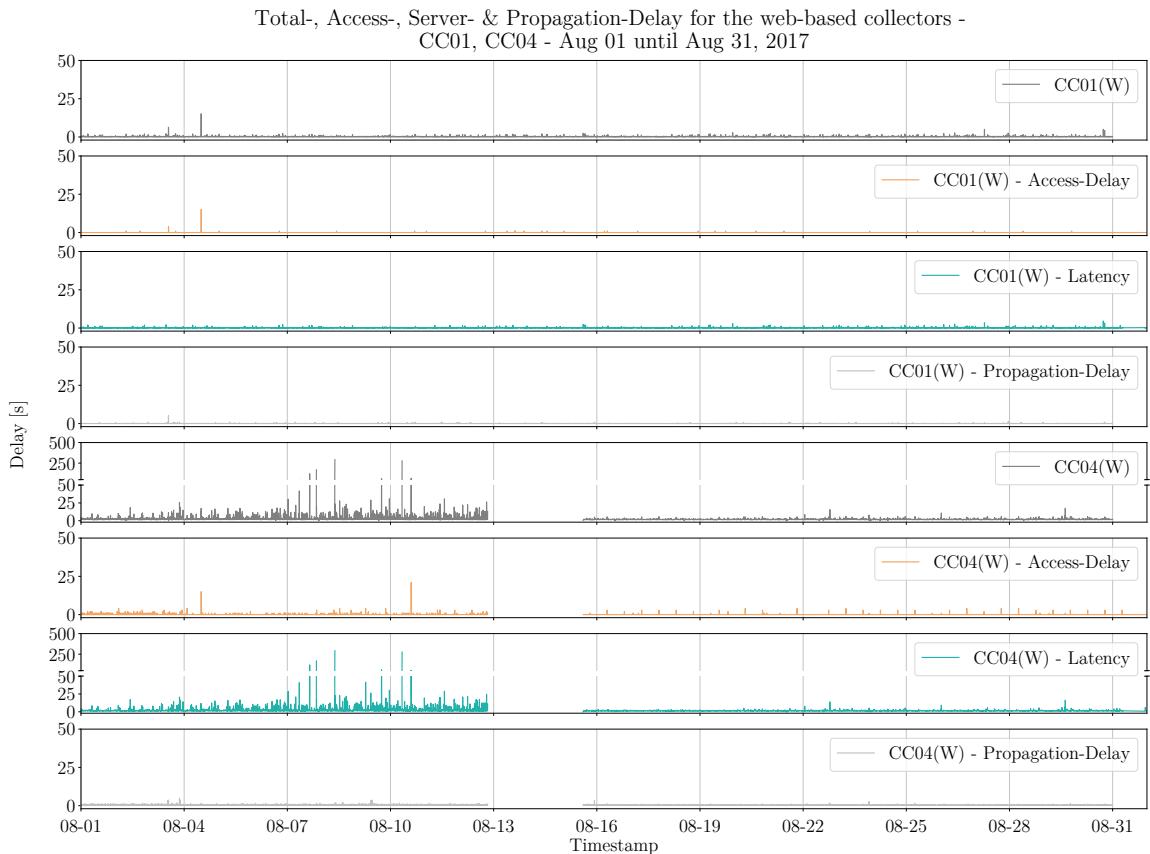


Abbildung 4.5: Messergebnisse der Delay-Analyse für alle Übertragungswege [W-Web]

Die Abbildung 4.5 stellt die Verzögerungen der einzelnen Übertragungs- bzw. Bearbeitungswege dar. Neben dem Referenzkollektor CC01(Web) wurde auch CC04(Web) aufgrund des extremen Anstiegs für diese Messung ausgewählt. Für jeden der beiden Kollektor wird jeweils die Gesamtverzögerung, der Access-, Server- und Propagation-Delay in einzelnen Graphen mit vertikaler Ausprägung in Sekunden dargestellt. Der Server-Delay (Latency) entspricht der benötigten Systemlaufzeit des jeweiligen Cache Servers zur Bearbeitung der Anfrage.

Die einzelnen Achsen stellen teilweise geteilte Wertebereiche dar, um so einige Maximalstellen hervorzuheben.

Der Referenzkollektor CC01(Web) besitzt abgesehen von wenigen Ausnahmen eine geringe Gesamtverzögerung, wodurch sich analog auch geringe Wertebereiche für die einzelnen Übertragungs- und Bearbeitungswege ergeben. Diese Ausnahmen haben Werte in einem Bereich von 0 - 15 Sekunden, wobei das Maximum bei der Gesamtverzögerung und dem Access-Delay am 04.08 auftritt. Der Server- und Propagation-Delay liegt während der Messung durchgängig in einem sehr niedrigen Wertebereich von wenigen Sekunden.

Der Kollektor CC04(Web) spiegelt einen anderen Verlauf der jeweiligen Verzögerungen wider. Es kommt zu mehreren zeitgleich hohen Anstiegen bei der Gesamtverzögerung und dem Server-Delay in einem Wertebereich von 60 - 300 Sekunden. Die Gesamtverzögerung hat einen nahezu deckungsgleichen Verlauf wie die Systemlaufzeit des Servers.

Der Access-Delay des Kollektors CC04(Web) weist nur wenige Anstiege im Bereich von rund 0 - 20 Sekunden auf. Diese treten teilweise zeitgleich mit den Anstiegen der Gesamtverzögerung und des Server-Delays auf. Ähnlich wie in den vorangegangenen Abbildungen ist die Absenkung der Verzögerungen nach dem mehrtägigen Ausfall deutlich erkennbar.

Nachfolgend wird die Auswertung der einzelnen Messergebnisse erläutert.

Auswertung

Die Messergebnissen der ersten beiden Abbildungen machen deutlich, dass es zu starken Schwankungen im Wertebereich der Kollektoren aber auch zwischen den Kollektoren kommt. Da die Gesamtverzögerung jedoch von zu vielen Faktoren abhängt, haben diese Werte nur wenig Aussagekraft für die Quantifizierung der Messfehler. Daher werden nachfolgend ausschließlich die Werte der dritten Abbildung für die Auswertung herangezogen.

Für die genaue Betrachtung der Messergebnisse der einzelnen Verzögerungen muss zunächst die vorkommende Zeitproblematik thematisiert werden. Dafür zeigt die Abbildung 4.6 drei unterschiedliche Verläufe der ROA-Datensatz-Beschaffung, wobei das Archiv in allen Fällen den aktuellen ROA-Datensatz des Zeitpunktes t anfragt, um diesen zu archivieren. Eine vollständige ROA-Datensatz-Beschaffung besteht dabei aus einer Datenanfrage des Archivs, das Berechnen des aktuellen ROA-Datensatzes des Cache Servers und der Antwort des Cache Servers mit dem ROA-Datensatz. Für die Veranschaulichung der Verläufe werden zwei Zeitstrahlen für das Archiv (links) und den Cache Server (rechts) verwendet. Der Zeitstrahl des Archivs zeigt die Gesamtverzögerung des Datenaustausches vom Zeitpunkt t bis $t+4$ für den Fall #1 und von t bis $t+11$ für die Fälle #2 und #3. Zwischen diesen Zeitstrahlen werden die Interaktionen mit zeitlichen Markierungen an den Zeitstrahlen gekennzeichnet. Die Interaktionen entsprechen den einzelnen Teilen der ROA-Datensatz-Beschaffung: Request

ROA-Dump, Calculate ROA-Dump, Send ROA-Dump.

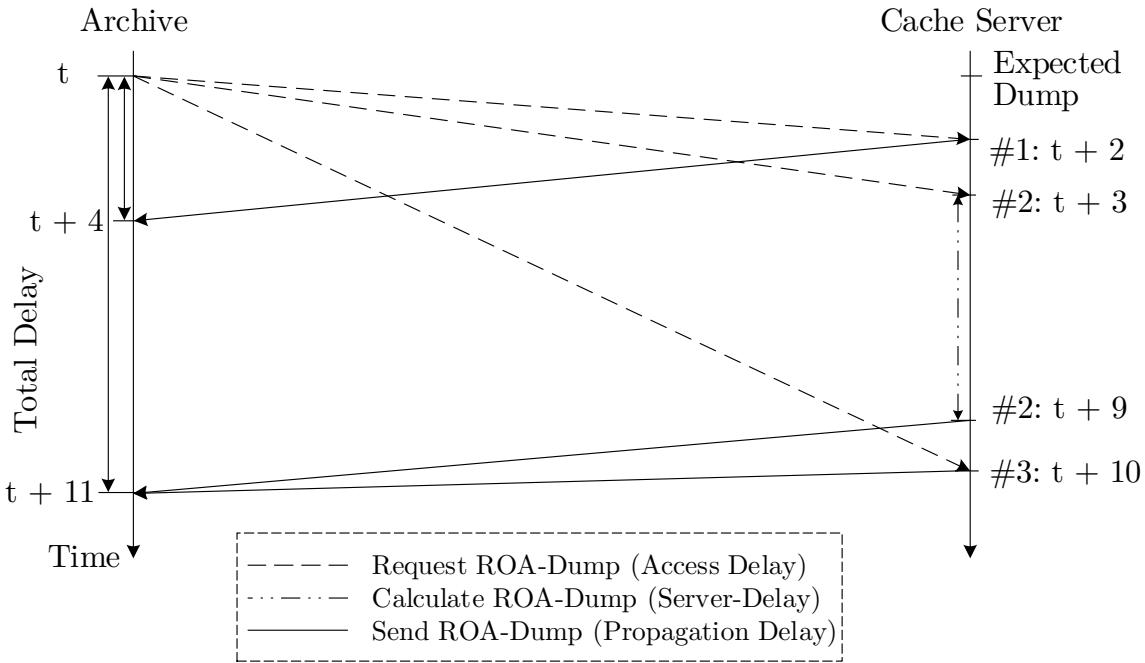


Abbildung 4.6: Unterschiedliche Möglichkeiten des zeitlichen Verlaufs einer Cache Server Anfrage

Der erste anzunehmende Fall #1 beschreibt eine ähnlich kurze Verzögerung der beiden Übertragungswege zwischen dem Archiv und Cache Server. In diesem Fall geht der Access-Delay von t bis $t+2$ und der Propagation-Delay von $t+2$ bis $t+4$. Es kommt somit zu keiner einseitigen Verzögerungsausprägung bei einem der beiden Übertragungswege. Die Datenanfrage erreicht den Cache Server zum Zeitpunkt $t+2$, der daraufhin ein Mitschnitt des aktuellen ROA-Bestandes erzeugt. Der Access-Delay spiegelt neben der Verzögerung des Übertragungsweges ebenfalls die Verzögerung zwischen dem gewünschten Zeitpunkt des ROA-Mitschnitts t (Desired Dump) und dem tatsächlichen Zeitpunkt des ROA-Mitschnitts $t+2$ wider. Der Server-Delay beträgt im Vergleich zu den Verzögerungen der Übertragungswege in einem sehr kleinen Wertebereich und kann in diesem Fall vernachlässigt werden. Ein kurzer Server-Delay kann auftreten, wenn der Cache Server während des Anfragezeitraums mit wenigen weiteren parallelen Anfragen konfrontiert wird. Der erstellte ROA-Datensatz wird daraufhin an das Archiv gesendet, wobei der Propagation-Delay die Verzögerung dieser Interaktion beschreibt. In diesem Fall liegt dieser in einem ähnlich geringen Wertebereich wie der Access-Delay, wodurch das Archiv den ROA-Mitschnitt zum Zeitpunkt $t+4$ erhält und mithilfe des Zeitstamps t archiviert. Je kürzer der Access-Delay desto geringer die Wahrscheinlichkeit, dass während der Übertragung der Anfrage der Cache Server ein Fetching durchführt und somit andere ROA-Daten als zum gewünschten Zeitpunkt t enthalten sind. Daher enthält das Archiv im Fall #1 mit hoher Wahrscheinlichkeit einen konsistenten ROA-Datensatz.

Der Fall #2 beschreibt neben einem leicht erhöhten Access-Delay (von t bis $t+3$) einen sehr hohen Server-Delay (von $t+3$ bis $t+9$). Dies kann auftreten, wenn es zum Zeitpunkt

der Anfrage eine hohe Anzahl an parallelen Anfragen gibt. Dies verursacht eine hohe Last des Cache Server und längere Bearbeitungszeiten. Für die Konsistenz des ROA-Datensatzes ist neben dem Access-Delay auch der Server-Delay maßgebend, wobei der exakte Zeitpunkt der Erzeugung des aktuellen ROA-Datensatzes während des Server-Delays nicht bekannt ist (Abschnitt 10). Im schlimmsten Fall kann der Mitschnitt des ROA-Datensatzes erst zum Zeitpunkt $t + 9$ erfolgen. Die enorme Bearbeitungsdauer des Cache Servers führt zu einer hohen Wahrscheinlichkeit eines Fetching während des Server-Delays. Der mitgeschnittene ROA-Datensatz enthält so mit hoher Wahrscheinlichkeit bereits andere ROA-Daten. Das Archiv erhält den ROA-Datensatz des Zeitpunkts $t + 9$ aufgrund des kurzen Propagation-Delay zum Zeitpunkt $t + 11$ und archiviert diesen fälschlicher Weise mithilfe des Zeitstempels t .

Der Fall #3 veranschaulicht eine ähnliche Problematik, wobei der Access-Delay Grund für die hohe Wahrscheinlichkeit anderer ROA-Daten als zum Zeitpunkt t ist. In diesem Fall empfängt der Cache Server die Anfrage für den ROA-Datensatz erst zum Zeitpunkt $t + 10$. Es folgt der Mitschnitt des aktuellen ROA-Datensatzes mit einem vergleichsweise geringen Server-Delay. Anschließend erhält das Archiv aufgrund des kleinen Propagation-Delay den ROA-Datensatz bereits zum Zeitpunkt $t + 11$ und archiviert diesen fälschlicher Weise mit dem Zeitstempel t .

Mithilfe der Verdeutlichung der Zeitproblematik können nun die gezeigten Messergebnissen der Abbildung 4.5 nachfolgend besser gedeutet werden.

Ein Beispiel für den Fall #1 sind die Zeitpunkte, die keine einseitige Ausprägung hinsichtlich der Verzögerungen aufweisen. Die entsprechenden ROA-Datensätze enthalten somit keine verfälschten Messdaten. Ein Beispiel für den Fall #3 befindet sich in den Messdaten der Gesamtverzögerung und des Access-Delays am 04.08 von CC01(Web). Dort besitzt der Access-Delay einen ähnlichen Wert wie die Gesamtverzögerung (in diesem Fall 15 Sekunden), wobei der Propagation- und Server-Delay einen vergleichsweise kleinen Wert aufweist. In diesem Fall ist die Wahrscheinlichkeit für ein zwischenzeitliches Fetching erhöht, wodurch die ROA-Daten verändert und mit einem inkorrekten Zeitstempel archiviert wurden. Dies kann Messfehlern verursacht haben. Die zeitgleich stattfindenden starken Schwankungen der Gesamtverzögerung und des Server-Delays von CC04(Web) zwischen dem 04.08 und 12.08 sind deutliche Beispiele für den Fall #2. In diesem Fall kommt es zu einer Wahrscheinlichkeit von rund 40-50% für Messfehler, da die Verzögerung der Server-Delay rund die Hälfte des konfigurierten Fetching-Intervalls einnimmt.

Die Messergebnisse der Abbildung 4.5 veranschaulichen die Messfehler in den Fällen #2 und #3 entstehen. Diese werden für folgenden Inkonsistenz-Analysen ausgeschlossen, um eine durchgängige Konsistenz der Daten zu gewährleisten.

Dieser Messfehler kann unter Umständen in der Realität zu tatsächlichen RPKI-Inkonsistenzen führen, wenn BGP-Router die ROA-Datensätze mithilfe des Web-Interfaces statt des standardisierten RTR-Protokolls anfragen. Da der Standard jedoch die Nutzung der RPKI vorsieht, wird dieser Fall hier und im Folgenden nicht weiter thematisiert.

Nachfolgend werden die einzelnen Inkonsistenz-Analyse vorgestellt und erläutert.

4.2 Inkonsistenz-Analysen

Die Inkonsistenzen-Analysen untersuchen die Unterschiede zwischen den einzelnen Cache Servern über den gesamten Messzeitraum hinsichtlich der

- Verzögerung innerhalb der RPKI-Infrastruktur (Beacon-Delay),
- Präsenz einzelner Trustanchor (Trustanchor),
- Differenz zwischen den ROA-Datensätzen (ROA-Unterschiede).

Des Weiteren werden die Messergebnisse der ROA-Unterschiedsanalyse durch genauere Be trachtungen hinsichtlich des zuständigen RIRs (ROA-Unterschiede per RIR), der betroffenen ASN (ROA-Unterschiede per ASN) und des konsekutiven Auftretens der einzelnen ASN-Betroffenheiten (ROA-Unterschiede per ASN - konsekutive Segmente) untersucht.

Nachfolgend wird die Verzögerung innerhalb der RPKI-Infrastruktur mithilfe der Beacon-Delay Analyse vorgestellt.

4.2.1 Beacon Delay

Beacons dienen der Verifikation der Infrastruktur gegenüber dem Standard. Im Fall von RPKI kann durch eine periodische Änderungen der Beacons-ROA die Umsetzung des RPKI-Standards innerhalb der Infrastruktur verifiziert und untersucht werden. So verschwinden die passenden Cache Server Einträge der Beacons bei einem Zertifikatsrückzug und erscheinen bei einer erneuten Zertifikatsausstellung. Neben der Prüfung der Existenz der Einträge kann ebenfalls die Verzögerung zwischen dem Zeitpunkt der Änderung im globalen Repository und dem Erscheinen bzw. Verschwinden der Einträge auf den Cache Servern ermittelt werden. Diese Verzögerung entspricht der Übertragungsdauer zwischen globalem Repository und Cache Server.

Die Beacon-Analyse untersucht diese Übertragungsdauer innerhalb der RPKI, um einen möglichen signifikanten Unterschied zwischen den Cache Servern zu identifizieren. Dieser Unterschied kann zu unterschiedlichen ROA-Datensätzen (Inkonsistenzen) auf den Cache Servern führen.

Im Folgenden wird für die Beacon-Analyse verwendete Messmethodik näher erläutert.

Messmethodik und Umsetzung

Für die Analyse der zeitlichen Verzögerung der ROA-Beacons werden die Zeitpunkte des Erscheinens und des Verschwindens der Beacons-Einträge für jeden Cache Server gemessen. Daraufhin wird die Differenz zwischen dem Soll-Wert des Erscheinens (4:00 Uhr UTC) und dem tatsächlichen Zeitpunkt des Erscheinens berechnet. Analog wird die Differenz für das Verschwinden der Beacon-Einträge mit einem Soll-Wert von (12:00 Uhr UTC) berechnet. Um zu prüfen, ob ein Beacon-Eintrag erscheint bzw. verschwindet, werden die ROA-Datensätze der Cache Server auf die folgenden einzigartigen Beacon-Einträge untersucht:

ASN: 51224, Max-Länge: 24, Präfixe:

- 147.28.241.0/24 • 147.28.244.0/24 • 147.28.245.0/24 • 147.28.247.0/24 • 147.28.249.0/24

Die ROA-Beacons gelten als sichtbar, falls alle Einträge in einem ROA-Datensatz nach dem Soll-Wert auftreten. In diesem Fall wird der Zeitstempel des ROA-Datensatzes als Erscheinen der ROA-Beacons definiert. Die ROA-Beacons werden als nicht sichtbar definiert, wenn keiner der Einträge enthalten ist.

Für diese Messmethodik muss das System der Zertifikatsausstellung und das Archiv einer Zeit-Synchronisierung unterliegen, um Zeitdifferenzen zu vermeiden. Die Umsetzung dieser Synchronisierung erfolgt durch eine konfigurierte NTP-gesteuerte Systemzeit auf beiden Systemen. Des Weiteren ist die Granularität der Zeitpunkte des Erscheinens bzw. Verschwindens durch das Messintervall beschränkt. So kann ein Cache Server die ROA-Beacons kurz nach dem Messintervall erhalten, wodurch das Erscheinen bzw. Verschwinden erst mit dem nachfolgenden Messintervall datiert wird. Für die Messwerte ergibt sich eine Toleranz von 180 Sekunden.

Für die Umsetzung der Messmethodik werden zunächst alle ROA-Datensätze hinsichtlich des jeweiligen Zeitraums gefiltert. Dabei werden nur ROA-Datensätze verwendet, die zwischen dem Soll-Wert und einer maximalen Verzögerung von 60 Minuten liegen. Für die Analyse werden nur Messungen mit einer vollständigen Menge an ROA-Datensätzen verwendet. Dadurch kann das Erscheinen bzw. Verschwinden der ROA-Beacons nicht in einem fehlenden ROA-Datensatz stattfinden. Anschließend wird die Menge an ROA-Datensätzen hinsichtlich der Beacon-Einträge untersucht und der Zeitstempel des ersten Erscheinens bzw. Verschwindens protokolliert. Es entstehen für jeden Tag des Messzeitraums zwei Messwerte für jeden Cache Server.

Nachfolgend werden die Messergebnisse für den gesamten Messzeitraum vorgestellt.

Messergebnisse

Die Abbildung 4.8 zeigt die zeitliche Verzögerung zwischen dem Zeitpunkt des Erhalts bzw.

Rückzugs der ROA-Beacons und der Sichtbarkeit dieser Änderung in den ROA-Datensätzen der Cache Server. Die Messwerte werden durch einen Boxplot dargestellt. Jede einzelne Box wird durch die gezeigten Parameter aus Abbildung 4.7 bestimmt. So wird für alle Messwerte eines Kollektors der Durchschnitt, der Median, das oberen und untere Quartil und die Ausreißer dargestellt.

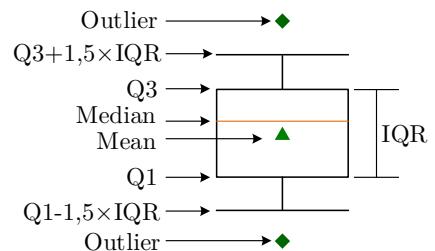


Abbildung 4.7: Legende für Boxplot

Die Cache Server sind nach den jeweiligen Paaren aufsteigend sortiert.

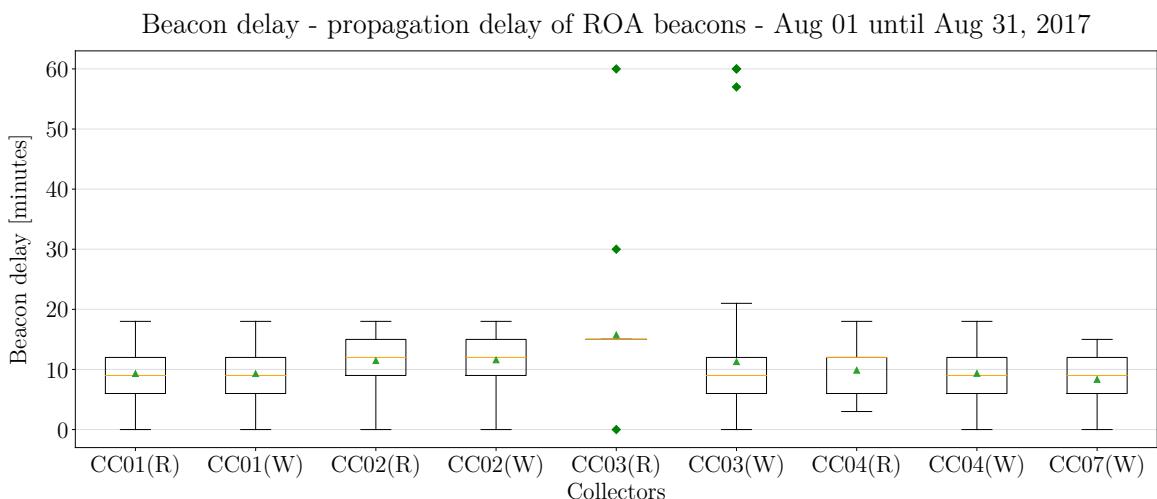


Abbildung 4.8: Messergebnisse der Beacon-Analyse in Minuten [R-RTR/ W-Web]

Die Messergebnissen zeigen, dass die Beacon-Verzögerungen zwischen den Kollektoren in einem Bereich von 0 - 22 Minuten liegen. Die Werte eines einzelnen Cache Servers schwanken ebenfalls in diesem Wertebereich. Die Schwankungen zwischen den Kollektoren kann eine Differenz von 22 Minuten annehmen, wenn ein Kollektor während einer Messung den minimalen und ein weiterer den maximalen Verzögerungswert erreicht. Die Kollektoren haben im Durchschnitt eine Verzögerung zwischen 7 - 15 Minuten und im Median einen Wert zwischen 8 - 13 Minuten. Es ergeben sich Schwankungen im Durchschnitt bzw. im Median von 7 bzw. 5 Minuten.

Während des Messzeitraums kam es ausschließlich beim Kollektor-Paar CC03 zu Ausreißern. So verzeichnete CC03(RTR) verschiedene Ausreißer mit den Werten 0, 30 und 60 Minuten und CC03(Web) mit 57 und 60 Minuten. Eine weitere Auffälligkeit ist die kontinuierliche Verzögerung von exakt 15 Minuten von CC03(RTR). Ursache dafür ist die Versionsunterschiede des RIPE Validators (2.17) mit älteren Algorithmen des Paares CC03 gegenüber den anderen Kollektoren (2.23). Dadurch ist das einzigartig konfigurierte Fetching-Intervall von 15 Minuten sehr statisch ausgeprägt. Die Ausreißer des Cache Servers beschränken sich daher auf die nächsten Fetching-Zeitpunkt nach 30 oder 60 Minuten. Diese werden im Folgenden nicht miteinbezogen, da diese nicht das allgemeine Verhalten widerspiegeln.

Nachfolgend werden die Messergebnisse ausgewertet und der Bezug zur RPKI-Inkonsistenz geschaffen.

Auswertung

Der allgemeine Wertebereich der Beacon-Verzögerung liegt bei 0 - 22 Minuten. Durch die Granularität des Messintervalls ergibt sich eine realistische Schwankung von 0 - 19 Minuten zwischen den einzelnen Cache Servern. Dieser Wertebereich kann auf das asynchrone Fetching der Cache Server mit unterschiedlichen Intervallen zurückgeführt werden. Jeder Cache Server synchronisiert im konfigurierten Fetching-Intervall (Standard: 11 Minuten) neue Objekte über das Übertragungsprotokoll RSYNC vom globalen Repository und validiert diese. Je nach Anzahl an Objekten und Performance des Cache Servers kann es Verzögerungen zu Schwankungen oberhalb des Fetching-Intervalls geben. Die Schwankungen unterhalb des Fetching-Intervalls ergeben sich durch den ungeraden Wert des Fetching-Intervalls. Daher starten und enden die Fetching-Prozesse an aufeinanderfolgenden Tagen stets leicht versetzt. Im extremen Fall kann der Fetching-Prozess exakt zum Erscheinen der ROA-Beacons beginnen und enden, wodurch sich eine geringe Beacon-Verzögerung ergibt. Bei einer Schwankungen zwischen 0 - 19 Minuten kann ein Cache Server aufgrund einer geringen Verzögerung bereits nach weniger als einer Minute das Fetching abgeschlossen haben. Ein anderer Cache Server kann zu diesem Zeitpunkt noch im Fetching-Prozess sein, wodurch der ROA-Datensatz des schnelleren Cache Servers erst 19 Minuten später verfügbar ist. Dadurch haben die Cache Server bereits zum Zeitpunkt des fertigen Fetching des schnelleren Cache Servers unterschiedliche ROA-Datensätze. Die Cache Server sind in diesem Fall nicht konsistent zueinander. Der Grad der Inkonsistenz ist von der Anzahl an veränderten Objekten abhängig. Im Fall eines erneuten Fetching des schnelleren Cache Server während der langsamere Cache Server noch kein Fetching beendet hat, wird die Inkonsistenz verstärkt. Der Grad der Inkonsistenz ist daher abhängig von der Anzahl der geänderten Daten eines Fetching und der Anzahl durchgeführten Fetching. Da der Durchschnitt der Verzögerung eine Schwankung von maximal 7 Minuten über die Cache Server besitzt, kein Cache Server dauerhaft mehr Iterationen durchführt als ein anderer.

Die Beacons und deren Messzeitpunkte stehen in dieser Analyse repräsentativ für die Gesamtheit der ROA des globalen Repository. Daher kann die Beeinträchtigung durch die Verzögerung auf alle ROA verallgemeinert werden. Die Verzögerung innerhalb der RPKI zwischen dem globalen Repository und den Cache Servern führt zu RPKI-Inkonsistenzen.

Nachfolgend wird die Präsenz der einzelnen Trust Anchor in den ROA-Datensätzen untersucht.

4.2.2 Trust Anchor

Die Trust-Anchor-Analyse untersucht die Präsenz der einzelnen Trust Anchor in den ROA-Datensätzen der Kollektoren für den gesamten Messzeitraum. Des Weiteren beschränkt sich die Analyse ausschließlich auf die Web-Versionen der Kollektoren, da das RTR-Protokoll keine Trust-Anchor-Informationen übermittelt. Es werden nur Trust Anchor betrachtet, die durch ein RIR organisiert werden. In der Messung werden somit die nachfolgenden Trust Anchor analysiert:

- ARIN • AfriNIC RPKI Root • LACNIC RPKI Root • RIPE NCC RPKI Root
- APNIC from AFRINIC RPKI Root • APNIC from ARIN RPKI Root
- APNIC from IANA RPKI Root • APNIC from LACNIC RPKI Root
- APNIC from RIPE RPKI Root

Die Analyse zeigt, ob bestimmte Trust Anchor für einige Kollektoren präsent oder nicht präsent sind. Nachfolgend wird die Messmethodik der Trust-Anchor-Analyse vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Für die Analyse der Trust-Anchor-Präsenz werden alle ROA-Datensätze eines Cache Servers hinsichtlich diesbezüglicher Einträge der Trust Anchor untersucht. Ein Trustanchor wird für einen Cache Server als präsent markiert, wenn im ROA-Datensatz mindestens ein Eintrag existiert, der von diesem Trust Anchor verwaltet wird.

Für die Umsetzung der Messmethodik werden die ROA-Datensätze zunächst nach dem jeweiligen Zeitraum gefiltert, nach Datum gruppiert und nach Zeitstempel sortiert. Anschließend kann die Präsenz jedes Trust Anchor für jeden einzelnen Zeitpunkt bestimmt und explizit protokolliert werden. Da aufgrund der Meta-Analysen infrage kommende Messfehler bereits ausgeschlossen wurden, können die Daten ohne weitere Aufbereitung direkt als Messergebnisse visualisiert werden.

Messergebnisse

Die Abbildung 4.9 zeigt die Präsenz der einzelnen Trust Anchor in den ROA-Datensätzen für alle Web-Versionen der Kollektor-Paare über den gesamten Messzeitraum.

Die Messwerte werden dabei durch einen Scatter-Plot dargestellt, wobei jede Präsenz eines Trust Anchor für einen Zeitpunkt eines ROA-Datensatzes durch einen einzelnen Messpunkt eingezeichnet wird. Das Fehlen eines Trustanchor ist dementsprechend ohne Datenpunkt als leere Fläche erkennbar.

Aufgrund der Granularität des Messintervalls gegenüber dem gesamten Messzeitraum kommt es bei einzelnen Datenpunkten zu Überschneidungen, wodurch je nach Präsenz eine durchgängige oder leicht unterbrochene Linie entsteht.

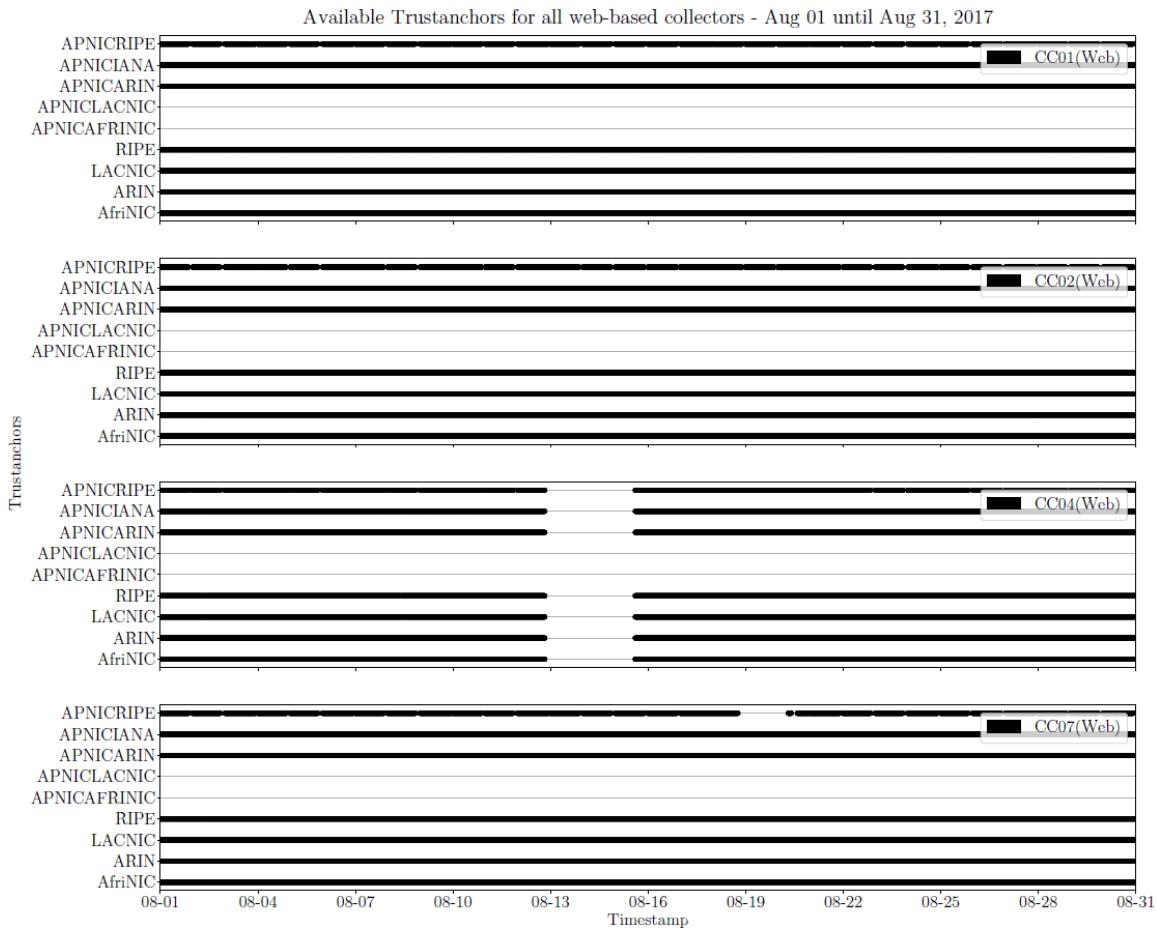


Abbildung 4.9: Messergebnisse der Trust-Anchor-Analyse

Die Messergebnisse zeigen, dass die Mehrheit der Trust Anchor nahezu durchgängig präsent ist, wobei aufgrund der Granularität nicht jedes Fehlen eines Datenpunktes sichtbar ist.

Auffällig ist die über den gesamten Messzeitraum fehlende Präsenz der beiden Trust Anchor "APNIC from LACNIC RPKI Root" und "APNIC from AFRINIC RPKI Root". Ursache hierfür ist, dass diese nur wenige Zertifikate und keine Objekte verwalten. Daher existiert in keinem ROA-Datensatz innerhalb des Messzeitraums ein dazugehöriger Eintrag der beiden Trust Anchor. Der Leerraum im Graphen des Kollektors CC04(Web) ist auf eine Nicht-Erreichbarkeit des Cache Server zurückzuführen und nicht auf eine fehlende Präsenz aller konfigurierten Trust Anchor.

Des Weiteren fällt eine mehrere Stunden dauernde fehlende Präsenz des Trust Anchor "APNIC RIPE" am 19.08 bis 20.08 bei CC07(Web) auf. Auffällig ist weiterhin die bei jedem Kollektor auftretende unterbrochene Präsenz des Trustanchor "APNIC RIPE" in nahezu regelmäßigen Abständen. Die jeweiligen Unterbrechungen sind dabei jedoch von Kollektor zu Kollektor unterschiedlich.

Nachfolgend wird die Auswertung hinsichtlich dieser Unregelmäßigkeiten in der Präsenz erläutert.

Auswertung

Da die Unregelmäßigkeiten ausschließlich den Trust Anchor "APNIC RIPE" betreffen, wurden die Messergebnisse hinsichtlich dieser Anomalien untersucht. So kommt es aufgrund der absoluten Menge von nur zwei ROA-Objekten von "APNIC RIPE" während des Messzeitraums zu bestimmten Momenten, in denen beide ROA-Objekte des Trust Anchor zurückgezogen werden und folglich zu diesen Zeitpunkten nicht mehr in den ROA-Datensätzen existieren. Dies äußerst sich nach der zuvor genannten Messmethodik in einer fehlenden Präsenz des Trust Anchor.

Die Messergebnisse zeigen, dass die Einträge des Trust Anchor zwischen den Kollektoren zu unterschiedlichen Zeiten verschwinden. Ursache dafür ist erneut die Verzögerung zwischen dem Cache Server und dem globalen Repository. Durch die unterschiedlichen ROA-Datensätze bzw. RPKI-Inkonsistenzen kann es vorkommen, dass alle Einträge eines Trust Anchor betroffen sind und somit fehlen.

Nachfolgend werden die Unterschiede zwischen den ROA-Datensätzen hinsichtlich möglicher Inkonsistenzen analysiert und ausgewertet.

4.2.3 ROA-Unterschiede

Die Analyse der ROA-Unterschiede untersucht die Differenz der ROA-Datensätze aller Kollektoren zu einem Zeitpunkt gegenüber dem ROA-Datensatz des Referenz-Kollektors. Für jeden Zeitpunkt ergeben sich die tatsächlichen Unterschiede zwischen den ROA-Datenbeständen. Hierbei handelt es sich um RPKI-Inkonsistenzen zwischen den Kollektoren.

Da alle bekannten Messfehler bereits ausgeschlossen wurden, können die bestehenden ROA-Datensätze unverändert für die ROA-Unterschiedsanalysen verwendet werden.

Messmethodik und Umsetzung

Für den Vergleich der ROA-Datensätze wird für alle Datensätze des Referenz-Kollektors geprüft, ob alle darin enthaltenen Einträge ebenfalls in den Datensätzen der anderen Kollektoren enthalten sind.

Im Falle eines Unterschieds einer der Faktoren, wird der Eintrag als unterschiedlich angesehen. Falls in beiden ROA-Datensätzen alle Einträge enthalten sind, werden diese ROA-Datensätze als gleichwertig angesehen. Die beiden Cache Server haben zu diesem Zeitpunkt eine identische Menge an Cache Server Einträgen gehabt.

Im Falle eines Unterschieds seitens eines Vergleichskollektors, wird die Menge an nicht-existierenden Cache Server Einträgen als Differenz gewertet. Analog verhält es sich mit einem Unterschied seitens des Referenzkollektors. Die Anzahl an Inkonsistenzen ist die Summe der beiden Differenzen. Im Falle eines fehlenden ROA-Datensatzes wird keine Differenz für die involvierten Kollektoren berechnet.

Für die Umsetzung der Messmethodik werden für jeden ROA-Datensatz des Referenz-Kollektors, alle ROA-Datensätze der Vergleichskollektoren des gleichen Zeitpunkts gruppiert. Jeder dieser ROA-Datensätze wird mit dem ROA-Datensatz des Referenz-Kollektors verglichen. Beim Vergleich zweier ROA-Datensätze von Web-Kollektoren alle Faktoren

(ASN, Präfix, Max-Länge, Trust Anchor) als Vergleichskriterium definiert. Im Falle zweier RTR-Kollektoren oder einem RTR- und einem Web-Kollektor wird das kleinste gemeinsame Vergleichskriterium (ASN, Präfix, Max-Länge) definiert. Anschließend werden für jeden Kollektor die beiden Differenzen zwischen dem ROA-Datensatzes und dem ROA-Datensatz des Referenzkollektors protokolliert. Es ergibt sich für einen Zeitpunkt der Messung für jeden Kollektor eine Anzahl an Differenzen. Diese Differenzen können direkt visualisiert und hinsichtlich RPKI-Inkonsistenzen seitens der Cache Server ausgewertet werden.

Messergebnisse

Die Abbildung 4.10 zeigt die Anzahl der Differenzen zwischen den einzelnen Kollektoren gegenüber dem Referenzkollektor für jeden ROA-Datensatz innerhalb des gesamten Messzeitraums. Die Messwerte werden dabei durch jeweils einzelne Graphen dargestellt.

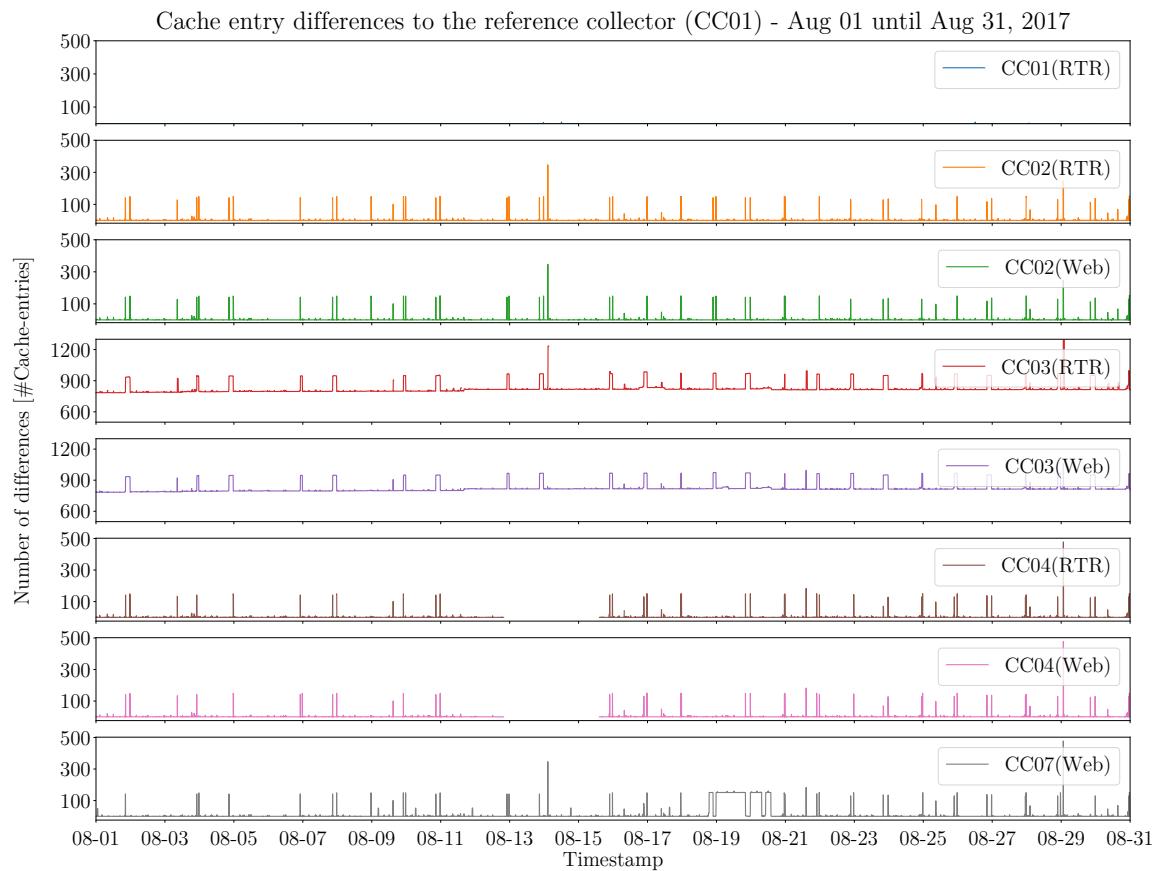


Abbildung 4.10: Messergebnisse der ROA-Unterschiedsanalyse für den Messzeitraum

Jeder Graph stellt den Vergleich zwischen dem jeweiligen Kollektor und dem Referenzkollektor CC01(Web) dar. Da sich diese Messung auf keine spezielle Version der Kollektor-Paare beschränkt, werden alle Kollektoren betrachtet. Die Achseninteilungen skalieren aufgrund der teilweise unterschiedlichen Wertebereiche im Falle von CC03(Web) und CC03(RTR) unterschiedlich.

Die Messergebnisse zeigen, dass CC01(RTR) kaum bis gar keine Differenzen aufweist, da es sich um die RTR-Version des Referenzkollektors handelt. Diese beiden Kollektoren teilen den gleichen ROA-Datenbestand. In seltenen Fällen kommt es zu kurzzeitigen einstelligen Unterschieden, die aufgrund der Implementierung des RTR-Servers stattfinden und somit nicht weiter untersucht werden.

Des Weiteren fällt auf, dass im Falle des Kollektor-Paars CC02 ein wiederkehrendes Schema innerhalb der Differenzen erkennbar ist. Für jeden Tag des Messzeitraums sind zwei Ausschläge zwischen den Uhrzeiten 20-24 Uhr in einem Wertebereich von 130-150 Unterschieden sichtbar. Dieses wiederkehrende Schema ist über den gesamten Messzeitraum gesehen mit sehr wenigen Ausnahmen beständig. Innerhalb der Uhrzeiten schwankt es jedoch bei einem stets gleichen Wertebereich leicht. Dadurch kann es aufgrund der Granularität der Abbildung zu Überschneidungen kommen, so dass diese als einzelner Ausschlag wahrgenommen werden. Das Schema wird von zwei Ausschlägen an den Tagen 14.08 und 29.08 in einem erhöhten Wertebereich von 240 bis 480 Differenzen unterbrochen. Des Weiteren gibt es einige Unregelmäßigkeiten zwischen den einzelnen Iterationen des Schemas die unterschiedliche Wertebereiche besitzen. Beispiele dafür sind die Ausschläge an den Tagen 11.08, 21.08 und 30.8 die bei allen Kollektoren auftreten. Aufgrund der allgemeinen Betrachtung in dieser Analyse können diese jedoch nicht ausgewertet werden.

Das Kollektor-Paar CC03 hat einen kontinuierlichen Unterschied von rund 800 - 900 Differenzen. Ursache dafür ist die Nicht-Einbindung des Trustanchor "ARIN" und der bereits erwähnte Versionsunterschied des RIPE Validators. Aufgrund dieser unterschiedlichen Vergleichsbasis wird das Kollektor-Paar in den nachfolgenden Analysen nicht weiter analysieren, um die Aussage der Messdaten nicht zu verfälschen. Dennoch ist auch hier das wiederkehrende Schema innerhalb der Differenzen erkennbar.

Erneut gibt es aufgrund einer dauerhaften Nicht-Erreichbarkeit des Kollektor-Paars CC04 keine Daten zwischen dem 13.08 und 15.08. Die weiteren Messdaten ähneln erneut dem Schema mit zwei Ausschlägen pro Messtag. Daher ist nur eine der beiden Unterbrechungen am 29.08 in einem Wertebereich von 480 Differenzen sichtbar.

Das wiederkehrende Schema ist innerhalb eines Kollektoren-Paars identisch, wobei es zwischen den Kollektoren-Paaren zu Unterschiede kommt. Dies wird am Beispiel des Kollektors CC07(Web) deutlich. Dieser weist erneut einen ähnlichen Verlauf auf, wobei zwischen dem 19.08 und 21.08 einige der Ausschläge durchgängig sind. Ursache dafür können die Ausfälle einzelner Trustanchor des RIRs "APNIC" in diesem Zeitraum sein.

Eine weitere Auffälligkeit sind die leichten Ausschläge im Wertebereich von rund 10 Differenzen. Diese treten bei den Kollektoren ebenfalls paarweise an jedem Tag der Messung auf. Des Weiteren existieren bei den einzelnen Kollektoren unterschiedliche Differenzen die keinem Schema folgen und vergleichsweise geringe Werte aufweisen.

Die Ursachen für alle zuvorgenannten Anomalien werden nachfolgend in der Auswertung genauer erläutert.

Auswertung

Die Betrachtungen der Messergebnisse zeigen deutlich ein wiederkehrendes Schema, das bis auf das Referenz-Kollektor-Paar alle untersuchten Kollektoren betrifft. Genauere Nach-

forschung dieses Schemas ergaben, dass eine einheitliche Ursache für diese Inkonsistenz existiert. So sind alle Einträge der betroffenen ROA-Datensätze dem Trustanchor “APNIC IANA“ zugeteilt.

Des Weiteren werden die Einträge nicht im RIR-organisierten Repository sondern in einem delegierten Repository (rpki-repository.nic.ad.jp) gehalten. Dieses Repository wird neben allen anderen externen Repository in das Repository des RIRs eingespeist. Mithilfe dieser Informationen wird deutlich, dass alle Einträge, die auf ROAs dieses delegierten Repository basieren, in regelmäßigen Abständen zu Differenzen zwischen den Kollektoren und dem Referenz-Kollektor führen. Diese Differenzen verursachen erhebliche RPKI-Inkonsistenzen. Ursache dafür ist ein Synchronisierungsfehler zwischen dem delegierten Repository und dem Repository des RIRs (APNIC). Dadurch kommt es regelmäßig zwischen den Uhrzeiten 20 und 24 Uhr zu einem fälschlicherweise stattfindenden Zurückziehen (20 Uhr) und erneutem Ausstellen (24 Uhr) der dort gespeicherten ROAs. Da diese Aktion im globalen Repository stattfindet und alle Cache Server damit verbunden sind, findet diese Änderungen auf allen Cache Server statt. Aufgrund der Verzögerung zwischen dem Cache Server und dem globalen Repository kommt es zu Unterschieden zwischen den einzelnen Cache Servern hinsichtlich dieser Änderungen. Es entsteht so je nach Verzögerung des Kollektors eine kurzzeitige Differenz in Höhe der Anzahl der Einträge der dort gespeicherten ROAs (rund 130-150 Objekten). Der Auslöser für diese erhebliche Inkonsistenz ist das “nicht-legitime“ Synchronisierungsproblem des globalen Repository. Die Ursache ist hingegen die Verzögerung innerhalb der Infrastruktur zwischen den Cache Servern und dem globalen Repository.

Die weitere Auffälligkeit der Unterbrechungen dieses Schemas in Form von zwei Ausschlägen an den Tagen 14.08 und 29.08 hat eine ähnliche Kausalität, wobei es sich um einen legitimen Auslöser handelt. Der Auslöser für die einmaligen Ausschläge ist die (Neu-)Signierung aller ROAs eines Zertifikats (XS3RVLXc4h-3hsUm297xsEWSirg.cer). Da diese Aktion erneut im globalen Repository stattfindet und alle Cache Server damit verbunden sind, findet diese Änderungen auf allen Cache Server statt. Aufgrund der Verzögerung zwischen den Cache Servern und dem globalen Repository erscheinen die Änderungen bei den Cache Server zu unterschiedlichen Zeitpunkten. Es entsteht so je nach Verzögerung des Cache Servers eine kurzzeitige Differenz in Höhe der Anzahl der Einträge der ROAs (240-480), dessen Signierung zu diesem Zeitpunkt stattfand. Da ein Zertifikat nur den letzten Signierungszeitpunkt (29.08) speichert, kann nicht mit hinreichender Genauigkeit der Auslöser für den 14.08 bestimmt werden (siehe Abschnitt 10). Diese Inkonsistenz tritt an mehreren Stellen während des Messzeitraums auf. Im Fall von CC04(Web) äußert sich diese Anomalie in Form von 0-10 Differenzen am 11.08. Dabei sind nicht immer alle ROA eines Zertifikats betroffen sondern auch einzelne ROA.

Aus diesem Sachverhalt wird deutlich, dass eine Änderung der ROAs durch eine legitime (gültige) Aktion wie die Signierung von ROAs oder einer nicht-legitimen (ungültigen) Aktion, wie das Synchronisierungsproblem zwischen dem delegierten und globalen Repository, Auslöser für eine Inkonsistenz sein kann. Diese Inkonsistenz tritt jedoch nur in Kombination mit der Verzögerung zwischen dem Cache Server und dem globalen Repository auf. Je nach Aktion findet diese Inkonsistenz einmalig oder wiederkehrend statt.

Da das Signieren neuer ROAs der Änderung der ROA-Beacons ähnelt, löst diese legitime Aktion ebenfalls “Inkonsistenzen“ in Höhe der Anzahl der Beacons aus.

Mithilfe dieser allgemeinen ROA-Unterschiedsanalyse können keine weiteren Aussagen über die restlichen geringen Ausschläge gemacht werden. Diese werden neben den Auswirkungen der bereits bekannten Inkonsistenzen in den nachfolgenden Analysen weiter untersucht.

4.2.4 ROA-Unterschiede pro RIR

Die Analyse der ROA-Unterschiede pro RIR basiert auf der vorangegangenen ROA-Unterschiedsanalyse. Die Analyse untersucht das maximale Differenz-Verhältnis eines RIR pro Tag. Das Differenz-Verhältnis ist dabei der Quotient aus der Anzahl an zum RIR gehörenden Differenzen und der Gesamtanzahl der zum RIR gehörenden Einträge zu diesem Zeitpunkt.

Mithilfe der Analyse können die Differenzen hinsichtlich der verantwortlichen RIR genauer untersucht werden. Es kann festgestellt werden, ob die Differenzen einen signifikanten Anteil an der Gesamtmenge der Einträge dieses RIR besitzen.

Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Die Messmethodik der Analyse beschränkt sich ausschließlich auf Kollektoren, die über ein Web-Interface angesprochen werden, da das RTR-Protokoll keine Informationen zum entsprechenden RIR überträgt. Das Kollektor-Paar CC03 wird hier und im Folgenden aufgrund der ungleichen Ausgangslage der Cache Server nicht weiter in die Analysen integriert.

Für die Analyse der Differenzen hinsichtlich der verantwortlichen RIR werden die Analyse-Dateien, die alle Differenzen für alle Zeitpunkte enthalten, verwendet. Diese Dateien werden zunächst für das entsprechende Intervall gefiltert, nach Messtag gruppiert und nach Zeitstempel sortiert. Für jeden Zeitpunkt wird das Verhältnis zwischen der Anzahl an Differenzen des RIR und der Summe der gesamten Anzahl an Einträgen dieses RIR in den Datensätzen der beiden verglichenen Kollektoren berechnet:

$$\text{Differenz-Verhältnis} = \frac{|(D_1/(D_1 \cap D_2)) \cup (D_2/(D_1 \cap D_2))|}{|D_1| + |D_2|}$$

D_1 und D_2 spiegeln dabei die Menge an Einträgen des RIR in den ROA-Datensätzen der beiden Kollektoren wider. Nachdem dieses Verhältnis für alle Differenzen eines Tages berechnet wurde, wird für jedes einzelne RIR der Maximalwert ermittelt. Der Maximalwert entspricht so dem relativen Differenz-Verhältnis und nicht dem absoluten Wert der Differenz eines RIR. Grund dafür ist die unterschiedliche Anzahl an Einträgen der RIR. Es ergibt für jedes RIR ein Maximalwert für jeden Tag des Messzeitraums, der ohne weitere Aufbereitung als Messergebnis visualisiert werden kann.

Messergebnisse

Die Abbildung 4.11 zeigt für jedes RIR den Maximalwert des Differenz-Verhältnisses für jeden Tag des Messzeitraums. Die Messwerte werden dabei durch einen Scatter-Plot dargestellt. Jeder Datenpunkt repräsentiert den Maximalwert der Differenz des RIR für diesen Tag.

Aufgrund der Granularität kommt es bei den Datenpunkten zu Überschneidungen, wodurch nicht alle Datenpunkte vollständig disjunkt visualisiert werden.

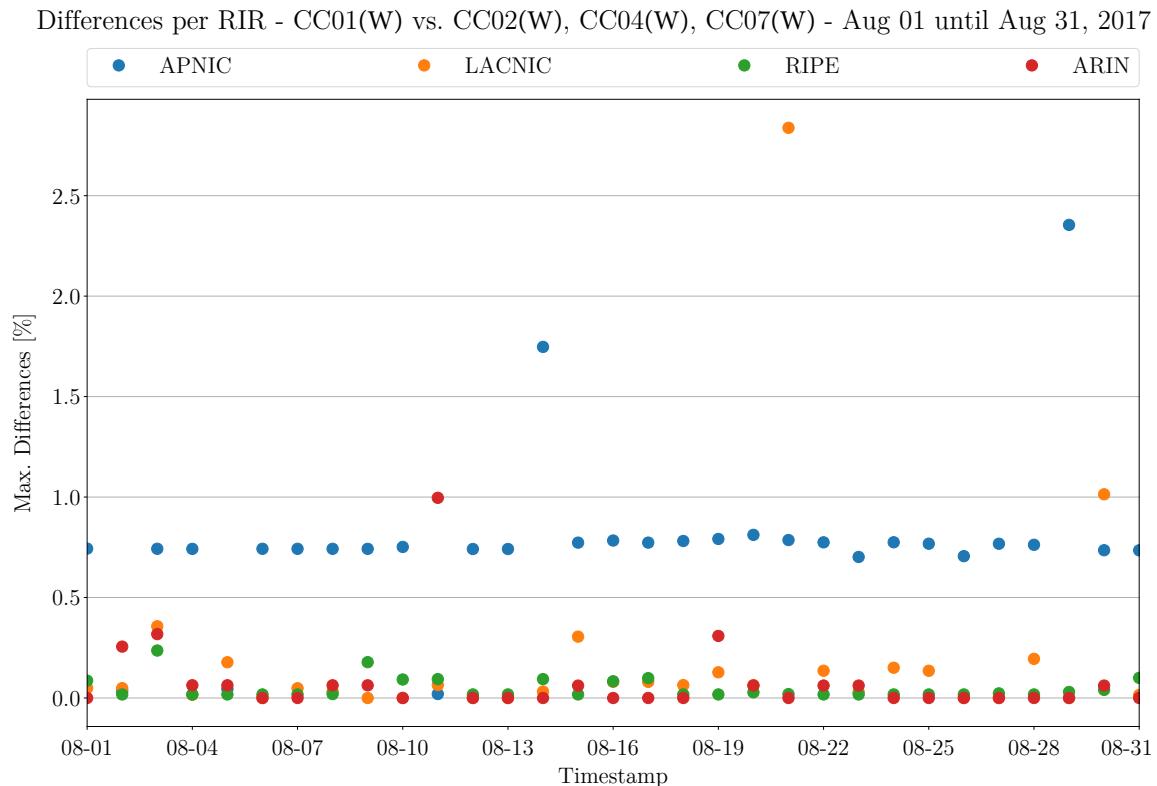


Abbildung 4.11: Messergebnisse der ROA-Unterschiedsanalyse pro RIR [W-Web]

Die Messergebnisse zeigen, dass das RIR “AfriNIC“ keine Datenpunkte in den Messergebnissen besitzt. Ursache dafür ist, dass keine Differenzen während des gesamten Messzeitraums gemessen wurden. Die Datenpunkte der restlichen RIR spiegeln unterschiedliche Verläufe wieder, wobei alle in einem Wertebereich von 0 - 3% liegen.

Der Verlauf von “RIPE“ besitzt im Gegensatz zu den anderen RIR keine deutlichen Ausschläge. Die Datenpunkte schwanken nur leicht in einem Bereich von 0 - 0.25%. Die Datenpunkte von “ARIN“ zeigen einen ähnlichen Verlauf, wobei diese in einem Wertebereich von 0 - 1% liegen. Es kommt zu etwas höheren einzelnen Schwankungen, wobei jedoch rund zwei Drittel der Datenpunkte einen Wert von rund 0%. Am 11.08 kommt es zu einem deutlichen Anstieg auf 1%. Der Verlauf des RIR “LACNIC“ besitzt einen anderen Verlauf mit deutlich höheren Schwankungen. So liegt die Mehrheit der Datenpunkte im Wertebereich von 0 - 0,5%, wobei zwei deutliche Anstiege mit 2,8% am 21.08 und 1% am 30.08 existieren. Die Datenpunkte von “APNIC“ zeigen im Gegensatz zu den anderen RIR einen gänzlich anderen Verlauf. So liegen alle Datenpunkte mit Ausnahme von zwei Anstiegen in einem Wertebereich knapp 0,6 - 0,8%. Die beiden Anstiege bilden am 14.08 mit rund 1,7% und am 29.08 mit rund 2,4% das Maximum des Wertebereichs aller RIR. Eine weitere Ausnahme bilden drei Datenpunkte im Bereich von knapp über 0% an den Tagen 02.08, 05.08 und 11.08.

Auswertung

Aus den Betrachtungen der Messergebnisse geht zunächst der gänzlich andere Verlauf des RIR "APNIC" hervor. Auslöser dafür ist das nicht-legitime Ereignis des Synchronisierungsproblems zwischen dem delegierten und dem Repository des RIR. Aufgrund der wiederkehrenden Inkonsistenzen in einem Bereich von 130-150 und der relativ konstanten Gesamtmenge von rund 10500 Einträgen beeinflussen die Inkonsistenzen kontinuierlich mehr als 0,7% der Einträge des RIR. Auslöser für die beiden Anstiege mit 1,7 und 2,4% ist das legitime Ereignis der Signierung neuer ROA in Kombination mit der Verzögerung. Diese Signierung der ROA führt dazu, dass an diesen Tagen mehr als 2,4% der Einträge des RIR Teil einer Inkonsistenz sind. Dies macht erneut deutlich, welche Auswirkungen diese Inkonsistenzen in diesem Fall aus der Perspektive der RIR haben.

Eine weitere Auffälligkeit sind die Ausschläge des RIR "LACNIC" an den Tagen 21.08 und 30.08. Da diese Ausschläge bereits in der ROA-Unterschiedsanalyse aufgefallen sind, steht das RIR "LACNIC" in Verbindung mit der Mehrheit dieser Inkonsistenzen. In diesem Fall sind mehr als 2,7% der Einträge von LACNIC an Inkonsistenzen beteiligt. Dabei werden die Differenzen jedoch nicht von den bereits bekannten Ereignissen ausgelöst. Deren unbekannte Ursache wird hinsichtlich der beteiligten ASNs in den nachfolgenden Analysen genauer untersucht.

Ein ähnlicher Fall liegt mit dem Ausschlag am 11.08 des RIRs "ARIN" vor. Das RIR steht in Verbindung mit der Mehrheit der Inkonsistenzen an diesem Tag und beeinflusst rund 1% der Einträge des RIR. Auslöser hierfür ist jedoch das bereits bekannte legitime Ereignis des Signierens. Dabei wurden jedoch nicht die ROA signiert sondern das Zertifikat, dem die ROA angehören. Diese Änderung im globalen Repository löst erneut in Kombination mit der Verzögerung eine RPKI-Inkonsistenz aus, die rund 1% des RIR betrifft.

Die Messergebnisse zeigen, dass RPKI-Inkonsistenzen durch nicht-identifizierte Anomalien, Synchronisierungsprobleme und Signierung in Kombination mit der Verzögerungen eine durchaus erhebliche Auswirkung auf die Konsistenz der RIR haben.

4.2.5 ROA-Unterschiede pro ASN

Die Analyse der ROA-Unterschiede pro ASN basiert ebenfalls auf der vorangegangenen ROA-Unterschiedsanalyse. Die Analyse untersucht das maximale Differenz-Verhältnis einer ASN pro Tag. Das Differenz-Verhältnis ist dabei der Quotient aus der Anzahl Differenzen der ASN und der Gesamtanzahl Einträge der ASN zu diesem Zeitpunkt.

Mithilfe der Analyse können die Differenzen hinsichtlich der verantwortlichen ASN genauer untersucht werden. Es kann festgestellt werden, ob die Differenzen signifikante Anteile an der Gesamtmenge der Einträge der verantwortlichen AS besitzen.

Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Im Gegensatz zur ROA-Unterschiedsanalyse pro RIR werden bei dieser Analyse beide Version der Kollektor-Paare verwendet, da beide Versionen die ASN-Information übertragen.

Erneut wird das Kollektor-Paar CC03 ausgeschlossen, um die Messwerte nicht zu verfälschen.

Für die Analyse der Differenzen hinsichtlich der ASN werden die Analyse-Dateien erneut verwendet. Diese Dateien werden zunächst für das entsprechende Intervall gefiltert, nach Messtag gruppiert und nach Zeitstempel sortiert. Für jeden Zeitpunkt wird das Verhältnis zwischen der Anzahl an Differenzen der ASN und der Summe der gesamten Anzahl an Einträgen dieser ASN in den Datensätzen der beiden verglichenen Kollektoren berechnet. Die verwendete Berechnung entspricht somit der vorangegangenen Formel. Die beiden Faktoren entsprechen in diesem Fall der Menge an Einträgen des ASN in den beiden ROA-Datensätzen. Nachdem das Verhältnis für alle Differenz dieses Tages berechnet wurde, kann für jedes einzelne ASN der Maximalwert ermittelt werden. Somit ergibt sich für jedes ASN ein Maximalwert für jeden Tag des Messzeitraums, der daraufhin als Messergebnis visualisiert werden kann.

Messergebnisse

Die Abbildung 4.12 zeigt für jeden Tag innerhalb des Messzeitraums die Verteilung aller Maximalwert für alle enthaltenen ASN. Die Messwerte werden dabei durch einen Boxplot-Plot dargestellt. Die Verteilung eines Tages wird durch eine einzelne Box dargestellt (Legende in Abbildung 4.7). Aufgrund teilweise sehr kleiner Wertebereiche sind nicht alle Boxen vollständig dargestellt.

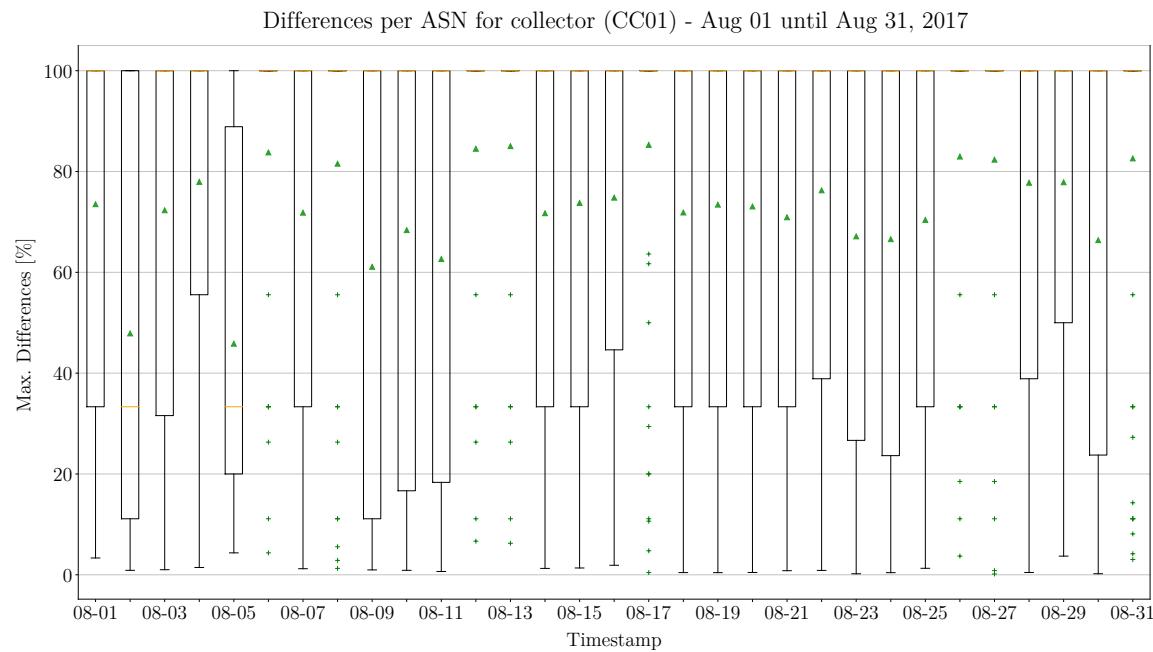


Abbildung 4.12: Messergebnisse der ROA-Unterschiedsanalyse pro ASN

Die Messergebnisse zeigen, dass alle Maximalwerte in einem stark durchwachsenen Wertebereich von 0 - 100% liegen. Aufgrund der Definition der maximalen Differenz kann es keine Werte oberhalb der 100% geben. Die Maximalwerte der einzelnen Tage des Messzeitraums

zeigen unterschiedliche Verteilungen über alle präsenten ASNs. Die Verteilungen können in drei unterschiedliche Typen I, II, III unterteilt werden.

Die Mehrheit der Verteilungen sind vom Typs I, dessen oberes Quartil bei 100% und unteres Quartil bei einem variablen Wert im Bereich von 10 - 60% liegt. Beispiele für diesen Typ sind die Verteilungen von 01.08 bis 04.08. Diese Verteilungen erscheinen recht einseitig hinsichtlich der Orientierung zur 100%. Der Durchschnitt dieser Verteilungen liegt in einem Wertebereich von 40 - 80%, wobei die Mehrheit der Durchschnitte dieser Verteilungen bei rund 70% liegt. Aufgrund der stark gestreuten Verteilung existieren keine Ausreißer am unteren Quartil. Der Median der Verteilungen liegt hierbei an allen Messtagen mit Ausnahme des Messtages 02.08 (35%) stets bei 100%.

Die Verteilungen des Typ II definieren sich durch keine Einseitigkeit der Messwerte. Ein Beispiel für diesen Typ ist die Verteilung der Maximalwerte am 05.08. Das obere Quartil liegt dabei in einem Wertebereich von rund 90% und das untere Quartil bei rund 20%. Der obere Whisker reicht bis zu einer Grenze von 100% und der untere Whisker bis knapp 5%. Sowohl der Durchschnitt als auch der Median liegen jeweils unter 50% bei rund 45% und 35%. Erneut existieren aufgrund der Eigenschaften dieser Verteilung keine Ausreißer an den Enden der Whisker.

Die Verteilungen des Typs III fallen aufgrund ihrer extremen Einseitigkeit auf. So kommt diese Art der Verteilung an 8 unterschiedlichen, teilweise aufeinanderfolgenden Messtagen vor, 06.08, 12.08 - 13.08 und 26.08 - 27.08. Aufgrund der Einseitigkeit kommt es zu Überlappungen der einzelnen Elemente der Boxen. Alle Elemente der Box liegen – mit Ausnahme des Durchschnitts und der Ausreißer – bei einem Wert von 100%. In diesem Fall liegt der Durchschnitt in einem Wertebereich von 90 - 95% und die Ausreißer in einem Bereich von 0 - 65%. Die Ausreißer sind dabei mehrheitlich recht gleichmäßig verteilt.

Auswertung

Die Messwerte verdeutlichen, dass die Verteilungen der einzelnen Tage unterschiedlicher Typen angehören. Die Eigenschaften der Verteilung (05.08) vom Typ II zeigen, dass die Mehrheit der Maximalwerte in einem Wertebereich von 20 - 90% und durchschnittlich bei rund 45% liegen. In diesem Fall beeinflussen somit die Inkonsistenzen im Durchschnitt rund die Hälfte der Einträge der ASN. Bei einigen ASN kommt es zu einem größeren Wert (Whisker) von 100% oder zu einem geringen Wert von rund 5%. Aus der ROA-Unterschiedsanalyse ist dieser Tag dadurch aufgefallen, dass das Synchronisierungsproblem dort sehr gering ausgefallen ist. Dieser Fakt beeinflusst die Verteilung der Maximalwerte dahingehend, dass es im Gegensatz zu den anderen Tagen weniger Maximalwerte mit 100% gibt. Ursache dafür ist, dass die ROA eines AS meist auf einem Repository gelagert werden. Im Falle des Synchronisierungsproblems fallen entsprechend alle ROAs der betroffenen ASN aus, wodurch 100% der Einträge der ASN betroffen sind. Das heißt, dass an Tagen mit Synchronisierungsproblem eine Vielzahl an ASN aus diesem Grund einen Maximalwert von 100% erreichen. Am 05.08 hingegen ist nicht der Fall. Im Falle des Auslösers durch das Signieren von ROA sind die Maximalwerte von der Anzahl der betroffenen ROA abhängig. So können alle ROAs einer ASN neu signiert werden (100%) oder nur ein Bruchteil, da die ROA in unterschiedlichen Zertifikaten verteilt sind. Daher kann keine genaue Aussage über die Auswirkungen dieses Auslösers getroffen werden kann.

Die Eigenschaften der Verteilungen des Typs I spiegeln die Auswirkung des Synchronisationsproblems wider. So kommt es zu einer Vielzahl an Maximalwerte mit einem Wert von 100%, die eine Einseitigkeit der Werte erzeugen. Inkonsistenzen durch das Signieren der ROA oder unbekannter Auslöser können zu Maximalwerten unterhalb der 100% führen, wodurch eine Verteilung des Typs I entsteht. Die Inkonsistenzen des RIR "LACNIC" aus der vorherigen Analyse befinden sich in einer Verteilung des Typs I. Durch Nachforschungen wurde deutlich, dass diese ausschließlich von der ASN "28006" verursacht wurden. In diesem Fall wurden rund 77% der Einträge dieser ASN von dieser Inkonsistenz beeinflusst, wobei jedoch der Auslöser für diese Inkonsistenz weiterhin unbekannt ist.

Im Gegensatz zu den Verteilungen des Typs I kommt es beim Typ III zu einer extremen Auswirkung des Synchronisationsproblems. An diesen Tagen gab es neben diesen Inkonsistenzen kaum (Ausreißer) bis keine anderen Differenzen. Bei den Ausreißern handelt es sich um Inkonsistenzen durch das Signieren neuer ROAs oder nicht-identifizierte Auslöser die jedoch nur eine geringe Auswirkung auf die Konsistenz der ASN haben.

Die Messergebnisse verdeutlichen, dass RPKI-Inkonsistenzen durch nicht-identifizierte Anomalien, Synchronisationsprobleme und Signierung in Kombination mit der Verzögerungen eine extreme Auswirkung auf die Konsistenz der ASN haben.

4.2.6 ROA-Unterschiede pro ASN - konsekutive Segmente

Die Analyse der konsekutiven Segmente basiert auf der vorangegangenen ROA-Unterschiedsanalyse pro ASN. Es wird festgestellt, wie hoch der Prozentsatz an ASN ist, die zwei oder mehrere Tage in Folge Teil von Inkonsistenzen sind. Mithilfe dieser Analyse wird untersucht, ob bestimmte ASN über einen längeren durchgängigen Zeitraum während der Messung Teil von Inkonsistenz sind.

Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Für die Analyse werden zunächst die Messergebnisse der ROA-Unterschiedsanalyse pro ASN als Basis genommen. Die Werte werden durch einen Wert ersetzt, der ausschließlich die Präsenz der ASN in den Differenzen beschreibt. Daher entsteht zunächst eine Statistik über die Präsenz einer ASN in den Inkonsistenzen für jeden Tag.

Anschließend wird die kumulative Summe für jede ASN für jeden Tag berechnet, wobei eine Nicht-Präsenz der ASN an einem Tag das Zurücksetzen der Summe zur Folge hat. Daraufhin wird für jeden Tag die Anzahl an ASN mit der gleichen kumulativen Summe berechnet. Die Werte der unterschiedlichen Summen ergeben für diesen Tag den Absolutwert der Menge an ASN, die an diesem Tag eine bestimmte Anzahl an aufeinanderfolgenden Tagen in den Inkonsistenzen präsent waren. Dieser Wert wird daraufhin ins Verhältnis zur Gesamtanzahl an allen präsenten ASN dieses Tages gesetzt.

Die entstehende Statistik kann nun als Messergebnisse visualisiert und weiter analysiert werden.

Messergebnisse

Die Abbildung 4.13 zeigt für jeden Tag die Prozentsätze an ASN, die an aufeinanderfolgenden Tagen in Inkonsistenzen präsent waren.

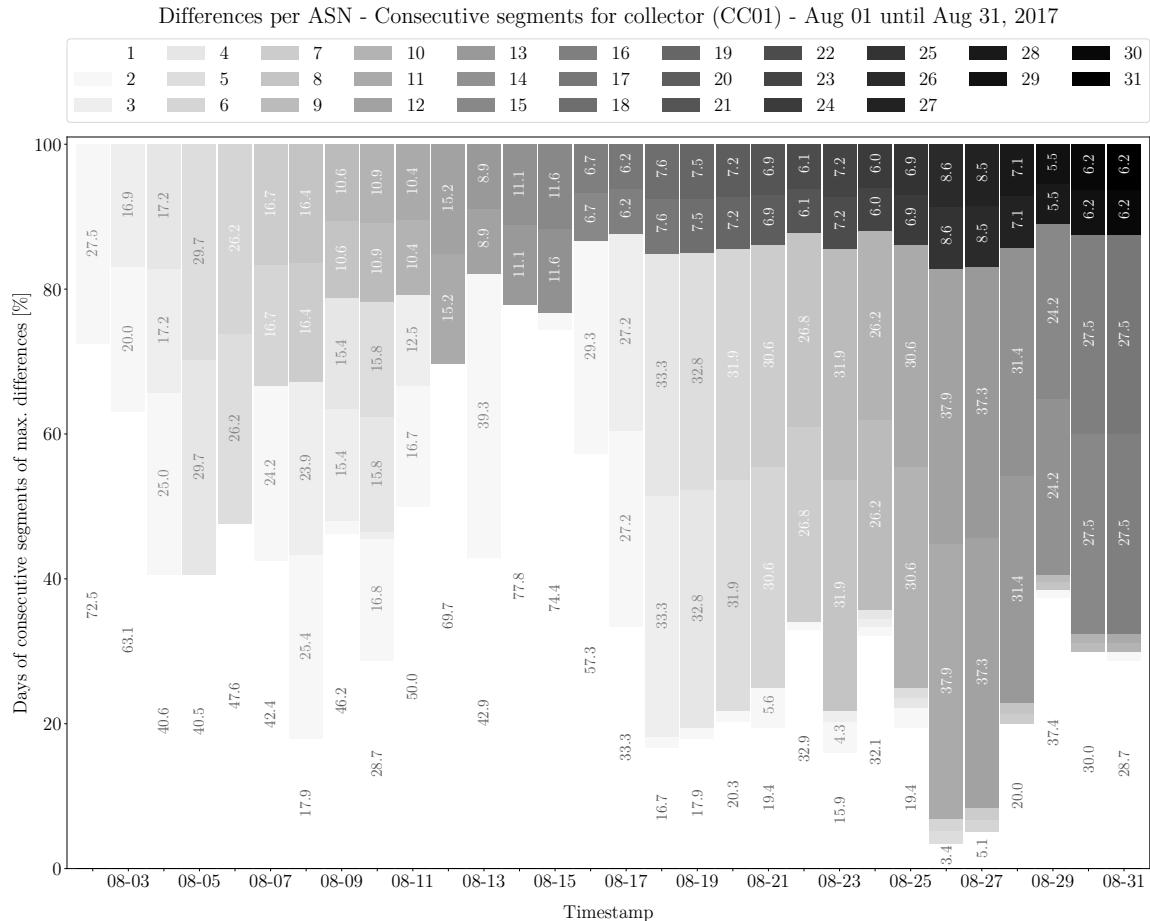


Abbildung 4.13: Messergebnisse der konsekutiven Segmente innerhalb der ROA-Unterschiedsanalyse pro ASN

Die Messwerte werden dabei durch ein gestapeltes Säulendiagramm dargestellt. Die Anzahl an aufeinanderfolgenden Tagen wird dabei durch eine Einfärbung repräsentiert. Der Farbraum verhält sich dabei analog zur Anzahl der Tage. Das Diagramm zeigt auf der X-Achse die einzelnen Tage des Messzeitraums. Auf der Y-Achse sind die unterschiedlichen Prozentsätze der ASN unterteilt nach den Tagen der konsekutiven Segmente. Die Summe der Segmente eines Tages repräsentiert daher die Menge aller ASN, die an einem Tag in den Inkonsistenzen präsent waren.

Die Messergebnisse zeigen, dass die Prozentsätze der konsekutiven Segmente in einem Wertebereich von 5 - 80% liegen. Die Prozentsätze der einzelnen Tage verteilen sich auf dabei auf wenige Segmente. Es fällt auf, dass sich die Messergebnisse in unterschiedliche Verläufe unterteilen lassen. So kann den Verlauf des erstmaligen Erscheinen (1 Tag), den Verlauf des mehrmaligen Erscheinen (2-17 Tage) und den Verlauf des durchgängigen Erscheinen (18-31 Tage) einer ASN aufteilen.

Der Verlauf des erstmaligen Erscheinens (1 Tag) beschreibt, den Prozentsatz an ASN, die an einem Tag der Messung zum ersten Mal in den Inkonsistenzen – nach einer möglichen Unterbrechung – erscheinen. Die Prozentsätze des Verlaufs sind im Bereich von 3 - 78% und werden mit der Farbe weiß dargestellt. Aufgrund der Trivialität wurde der erste Tag der Messung ausgeblendet. Der Prozentsatz des erstmaligen Erscheinens enthält zu dem alle darauffolgenden Prozentsätze des mehrmaligen Erscheinens an den Folgetagen. Der Verlauf beginnt mit einem Wert von 72,5% und fällt bis zum 08.08 leicht schwankend auf einen Wert von 17,9%. Daraufhin steigt dieser erneut bis zu seinem Maximum mit 77,8% am 14.08 und fällt auf einen Wert von 3,4% am 26.08. Am Ende der Messung steigt dieser wieder auf einen Wert von 28,7% am 31.08 an.

Der Verlauf des mehrmaligen Erscheinens (2-17 Tage) beginnt in der ersten Hälfte des Messzeitraums und teilt sich bis zum 11.08 im Laufe der Tage auf die höheren Dauern nahezu gleichmäßig auf. Es folgt daraufhin ein Übergang zum Verlauf des durchgängigen Erscheinens am 14.08-15.08. Dabei beschränken sich die Prozentsätze der beiden Tage ausschließlich auf die maximalen Dauern 13,14 und 14,15 Tage. Es folgt ein erneuter Beginn des Verlaufs ab dem 16.08 der sich bis zum Ende des Messzeitraums – unterhalb des durchgängigen Verlaufs – auf die maximal möglichen Dauern von 16 und 17 Tage am 31.08 aufteilt.

Der Verlauf des durchgängigen Erscheinens (18-31 Tage) beginnt in der Mitte des Messzeitraums. Mit zunehmendem Verlauf der Messung isolieren sich die jeweiligen Maximale der Dauern steigend, so dass beispielsweise am Tag 27.08 ausschließlich die Dauern 26 und 27 Tage präsent sind. Die Prozentsätze schwanken dabei leicht aufgrund der flexiblen Gesamtanzahl an präsenten ASN an den einzelnen Tagen. Der Verlauf des durchgängigen Erscheinens endet am 31.08 mit einem Wert von 6,2% mit den beiden maximalen Dauern von 30 und 31 Tagen.

Auswertung

Zunächst ist festzuhalten, dass Inkonsistenzen, die an aufeinanderfolgenden Tagen auftreten, einen größeren Einfluss auf die BGP-Validierung verursachen als einmalige Inkonsistenzen. Daher werden die beiden Verläufe des mehrmaligen und durchgängigen Erscheinens ausführlicher ausgewertet.

Der Verlauf des mehrmaligen Erscheinens macht deutlich, dass rund die Hälfte (50%) der in den Inkonsistenzen vorkommenden ASN an mehreren aufeinanderfolgenden Tagen präsent sind. Des Weiteren zeigen die Messergebnisse, dass ASN, die an zwei aufeinanderfolgenden Tagen Teil von Inkonsistenzen sind, dies mit hoher Wahrscheinlichkeit auch in den darauf folgenden Tagen sind. Aufgrund dieser beiden Fakten ist die Wahrscheinlichkeit von 50% der ASN als hoch einzustufen.

Der Verlauf des durchgängigen Erscheinens macht deutlich, dass rund 6 - 15% der ASN, die mehrmalig Teil von Inkonsistenzen sind, einer durchgängigen Inkonsistenz zu Grunde liegen. Dieser Prozentsatz enthält jedoch auch ASN, die Teil von RPKI-Beacons sind. Deren Änderungen verursachen aufgrund der Verzögerung täglich Inkonsistenzen, wodurch diese ASN nachfolgend ausgeschlossen werden. Ungeachtet davon sind jedoch die wiederkehrenden Inkonsistenzen des Synchronisierungsproblems ein Beispiel für den Verlauf des durchgängigen Erscheinens. Aufgrund der Regelmäßigkeit dieser Inkonsistenz sind die involvierten ASN an allen Tagen betroffen. sind und somit durchgängig Differenzen erzeugen. Aufgrund des

geringen Auftretens des Auslösers am 05.08, waren einige der betroffenen ASN nicht durchgängig Inkonsistenzen. In diesem Fall traten die ASN nur an 26 aufeinanderfolgenden Tagen in den Inkonsistenzen auf.

Dennoch ist während der Messung aufgefallen, dass unter den betroffenen ASN des Synchronisierungsproblems die ASN 0 enthalten ist. Diese darf laut Standard nicht für den Einsatz im globalen Routing verwendet werden. Durch Nachforschungen wurde festgestellt, dass auf dem delegierten Repository zwei ROA gelagert werden, die dieser ASN fälschlicherweise zugewiesen sind. Als einzige ASN verursacht diese an jedem Tag der Messung zu mindestens einem Zeitpunkt eine Inkonsistenz zwischen den Cache Servern. Daher wird deutlich, dass neben dem bekannten Synchronisierungsproblem auch eine invalide Zugehörigkeit von Präfixen zum ASN 0 erzeugt wurde. Dies kann erhebliche RPKI-Inkonsistenzen zwischen den Cache Servern und analog innerhalb der Validierung von BGP-Announcements erzeugen. Des Weiteren führt das Synchronisierungsproblem zu einer durchgängigen Präsenz der betroffenen ASN in den Inkonsistenzen. Dies hat für die erhebliche Einschränkungen der Konsistenz dieser ASN hinsichtlich BGP-Validierung zur Folge.

4.3 Zusammenfassung

Die Aufgabe der anfänglich durchgeföhrten Meta Analysen war die bestmögliche Isolierung der RPKI-Daten gegenüber Messartefakten. Die Analysen sollten gewährleisten, dass für die folgenden Inkonsistenz-Analyse ausschließlich möglichst reine RPKI-Daten analysiert werden.

Im Falle der Erreichbarkeit wurden Nebeneffekte, wie der Fehler des inkrementellen Updates des RIPE Validators ausgeschlossen. So wurden diesbezügliche Differenzen zwischen den ROA-Datensätzen nicht wie zunächst annehmbar als Inkonsistenz sondern als reines Messartefakt wahrgenommen. Indirekte Inkonsistenzen durch die Nicht-Erreichbarkeit eines Cache Servers wurden identifiziert jedoch aufgrund ihrer Trivialität nicht weiter untersucht.

Im Falle der Delay-Analyse wurden Zeitpunkte bei denen das Archiv den aktuellen ROA-Datensatz aufgrund von Verzögerungen weitaus später erhält zunächst als eine scheinbare Differenz wahrgenommen. Mithilfe der Zeitproblematik wurde jedoch deutlich, dass es sich bei diesen Differenzen nur um Messartefakte handelt. Diese wurden für nachfolgende Analysen ausgeschlossen. Im Falle einer ROA-Datensatz-Beschaffung über das Web-Interface kann diese Zeitproblematik hingegen zu einer RPKI-Inkonsistenz führen. Da dies jedoch nicht dem vorgesehenen Prozess des Standards entspricht, wurde dieser Fall nicht weiter thematisiert.

Die Inkonsistenz-Analysen sollten die Quantifizierung der RPKI-Inkonsistenzen zwischen den Datenbeständen der Cache Server ermöglichen und Ursachen identifizieren. Mithilfe der Beacon-Analyse konnte so festgestellt werden, dass die Verzögerung zwischen dem Cache Server und dem globalen Repository eine Ursache für Inkonsistenzen ist. Weitergehend wurde deutlich, dass die Verzögerung direkte Auswirkungen auf die ROA-Datenbestände der Cache Server hat. Dies hat zur Folge, dass Cache Server erst zu unterschiedlichen Zeitpunkten den gleichen ROA-Datensatz erhalten. Die RPKI-Inkonsistenz ist dabei von der Menge der aktualisierten ROA-Daten abhängig. Der Grad der Inkonsistenz kann durch ein mögliches erneutes Fetching eines schnelleren gegenüber einem langsameren Cache Server

verstärkt werden. Aufgrund der erhobenen Daten wurde jedoch deutlich, dass solch ein erneutes Fetching im Durchschnitt nicht vorkommt.

Die Trust-Anchor-Analyse ermöglichte die Untersuchung der Präsenz der Trust Anchor in den ROA-Datensätzen der Cache Server. Dabei wurde deutlich, dass aufgrund der Verzögerung innerhalb der RPKI unter Umständen die Präsenz einzelner Trust Anchor innerhalb der Datensätze zwischen den Cache Servern unterschiedlich sein kann.

Mithilfe der ROA-Unterschiedsanalyse war es möglich die direkte Differenz der Datensätze der Cache Server untereinander zu analysieren. So wurde festgestellt, dass es während der Messung zu einem durchgängigen Synchronisierungsproblem zwischen dem Repository von APNIC IANA und einem delegierten Repository kam. Die darauf gelagerten ROA-Objekte waren in periodischen Abständen für die Cache Server nicht sichtbar. Des Weiteren ergab die Untersuchung, dass die Signierung von ROA im globalen Repository aufgrund der Verzögerung dazu führt, dass die Einträge der ROA zu unterschiedlichen Zeitpunkten auf den Cache Server sichtbar sind. Es wurde deutlich, dass eine Änderung der ROA durch ein gültiges Ereignis (Signierung von ROA) oder einem ungültigen Ereignis (Synchronisierungsproblem) nur in Kombination mit der Verzögerung zwischen Cache Server und globalen Repository erhebliche einmalige und wiederkehrende Inkonsistenzen zur Folge hat.

Die ROA-Unterschiedsanalyse pro RIR untersuchte die Differenzen zwischen den ROA-Datensätzen der Cache Server hinsichtlich der RIR. Aus der Analyse ging zusammenfassend hervor, dass die Inkonsistenzen durch die Verzögerungen eine durchaus erhebliche Auswirkung auf die Konsistenz des RIR haben. Differenzen unbekannter Ursachen hatten keine eindeutige Auswirkung auf die Konsistenz der RIR.

Mithilfe der ROA-Unterschiedsanalyse pro ASN, konnte festgestellt werden, dass die Verzögerungen eine extreme Auswirkung auf die Konsistenz betroffener ASN hat.

Die darauf aufbauende Untersuchung der konsekutiven Segmente untersucht die Dauer der Präsenz der ASN innerhalb des Messzeitraums. Dabei wurde deutlich, dass Synchronisierungsprobleme dazu führen, dass ASN durchgängig Teil von RPKI-Inkonsistenzen sind. Deren Konsistenz wurde hinsichtlich der folgenden Validierung der BGP-Annoucements stark eingeschränkt. Des Weiteren wurde festgestellt, dass innerhalb der Messung ASN, die an wenigen aufeinanderfolgenden Tagen Teil von Inkonsistenzen sind, mit einer Chance von 50% an den darauffolgenden Tage erneut präsent sind.

Das nachfolgende Kapitel untersucht, ob ein Zusammenhang zwischen den RPKI-Inkonsistenzen und den Validierungsergebnissen der BGP-Annoucements besteht.

KAPITEL 5

Analyse der Auswirkungen der Inkonsistenzen auf BGP

Im Folgenden werden die Auswirkungen von RPKI-Inkonsistenzen auf den BGP-Verkehr untersucht. Entsprechende Analysen werden zeigen, ob die im Kapitel 4 thematisierten Inkonsistenzen direkte bzw. indirekte Auswirkungen auf die Validierung der im Messzeitraum stattfindenden BGP-Announcements haben. Jede Analyse wird hinsichtlich Messmethodik und Durchführung erläutert und die Messergebnisse anschließend ausgewertet.

5.1 Auswirkungen der Inkonsistenzen

Die Analyse untersucht mithilfe der archivierten ROA-Datensätze der Cache Server die Unterschiede der Validierungsergebnisse für alle BGP-Announcements des Messzeitraums.

Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Für die Analyse werden zunächst die Validierungsergebnisse aller BGP-Announcements des Messzeitraums mithilfe der ROA-Datensätze der Cache Server benötigt. Dafür wird – wie in den nachfolgenden Kapiteln genauer erläutert – der BGPSream Client (BGPReader) mit der Erweiterung durch die Bibliothek ROAFetchlib für die zu untersuchenden Cache Server gestartet.

Auch in dieser Analyse wird das Kollektor-Paar CC03 nicht weiter untersucht. CC01(RTR) wird wegen fehlender signifikanter Unterschiede zum Referenz-Kollektor ebenfalls nicht in die Analyse berücksichtigt.

Für die Untersuchung des BGP-Verkehrs werden die Mitschnitte vom Vantage-Point “RRC00“ des Projekts “RIS“ verwendet. Anschließend erfolgt die Validierung aller BGP-Announcements für den erforderlichen Zeitraum mithilfe der ROAFetchlib. Der dabei entstehende Output enthält sowohl die Informationen zu den jeweiligen BGP-Announcements (Timestamp, Origin ASN, Präfix) als auch die Validierungsergebnisse der einzelnen Kollektoren (Status der

Validierung, ASN, Präfix und Max-Länge des ROA). Der Validierungsoutput wird zunächst für die nachfolgenden Analysen hinsichtlich inkonsistenter Validierungsergebnisse gefiltert und als BGP-Inkonsistenz-Datei abgespeichert. Für jeden Zeitpunkt werden die Status der Validierungsergebnisse für jeden einzelnen Kollektoren gezählt. Es entsteht eine Statistik, die für jeden Kollektor und Zeitpunkt eine bestimmte Anzahl an Notfound, Valid und Invalid enthält. Mithilfe dieser Daten kann festgestellt werden, ob zu einem bestimmten Zeitpunkt zwischen dem Referenz- und Vergleichskollektor Unterschiede existieren.

Da es sich bei den BGP-Announcements um reale Daten handelt, können nur Einträge validiert werden, die in den BGP-Announcements enthalten sind. Dies hat zur Folge, dass nur bestimmte Auswirkungen der RPKI-Inkonsistenzen auf die BGP-Daten sichtbar werden. Es werden nur Inkonsistenzen sichtbar bei denen das zu validierende BGP-Announcement zu dem Zeitpunkt der Inkonsistenz auftritt. Daher ist es nicht möglich alle Auswirkungen der RPKI-Inkonsistenzen mit dieser Analysemethode sichtbar zu machen. Eine weitere Analyse, die diese Lücke auf theoretischer Basis schließt, wird in Kapitel 10 näher erläutert.

Nachfolgend werden die Messergebnisse vorgestellt und beschrieben.

Messergebnisse

Die Abbildung 5.1 zeigt für jeden untersuchten Vergleichskollektor die Differenzen hinsichtlich der Validierungsstatus gegenüber dem Referenzkollektor CC01(Web).

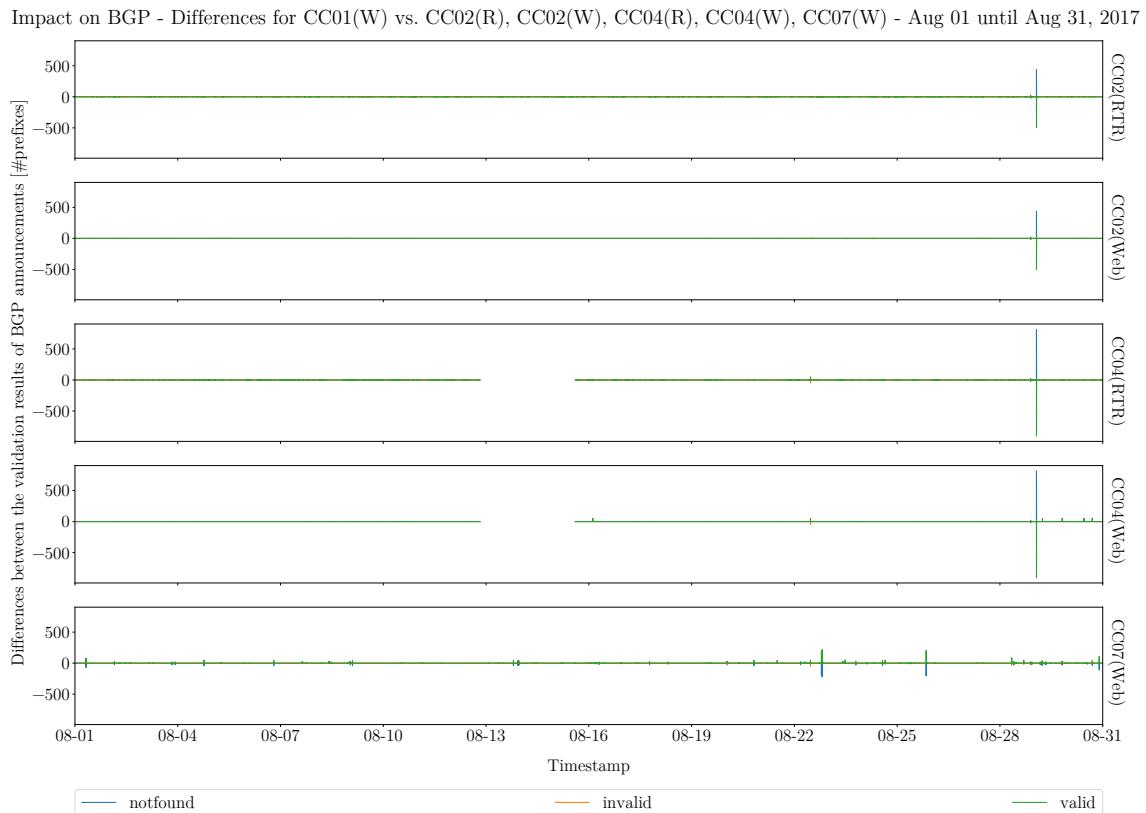


Abbildung 5.1: Messergebnisse der Analyse der BGP-Auswirkungen durch RPKI-Inkonsistenzen [R-RTR/ W-Web]

Bei Übereinstimmung der Anzahl der Validierungsstatus beider Kollektoren kommt es zu einer Überschneidung der Graphen. In der Abbildung sind so nur Abweichungen hinsichtlich Anzahl an unterschiedlich validierten BGP-Announcements sichtbar. Die einzelnen Validierungsstatus besitzen unterschiedlichen Einfärbungen.

Aus den Messergebnissen geht hervor, dass der Großteil der Datenpunkten von den einzelnen Vergleichs- und dem Referenzkollektor konvergieren und alle Datenpunkte in einem Wertebereich von 0 - 500 Präfixen liegen. Die Datenpunkte des Referenzkollektors sind auf der Negativachse dargestellt.

Die aus dem vorherigen Kapitel bereits bekannte Unterbrechung des Kollektor-Paars CC04 ist auf eine Nicht-Erreichbarkeit des Servers zurückzuführen und wird nicht weiter untersucht.

Die Graphen der Vergleichskollektoren der Paare CC02 und CC04 zeigen bis auf wenige Differenzen im geringen Wertebereich von 0 - 10 Präfixe einen extremen Ausschlag im Wertebereich von 430 - 440 Präfixen (CC02) und 810 - 820 Präfixen (CC04) am 29.08. Dabei handelt es sich um die Differenz zwischen dem Vergleichskollektor mit dem Validierungsstatus *notfound* und dem Referenzkollektor mit *valid*. Es fällt auf, dass dieser Ausschlag bei CC07(Web) nicht sichtbar ist, jedoch zwei weitere Ausschläge am 23.08 und 26.08 mit einem Wert von rund 200 Präfixen. Dabei handelt es sich um Differenzen zwischen dem Vergleichskollektor mit dem Validierungsstatus *valid* und dem Referenzkollektor mit *notfound*. Neben diesen Ausschlägen existieren, teilweise in regelmäßigen Abständen, Ausprägungen im geringen Wertebereich von 0 - 50 Präfixen über den gesamten Messzeitraum.

Nachfolgend werden die einzelnen Ausprägungen und deren Ursachen näher erläutert.

Auswertung

Die Messergebnisse lassen Parallelen bei den Differenzen zwischen den BGP-Validierungen und den RPKI-Inkonsistenzen aus Kapitel 4 erkennen.

Eine Parallele ist die starke Differenz zwischen den Vergleichskollektoren CC02 und CC04 und dem Referenzkollektor am 29.08. Dabei handelt es sich um die Auswirkungen der RPKI-Inkonsistenz, die durch das Signieren von ROA zustande kommt. In diesem Fall enthält der ROA-Datensatz des Referenzkollektors eine Vielzahl an Einträgen aus den betroffenen ROA. Diese sind in den ROA-Datensätzen der anderen Vergleichskollektoren nicht enthalten. Dadurch kommt es zu Differenzen zwischen den BGP-Validierungen, da der ROA-Datensatz des Referenzkollektors, aufgrund der passenden ROA, die entsprechenden BGP-Announcements mit einem validen Status (*valid*) validiert. Die Vergleichskollektoren validieren diese hingegen mit einem nicht-vorhandenen Status (*notfound*). Die Auswirkungen dieser RPKI-Inkonsistenz treten an weiteren Stellen während der Messung – jedoch in einem geringen Wertebereich – auf.

Die Inkonsistenzen können sich wie folgt unterschiedliche Kombinationen von BGP-Validierung ergeben:

- *valid – notfound*, durch ein fehlendes ROA
- *invalid – notfound*, durch ein fehlendes ROA
- *valid – invalid*, bei der Signierung des ROA (ASN oder Max-Länge verändert)

Im Falle von CC07 ist die Verzögerung, die beim Fetching der signierten ROA entsteht vergleichsweise niedrig. Dadurch entsteht die RPKI-Inkonsistenz im Gegensatz zu den anderen Kollektoren nur für einen kurzen Zeitraum. Zum Zeitpunkt des passenden BGP-Announcement waren die ROA-Datensätze des Vergleichskollektors CC07(Web) und des Referenzkollektors CC01(Web) bereits wieder identisch. Daher ist die RPKI-Inkonsistenz im Falle von CC07 aufgrund der Unbestimmtheit der Vorkommen der BGP-Announcements nicht sichtbar.

Eine weitere Parallele zwischen den Differenzen der BGP-Validierung und den RPKI-Inkonsistenzen sind die teilweise regelmäßigen Ausprägungen in einem geringen Wertebereich. Dabei handelt es sich um die wiederkehrenden RPKI-Inkonsistenzen, die durch das Synchronisierungsproblem ausgelöst wurden. Diese RPKI-Inkonsistenzen spiegeln sich in der BGP-Validierung wider. Die Stärke und Vorkommen der BGP-Inkonsistenzen ist dabei abhängig von der Menge der betroffenen Präfixe in den BGP-Announcements. Dies hat erneut zur Folge, dass nicht alle RPKI-Inkonsistenzen auf der BGP-Ebene einheitlich sichtbar sind.

Eine weitere Auffälligkeit der Messergebnisse sind die Ausschläge von CC07(Web). Diese BGP-Inkonsistenzen haben jedoch keine Parallele zu den analysierten RPKI-Inkonsistenzen. Ursache dafür ist, dass während der ROA-Unterschiedsanalyse nur Trust Anchor der RIR untersucht wurden. In den ROA-Datensätzen sind jedoch nach wie vor alle originalen Trust Anchor enthalten. Im Falle von CC07(Web) werden somit BGP-Announcements mit Einträgen von Trust Anchor validiert, die nur auf CC07(Web) konfiguriert sind. Es wird deutlich, dass die Einbindung unterschiedlicher Trust Anchor ebenfalls zu indirekten RPKI-Inkonsistenzen zwischen den ROA-Datensätzen der Cache Server und folglich auch zu BGP-Inkonsistenzen führen. Da die indirekten RPKI-Inkonsistenzen in den Analysen nicht weiter untersucht wurden, werden die BGP-Inkonsistenzen ebenfalls nicht analysiert.

Abschließend wird deutlich, dass die RPKI-Inkonsistenzen, die durch die erwähnten Auslöser in Kombination mit der Verzögerung verursacht wurden, ebenfalls BGP-Inkonsistenzen erzeugen und somit direkte Auswirkungen auf die BGP-Validierung haben.

Nachfolgend werden die einzelnen Auswirkungen auf die betroffenen ASN unterteilt.

5.2 Auswirkungen der Inkonsistenzen pro ASN

Die Analyse untersucht die Differenzen der Validierungsergebnisse pro ASN für alle BGP-Announcements des Messzeitraums mithilfe der archivierten ROA-Datensätze der Cache Servern. Es wird festgestellt, in welchem Maß die in Abschnitt 4.2.5 detektierten RPKI-Inkonsistenzen pro ASN Auswirkungen auf die Konsistenz der BGP-Validierung pro ASN haben. Dies ermöglicht einen Vergleich zwischen der Verteilung an BGP-Inkonsistenzen pro ASN und der Verteilung an RPKI-Inkonsistenzen pro ASN.

Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Für die Analyse wird zunächst die bereits erwähnte BGP-Inkonsistenz-Datei verwendet. Dort enthalten sind alle BGP-Announcements, die eine BGP-Inkonsistenz enthalten. Anschließend können für alle BGP-Announcements mit Inkonsistenzen die Vorkommen der

Origin-ASN für jeden Zeitstempel gezählt werden. Analog dazu werden die Vorkommen der Origin-ASN für alle BGP-Announcements mit oder ohne Inkonsistenzen gezählt. Anschließend kann das Verhältnis dieser beiden Werte für jeden Zeitstempel berechnet werden. Es entsteht eine Statistik darüber, wie oft ein ASN Teil einer BGP-Inkonsistenzen gegenüber der Gesamtanzahl war. Daraufhin wird das Maximum dieses Verhältnisses für jede ASN für jeden Tag des Messzeitraum berechnet.

Diese Messmethodik ist somit das Gegenstück der Messmethodik der ROA-Unterschiedsanalyse pro ASN auf BGP-Ebene. Dies ermöglicht einen direkten Vergleich der Messergebnissen.

Messergebnisse

Die Abbildung 5.2 zeigt für jeden Messtag die in der Messmethodik beschriebene Verteilung der Maximalwerte zwischen der Anzahl der BGP-Announcements mit Inkonsistenzen in die ein ASN involviert ist und die Gesamtanzahl an BGP-Announcements mit einer betroffenen ASN. Die Messwerte werden dabei durch einen Box-Plot dargestellt, wobei jede Verteilung eines Messtages durch eine Box gemäß der Abbildung 4.7 wiedergegeben wird. Da während der Messung für den letzten Tag 31.08 keine Inkonsistenzen aufgezeichnet wurden, wird hier und im Folgenden der Messzeitraum 01.08 - 30.08 betrachtet.

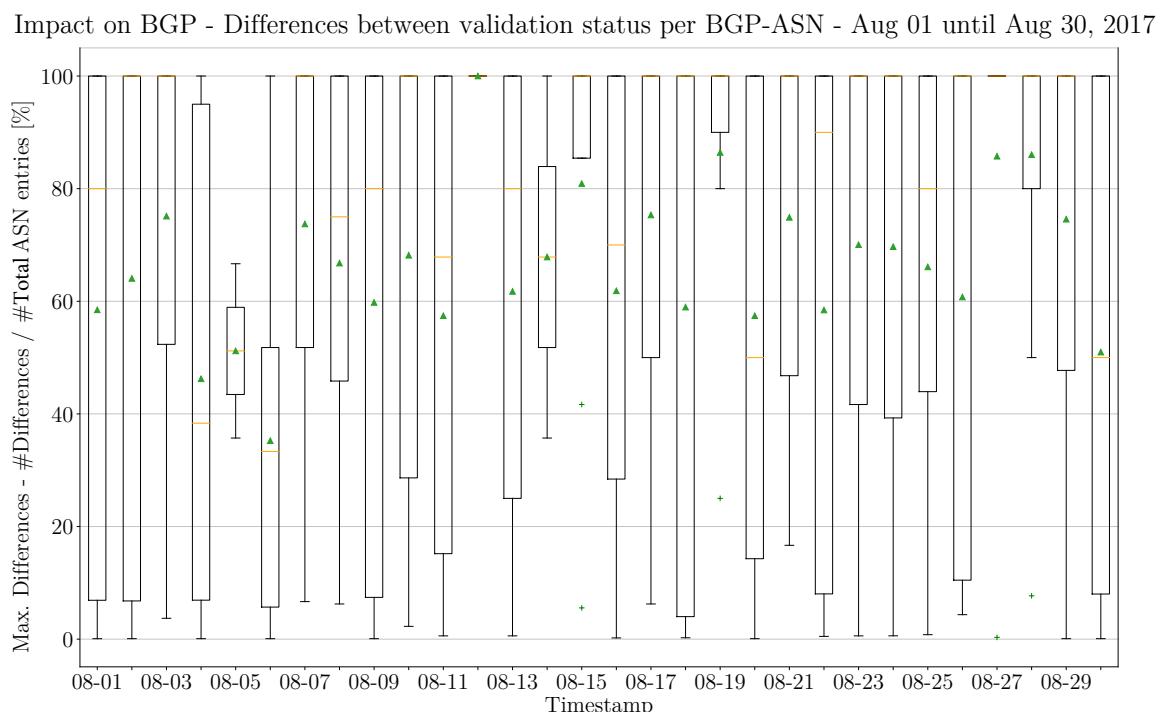


Abbildung 5.2: Messergebnisse der Analyse der BGP-Auswirkungen durch RPKI-Inkonsistenzen pro ASN

Die Messergebnisse zeigen, dass der Verlauf der maximalen Differenzen auf BGP-Ebene eine starke Ähnlichkeit zur RPKI Ebene besitzt. So liegen die Datenpunkte aller Maximalwerte in einem stark durchwachsenen Wertebereich von 0 - 100%. Aufgrund der Definition der maximalen Differenz kann es erneut keine Werte oberhalb der 100% geben. Analog kann der Maximalwert von exakt 0% nicht im Intervall enthalten sein, da stets mindestens eine Inkonsistenz enthalten sein muss. Die Verteilungen der einzelnen Messtage können erneut in drei unterschiedliche Typen I, II, III unterteilt werden.

Die Mehrheit der Verteilungen ist vom Typs I, dessen oberes Quartil bei 100% und das untere Quartil bei einem variablen Wert im Bereich von 5 - 60% liegt. Ein Beispiel für diesen Typ sind die Tage 20.08 bis 26.08. Diese Verteilungen sind ähnlich zur RPKI-Ebene recht einseitig zur 100%, wobei der Durchschnitt etwas geringer mit einem Bereich von 30 - 60% ausfällt. Die Mehrheit der Durchschnitte liegt bei rund 60 - 65%. Trotz der breit aufgefächerten Verteilung existieren in einigen Fällen Ausreißer am unteren Ende der Whisker. Der Median liegt im Gegensatz zur RPKI-Ebene an mehreren Tagen dieses Typs unterhalb der 100% im Wertebereich von 30 - 90%.

Die Verteilungen des Typ II definieren sich erneut durch keine Einseitigkeit der Messwerte. Ein Beispiel für diesen Typ ist die Verteilung der Maximalwerte am 05.08. Das obere Quartil liegt dabei in einem Wertebereich von rund 60% und das untere Quartil bei rund 45%. Der obere Whisker reicht bis zu einer Grenze von 100% und der untere Whisker bis knapp 0%. Im Gegensatz zu den Inkonsistenzen auf RPKI-Ebene ist die Anzahl an Verteilungen des Typs II leicht erhöht. Sowohl der Durchschnitt als auch der Median liegen in einem Wertebereich von 35 - 70%. Es existieren aufgrund dieser Eigenschaften der Verteilung keine Ausreißer an den Enden der Verteilung.

Die Verteilungen des Typs III fallen aufgrund ihrer extremen Einseitigkeit ähnlich zur RPKI-Ebene auf. Die Anzahl an Verteilungen dieses Typs liegt gegenüber den Messergebnissen der RPKI-Inkonsistenzen in einem geringeren Wertebereich. Diese Art der Verteilung kommt nur an zwei unterschiedlichen Messtagen innerhalb der Messung auf (12.08 und 27.08). Aufgrund der Einseitigkeit kommt es zu Überlappungen der einzelnen Elemente der Boxen. Alle Elemente der Box liegen – mit Ausnahme des Durchschnitts und der Ausreißer – bei einem Wert von 100%. In diesem Fall liegt der Durchschnitt in einem Wertebereich von 85 - 100%. Im Gegensatz zu den Messergebnissen der ROA-Unterschiedsanalyse pro ASN existieren bei diesem Typ der Verteilung kaum bis keine Ausreißer.

Auswertung

Aus dem Vergleich zwischen den RPKI- und BGP-Inkonsistenzen pro ASN geht deutlich hervor, dass die Messeergebnisse einen ähnlichen Verlauf der Verteilungen der maximalen Differenzen darstellen. Die Messergebnisse auf BGP-Ebene weisen jedoch einen leicht geringeren Wertebereich hinsichtlich der Anzahl Verteilungen des Typs I und III auf. Somit wird deutlich, dass die RPKI-Inkonsistenzen nicht nur eine erhebliche allgemeine Auswirkung auf die Konsistenz des BGP-Verkehrs besitzen sondern im Speziellen auch auf die Konsistenz des BGP-Verkehrs hinsichtlich der der Origin-ASN.

Nachfolgend wird eine Analyse hinsichtlich der Zusammensetzung invalider Validierungsergebnisse vorgestellt und erläutert.

5.3 Zusammensetzung invalider Validierungsergebnisse

Die Analyse untersucht die Auswirkungen der RPKI-Inkonsistenzen auf die Validierung von BGP-Announcements hinsichtlich des invaliden Validierungsstatus. Die Validierungszustände *valid* und *notfound* treten nur ein, wenn der BGP-Präfix in dem ROA-Datensatz verfügbar ist und die entsprechende ASN übereinstimmt (*valid*) oder wenn der BGP-Präfix nicht im ROA-Datensatz vorkommt (*notfound*). In diesen eindeutigen Fällen bedarf es keine weiteren Untersuchungen. Der Validierungszustand *invalid* tritt hingegen in genau zwei Fällen auf:

- wenn der BGP-Präfix im ROA-Datensatz vorhanden ist, die ASN jedoch nicht übereinstimmt oder
- wenn Subnetzmaske des BGP-Präfixes höher als die Max-Länge des Eintrags im ROA-Datensatz ist.

Diese Analyse soll es ermöglichen, die Auswirkungen der RPKI-Inkonsistenzen auf die BGP-Validierung hinsichtlich dieser beiden Fälle genauer zu untersuchen. Nachfolgend wird die dafür verwendete Messmethodik vorgestellt und näher erläutert.

Messmethodik und Umsetzung

Für die Analyse wird zunächst erneut die BGP-Inkonsistenz-Datei verwendet. Mithilfe dieser Datei können alle BGP-Inkonsistenzen gefiltert werden, die einen invaliden Validierungsstatus enthalten. Anschließend wird für jedes inkonsistent validierte BGP-Announcement die einzelnen Validierungsergebnisse hinsichtlich der invaliden Zustände untersucht. Dafür wird geprüft, ob ein Validierungsergebnis enthalten ist, das einem der beiden Fälle entspricht. Anschließend jeweiligen Vorkommen der beiden Fälle für jedes BGP-Announcement gezählt. Im Gegensatz zu den vorangegangenen Analysen werden daraufhin die absoluten Werte für jeden Tag des Messzeitraums summiert und ins Verhältnis zur Gesamtanzahl an inkonsistenten und invaliden Zuständen gesetzt. Somit entsteht eine Statistik über das Verhältnis der beiden Gründe für ein invalides Validierungsergebnis und die Gesamtanzahl an invaliden und Zuständen innerhalb der BGP-Inkonsistenzen.

Die erhaltenen Daten können ohne weitere Aufbereitungen direkt visualisiert und anschließend ausgewertet werden.

Messergebnisse

Die Abbildung 5.3 zeigt für jeden Messtag die Verteilung der beiden Gründe für ein invalides Validierungsergebnis. Die Messwerte der Verteilungen werden dabei durch ein gestapeltes Säulendiagramm dargestellt. Des Weiteren ist die Gesamtanzahl als einfachen Graphen überlagernd visualisiert. Aufgrund fehlender (invalider) Inkonsistenzen an zwei Tagen der Messung (03.08 und 31.08), werden diese Tage nicht in die Analyse integriert.

Die Messergebnisse zeigen, dass der Verlauf der Verteilung der beiden Gründe für ein invalides Validierungsergebnis sehr schwankend ist. Die Gesamtanzahl verhält sich ebenfalls schwankend und liegt dabei in einem Wertebereich von 0 - 80 invaliden BGP-Inkonsistenzen pro Tag.

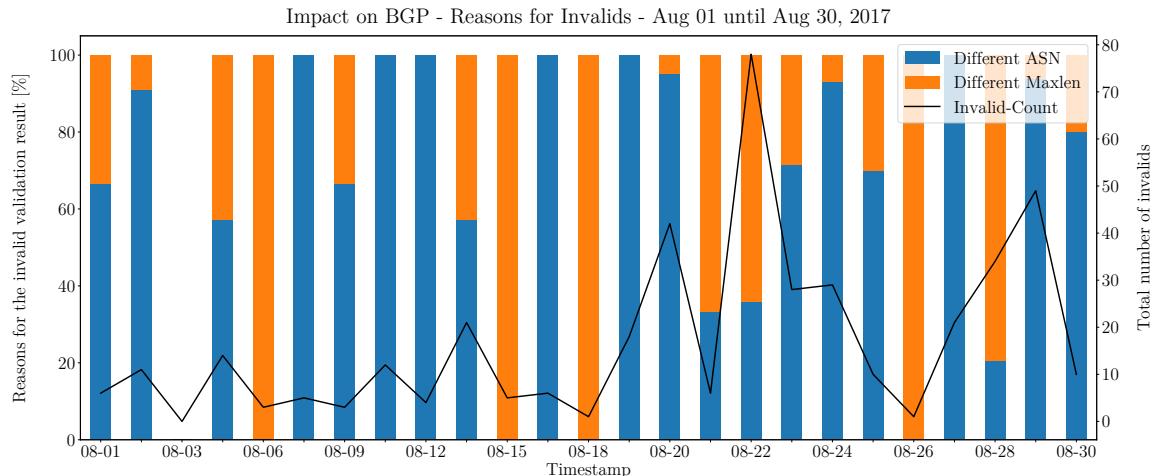


Abbildung 5.3: Messergebnisse der Analyse der Zusammensetzung invalider Validierungsergebnisse

Die Gesamtanzahl der invaliden BGP-Inkonsistenzen liegt innerhalb der ersten Tage des Messzeitraums in einem Bereich von 0 - 15 Inkonsistenzen. Daraufhin kommt es zu einem leichten Anstieg am 14.08 auf einen Wert von 20 Inkonsistenzen. Nach einem kurzzeitigen Abfallen der Werte kommt es zu einem hohen Anstieg am 20.08 auf einen Wert von knapp 50 und am 22.08 auf den Maximalwert von rund 80 Inkonsistenzen. Bis zum 26.08 fällt die Gesamtanzahl in einen Wertebereich von rund 0 - 5 Inkonsistenzen, woraufhin es einen erneuten Anstieg auf einen Wert von rund 50 invaliden Inkonsistenzen.

Für die Verteilung der beiden Gründe ist über den Messzeitraum kein eindeutiger Verlauf über mehrere Messtage hinweg erkennbar. An sechs Tagen des Messzeitraums kommt es zu einer Verteilung von 100% des Grunds einer unterschiedlichen ASN. An vier Tagen kommt es hingegen zu einer Verteilung von 100% des Grunds einer höheren Max-Länge. Des Weiteren beträgt der Durchschnitt im Falle einer unterschiedlichen ASN rund 64% und im Falle einer unterschiedlichen Max-Länge rund 36%. Der Durchschnitt enthält dagej doch nicht den Faktor der Gesamtanzahl der invaliden Validierungsergebnisse. Im Fall des 22.08 entspricht ein großer Anteil der gesamten Validierungszustände dem Grund der unterschiedlichen Max-Länge.

Auswertung

Aus den Betrachtungen der Messergebnisse geht hervor, dass die Gesamtanzahl an invaliden Validierungsergebnissen stark schwankend ist. Einige Tage mit hohen Anstiegen sind jedoch bereits aus den Inkonsistenz-Analysen bekannt. Die beiden Anstiege an den Tagen 14.08 und 29.08 sind Beispiele für die Inkonsistenzen durch das Signieren von ROA. Anhand dieser Tage ist sichtbar, dass sich diese RPKI-Inkonsistenzen erneut auf die BGP-Validierung in Form erheblicher BGP-Inkonsistenzen auswirken. Während der Signierung von ROA kommt es zu unterschiedlichen ROA-Datensätzen der Cache Server, wobei einige die Einträge der ROA bereits enthalten und andere nicht. Das führt dazu, dass BGP-Announcements mit den vollständigen ROA-Datensätzen als *valid* validiert und mit den unvollständigen ROA-Datensätzen als *invalid* validiert werden. Somit kam es während der Signierung der ROA

zu einem Hinzufügen einer weiteren ASN, die die Verteilung entsprechend beeinflusst hat. Somit ist neben der Gesamtanzahl an invaliden Validierungsergebnissen auch die Menge an Validierungen durch diesen Grund von der Signierung der ROA abhängig.

Für den Anstiegen und die Verteilung der beiden weiteren Anstiege kann keine genaue Ursache definiert werden. Nachfolgend werden die Analysen über die Auswirkung der RPKI-Inkonsistenzen auf den BGP-Verkehr zusammengefasst und deren Kernaussagen resümiert.

5.4 Zusammenfassung

Mithilfe der allgemeinen Analyse der Auswirkungen auf BGP war es möglich den allgemeinen Einfluss der RPKI-Inkonsistenzen auf die Validierungsergebnisse der zeitlich passenden BGP-Announcements genauer zu untersuchen. Es wurde deutlich, dass die RPKI-Inkonsistenzen, die durch gültige Ereignisse innerhalb der RPKI – wie das Signieren von ROA – oder ungültige Ereignisse – wie ein Synchronisierungsproblem – in Kombination mit einer Verzögerung verursacht wurden, eine direkte Auswirkungen auf die BGP-Validierung haben und diese somit maßgeblich verändern.

Im Falle der Analyse der Auswirkungen auf BGP pro ASN wurden die Auswirkungen der BGP-Inkonsistenzen, die durch die RPKI-Inkonsistenzen verursacht wurden, hinsichtlich der Konsistenz der betroffenen ASNs untersucht. Dabei ging hervor, dass die RPKI-Inkonsistenzen im Speziellen auch Einfluss auf die Konsistenz der Validierung der BGP-Announcements von Origin-ASN haben.

Im Falle der Analyse der Zusammensetzung invalider Validierungsergebnisse wurde die Verteilung der invaliden Validierungszustände der BGP-Inkonsistenzen genauer betrachtet. Dadurch konnte festgestellt werden, dass die Verteilung der invaliden Zustände stark von den Änderungen abhängt, die während einer RPKI-Inkonsistenz durch eine Verzögerung verursacht werden.

KAPITEL 6

Zwischenfazit: Cache-Inkonsistenz

Die einzelnen Analysen zur Untersuchung der Inkonsistenzen zwischen Cache Servern und den sich daraus ergebenden Auswirkungen auf BGP ergaben jeweils einige wichtige Aspekte. Im Folgenden wird deren Einfluss auf das gesamte Ökosystem bewertet.

Während der Analysen wurde zunächst die Existenz von Inkonsistenzen zwischen den untersuchten Cache Servern nachgewiesen. Es wurde deutlich, dass diese Inkonsistenzen durch verschiedene Auslöser hervorgerufen werden. So wurde festgestellt, dass das Signieren von ROA als legitimes Ereignis der RPKI innerhalb des globalen Repository Inkonsistenzen auslöst. Analog dazu können nicht-legitime Ereignisse, wie Synchronisierungsprobleme zwischen einem Repository eines RIR und einem eingebundenen delegierten Repository, ebenfalls Differenzen zwischen den Cache-Server-Einträgen veranlassen. Letztgenannte Auslöser führen jedoch nur in Kombination mit einer Verzögerung zwischen den einzelnen Cache Servern und dem globalen Repository innerhalb der RPKI zu Inkonsistenzen. Dies geschieht, da aufgrund der Verzögerung die einzelnen Cache Server erst zu unterschiedlichen Zeitpunkten den selben Datenbestand führen. Verzögerung ließ sich bei allen bekannten Auslösern als Ursache für die RPKI-Inkonsistenzen zwischen den Cache Servern identifizieren, wobei der Wertebereich der Unterschiede vom jeweiligen Auslöser abhängt. Die Differenzen entstehen auf den Cache Servern und können an die nachfolgenden Komponenten – BGP-Router – weitergeben.

Die Auswirkungsanalysen ergaben, dass die RPKI-Inkonsistenzen einen direkten Einfluss auf BGP – insbesondere auf die Validierung der BGP-Announcements – haben, da die Updates mit inkonsistenten Datenbeständen der einzelnen Cache Server validiert werden. Im Detail wird auch die Konsistenz der betroffenen AS beeinflusst.

Somit wird deutlich, dass die Verzögerung innerhalb der Infrastruktur die Vertrauenswürdigkeit, die durch die RPKI-Daten den BGP-Routern hinzugefügt werden, erheblich beeinflusst. Dadurch wird der Effekt der Autorisierung zur Verbesserung der Sicherheit des BGP-Verkehrs vermindert. Ursache dafür ist, dass die BGP-Router anhand dieser Informationen ihr Routing-Verhalten variieren. BGP-Router, die an verschiedene Cache Server mit unterschiedlichen Verzögerungen angebunden sind, besitzen unterschiedliche Vertrauenswürdigkeitsinformationen und weisen ein unterschiedliches Routing-Verhalten auf.

KAPITEL 7

Entwurf und Implementierung eines ROA-Dump-Archivs

Für die Durchführung der zuvor genannten Analysen, muss ein konsistentes Archiv für ROA-Dumps existieren. Dieses ist Teil einer Messinfrastruktur, die in die RPKI-Infrastruktur eingreift, um so die entscheidenden ROA-Daten zu erhalten und zu archivieren.

7.1 Messinfrastruktur

Mithilfe der Messinfrastruktur sollen in regelmäßigen Abständen die aktuellen Zustände der ROA-Datensätze der RPKI Cache Server abgefragt und archiviert werden. Dazu muss die Messinfrastruktur in der bestehenden RPKI die Position des BGP-Routers einnehmen, so dass sich die in Abbildung 7.1 gezeigte Gesamt-Infrastruktur ergibt.

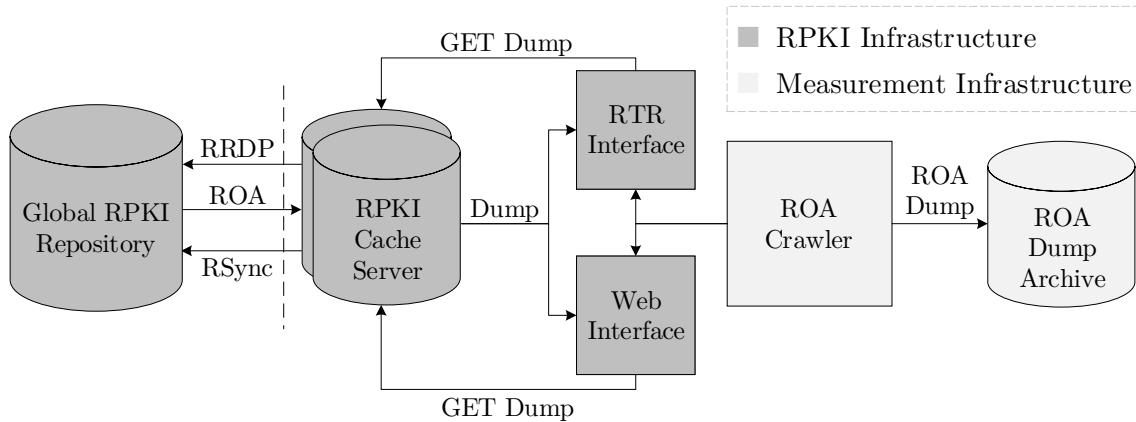


Abbildung 7.1: Entwurf der Messinfrastruktur

Die Messinfrastruktur besteht somit aus einem ROA-Crawler und dem eigentlichen ROA-Dump-Archiv. Diese und deren Aufgaben werden nachgehend einzeln beschrieben.

7.2 Implementierung der ROA-Crawler

Der ROA-Crawler übernimmt innerhalb der Messinfrastruktur die Aufgabe der Datenbeschaffung, in dem er den aktuellen ROA-Datensatz aller oder einzelner konfigurierter Cache Server (Kollektor) abfragt und diesen in ein einheitliches ROA-Dump speichert. Das gesamte Projekt des ROA-Crawlers wird in Python umgesetzt. Die im ROA-Crawler-Modul zur Spezifizierung der Cache-Server verwendete Konfiguration wird nachfolgend näher erläutert.

7.2.1 Konfiguration

Die Konfiguration des ROA-Crawlers ermöglicht die Festlegung verschiedener Parameter für die Durchführung der einzelnen ROA-Crawler-Instanzen:

- Startzeit
- Archiv-Speicherort
- Kollektor-Details (Projekt, Bezeichnung, URL, Web-Port, RTR-Port und Intervall)

Damit sind alle notwendigen Kollektor-Informationen vorhanden, um die jeweiligen ROA-Datensätze anzufragen. Das Intervall beschreibt dabei die Anzahl an Sekunden, die zwischen zwei zu erzeugenden ROA-Dumps liegen soll. Grundsätzlich kann jeder Kollektor ein individuelles Intervall erhalten.

Um Inkonsistenzen zwischen den Kollektoren zu untersuchen, ist es unabdingbar, dass alle Kollektoren im gleichen Intervall abgefragt und archiviert werden, da in den einzelnen Analysen alle ROA-Datensätze aller Cache Server zu einem bestimmten Zeitpunkt betrachtet werden. Dieses ROA-Dump beschreibt somit den Zustand des ROA-Datensatzes des Cache Server von einem Zeitpunkt bis zum Intervall-Ende. Haben die Intervalle unterschiedliche Parameter, gibt es keinen einheitlichen Start- und Endzeitpunkt für die jeweiligen Dumps. Da in den jeweiligen Zeit-Differenzen Änderungen am ROA-Datensatz entstanden sein können, können diese semantisch nicht verglichen werden ohne Messartefakte ausschließen zu können.

Im Nachfolgenden wird von einem einheitlichen Intervall von 180 Sekunden zwischen zwei ROA-Dumps ausgegangen.

Die Konfiguration umfasst zwei Teilbereiche: zum einen alle Kollektoren, die über eine Web-Schnittstelle angesprochen werden und zum anderen alle Kollektoren, die über eine RTR-Schnittstelle mithilfe des RTR-Protokolls abgefragt werden. Diese Aufteilung ist notwendig, da die beiden Schnittstellen von unterschiedlichen Crawler-Typen abgefragt werden müssen. Die Validierung der untersuchten Cache Server wird stets vom RIPE Validator durchgeführt. Daher sind beide Schnittstellen bei allen untersuchten Kollektoren über diesen erreichbar. Im Ergebnis ergibt sich für jeden Kollektor stets ein Web- und ein RTR-Eintrag in der Konfiguration. Die beiden Schnittstellen werden als unterschiedliche Kollektoren eines Kollektor-Paars angesehen.

Für die vorliegende Arbeit wurden die folgenden Kollektoren verwendet:

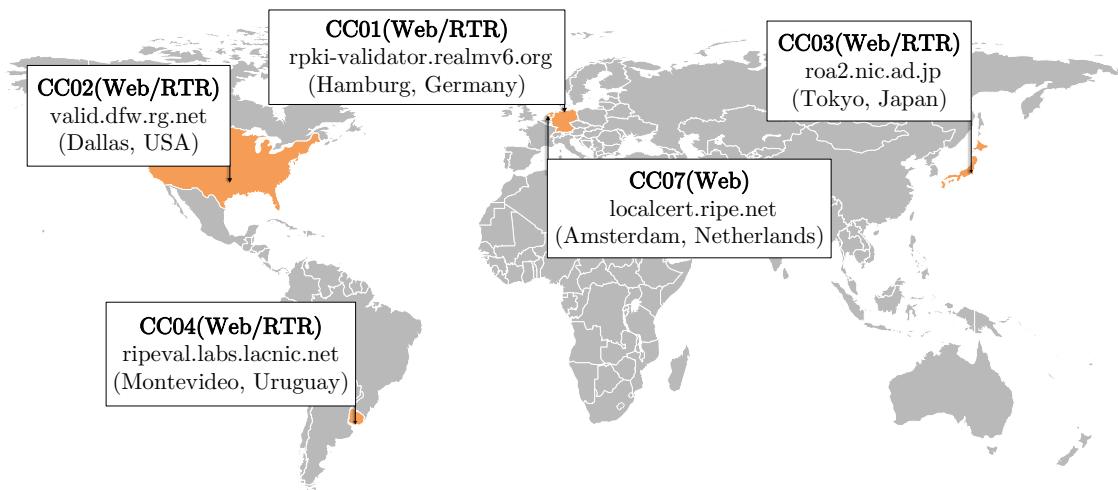


Abbildung 7.2: Standorte der konfigurierten Kollektoren

Nachfolgend werden die beiden unterschiedlichen Crawler-Typen und deren Implementierung vorgestellt.

7.2.2 Web-Sync

Aufgabe des Web-Sync-Moduls ist die periodische Speicherung des aktuellen ROA-Datensatzes eines Kollektors über eine Web-Schnittstelle. Dabei gibt es zwei unterschiedliche Phasen: das periodische Ausführen und die adäquate Speicherung des Dumps.

Das periodische Ausführen - der Start und die einzelnen periodischen Instanzen - erfolgt mithilfe eines Background-Schedulers des Moduls “APScheduler“. Dadurch ist es möglich, exakte Ausführungszeiten festzulegen und diese als Instanz als Hintergrundprozess laufen zu lassen. Der Startzeitpunkt ergibt sich aus der Konfiguration des ROA-Crawlers, wobei die Zeitpunkte der einzelnen Ausführungen durch das allgemeine Scheduling bestimmt werden. Da ist von verschiedenen Faktoren abhängig.

Das Skript wird mithilfe zweier Parameter ausgeführt: die Bezeichnung des Kollektors und ein Scheduling-Parameter, der beschreibt, ob dieser Prozess autark ausgeführt werden soll. Durch den Background-Scheduler wird die Ausführung der Speicherungsmethode auf den Konfigurationsstartzeitpunkt festgelegt. Ist dieser erreicht, wird der Prozess mithilfe der zuvorgenannten Parameter und dem Speicherort des Archivs ausgeführt. Zunächst werden die einzelnen Informationen der Konfigurationsdatei für diesen Kollektor entnommen und die notwendige File-Struktur aufgebaut. Da während des Web-Sync-Prozesses bereits Daten für die Analysen Delay und Reachability aufgezeichnet werden, werden die notwendigen Dateien – falls erforderlich – ebenfalls angelegt.

```

1  ## Parse shell-arguments and configurations from config-file
2  args = parse_shell('c'); cfg = parse_const();
3  content = parse('web', args.collector)
4  arguments = [cfg.archive_path, args.collector, 1 if args.mode != None else 0]

```

```

6  ## Add scheduler-job to start sync at start-time
7  job = sched.add_job(sync, 'interval', seconds=int(content[0].split(',')[-1]), \
8      name=args.collector, start_date=cfg.start_time, args=arguments)
9  sched.add_listener(job_listener, events.EVENT_JOB_ERROR)
10 while True:
11     time.sleep(1)

```

Quellcode 7.1: Start des Background-Schedulers des ROA-Crawlers

Nachfolgend wird die Kernaufgabe des Moduls durchgeführt – das Fetching der ROA-Daten durch die Web-Schnittstelle mithilfe der URL. Währenddessen werden die Erreichbarkeitszustände protokolliert: erreichbar, unerreichbar oder aufgrund von internen Netzwerkproblemen nicht auflösbar. Bei Nicht-Erreichbarkeit und Netzwerkproblemen wird der Prozess abgebrochen und bei der nächsten periodischen Ausführung erneut durchlaufen. Selbiges gilt, falls der ROA-Datensatz aufgrund von Anomalien leer sein sollte. Im Falle einer regulären Erreichbarkeit der Kollektor-Adresse wird die Zeitdifferenz der Übertragung im Millisekunden-Bereich aufgezeichnet und der Delay-Analyse-Datei hinzugefügt. Der nun lokal verfügbare ROA-Datensatz kann anschließend komprimiert als ROA-Dump archiviert werden.

```

1  try:
2      f = urllib2.urlopen(url)
3      with open(os.path.basename(url), 'wb') as f_name: f_name.write(f.read())
4  except urllib2.HTTPError as err:
5      with open(reach_file, 'a') as target: target.write(req_ts+ ',0\n')
6      logging.warning('HTTP Error: ' + str(err.code))
7      logging.warning('Fatal Error: Could not reach the server: ' + url)
8      raise Exception
9  except urllib2.URLError as err:
10     err_no = err.args[0][0]
11     logging.warning('URL Error: ' + str(err.reason))
12     if(err_no == -3):
13         with open(reach_file, 'a') as target: target.write(req_ts+ '-2\n')
14         logging.warning('Transport Error: Address could not be resolved')
15     else:
16         with open(reach_file, 'a') as target: target.write(req_ts+ ',0\n')
17         logging.warning('Fatal Error: Could not reach the server: ' + url)
18         raise Exception

```

Quellcode 7.2: Fetching der ROA-Daten und Meta-Protokollierung des ROA-Crawlers

Je nach Scheduling-Parameter wird das entsprechende (Re-)Scheduling ausgeführt, um bei einer möglichen Asynchronität den Zeitpunkt der nächsten Ausführung zu beeinflussen (nähere Erläuterungen in 7.2.4).

```

1  ## Check whether collector is synced with the common next timestamp
2  next_timestamp = check_next_common_timestamp(0)
3  current_timestamp = (next_timestamp[0] - timedelta(seconds=int(interval))).\
4      strftime('%Y-%m-%d %H:%M')
5  if(all_mode and current_timestamp != request_time.strftime('%Y-%m-%d %H:%M')):
6      all_sync = 1
7  raise Exception
8  if all_mode:
9      utc_next_timestamp = check_next_common_timestamp(1)
10     utc_current_timestamp = (utc_next_timestamp[0] - timedelta(seconds=int(interval\
11         ))).strftime('%Y-%m-%d %H:%M')
10     logging.info('common current timestamp: ' + utc_current_timestamp)

```

```

11     logging.info('The job is synced with the common next timestamp (TS, number of
12     collectors): ' + \
      utc_next_timestamp[0].strftime('%Y-%m-%d %H:%M') + ', ' + str(
      utc_next_timestamp[1]))

```

Quellcode 7.3: Ausführung des entsprechenden (Re-)Schedulings des ROA-Crawlers

Um zwischen RPKI-Inkonsistenzen und Netzwerkanomalien klar unterscheiden zu können, verfügt das Web-Sync-Modul über einen Logging-Mechanismus. Dieser fügt der entsprechenden Log-Datei des Kollektors alle Events bezüglich Erreichbarkeit, Inhalt, Scheduling zum jeweiligen Zeitpunkt hinzu. Somit kann zu einem späteren Zeitpunkt jedes Event und der möglicherweise fehlerhafte ROA-Datensatz identifiziert und als Messartefakt deklariert werden.

7.2.3 RTR-Sync

Analog zum Web-Sync-Modul ist die Aufgabe des RTR-Sync-Moduls die periodische Speicherung des aktuellen ROA-Datensatzes eines Kollektors über die RTR-Schnittstelle. Das Modul tritt als BGP-Router bzw. RTR-Client gegenüber dem Cache Server auf. Dafür wird eine Adaption des Python Bindings der RTRlib, das eine Umsetzung des RTR-Clients in Python darstellt, verwendet.

Ähnlich zum Web-Sync kann die Aufgabe des RTR-Sync in zwei unterschiedliche Teilbereiche aufgeteilt werden: das periodische Ausführen und die adäquate Speicherung des Dumps. Auch hier wird für die periodische Ausführung der Background-Scheduler verwendet. Jedoch wird – entgegen dem regulären RTR-Prinzip der inkrementellen Aktualisierung – periodisch ein Dump des gesamten ROA-Datensatzes erzeugt. Dies war eine Entwurfsentscheidung, die aufgrund von zwei verschiedenen Faktoren getroffen wurde. Zum einen unterstützte die aktuelle Version des RPKI-Validators zum Zeitpunkt der Arbeit kein inkrementelles Update, wodurch der ROA-Datensatz des Clients in unregelmäßigen Abständen erneut aufgebaut wurde. Zum anderen wird die mögliche Redundanz zwischen zwei aufeinanderfolgenden ROA-Dumps in Kauf genommen, um einen einheitlichen Aufbau des ROA-Archivs unabhängig vom Crawler-Typ zu gewährleisten. Somit können Analysen zu einem bestimmten Zeitpunkt den absoluten ROA-Datensatz bestimmen, ohne mithilfe der inkrementellen Updates einen absoluten Zustand zu reproduzieren.

```

1 def rtrclient(arg_collector, archive_path, date, all_mode):
2     """ Start RTR-Manager for selected collector and start BackgroundScheduler for
        periodic RTR dumps """
3
4     global mgr, interval, job, hostname, reach_file, file_name
5
6     ## Parse config file for RPKI Web Cache Server informations and build up file
        infrastructure
7     project, collector, hostname, port, interval = parse('rtr', arg_collector)[0].
        split(',')
8     col_path = archive_path + project + '/Collectors/' + collector + '/'
9     file_name = col_path + 'export.csv'
10    if not os.path.exists(col_path): os.makedirs(col_path)
11    archive_path_an = archive_path + project + '/Analysis/'
12    if not os.path.exists(archive_path_an + 'Reach'): os.makedirs(archive_path_an +
        'Reach')
13    reach_file = archive_path_an + 'Reach/' + 'reach.' + collector + '.csv'

```

```

15     ## Start RTR-Manager and dump job if the manager is synced
16     try:
17         with RTRManager(hostname, int(port), 60, status_callback =
18                         connection_status_callback) as mgr:
19             if mgr.is_synced():
20                 job = sched.add_job(create_dmp_file, 'interval', seconds=int(interval), \
21                                     name=collector, start_date=date, args=[col_path, date
22                                         , all_mode])
23             sched.add_listener(job_listener, events.EVENT_JOB_ERROR)
24             signal.pause()
25     except Exception as e:
26         logging.warning('Could not reach the server: ' + hostname + '\n\t' + type(e).
27                         __name__)
28     os._exit(-1)

```

Quellcode 7.4: Meta-Protokollierung der Erreichbarkeit des RTR-Sync

Analog zum Web-Sync-Modul werden anfänglich alle Informationen des Kollektors der Konfiguration entnommen und die File-Struktur für das Archiv und die beiden Delay- und Reachability-Analysen erstellt. Anschließend wird eine Instanz des RTR-Clients erzeugt, die mithilfe des Python Bindings einen ROA-Datenaustausch gemäß des RTR-Protokolls umsetzt. Um identische Analyse-Daten für den RTR-Crawler bezüglich Erreichbarkeit zu erzielen, wird ein Callback-Mechanismus des Bindings verwendet. Dieser wird bei Ausnahmen verwendet und protokolliert eine Nicht-Erreichbarkeit und eine Nicht-Auflösbarkeit der Kollektor-Adresse samt Zeitpunkt in der Reachability-Analyse-Datei.

```

1  ## Client could not resolve the server address
2  if(rtr_socket.state == rtr_socket.state.ERROR_TRANSPORT):
3      logging.warning('Transport Error: Address could not be resolved\n')
4      fill_reach(current_utc)
5      with open(reach_file, 'a') as target: target.write(current_utc + ',-2\n')
6      os._exit(-1)

8  ## The server is not reachable or produces fatal protocol error(s)
9  elif(rtr_socket.state == rtr_socket.state.ERROR_FATAL):
10     logging.warning('Fatal Error: Could not reach the server or protocol err: ' +
11                      hostname + '\n')
11     fill_reach(current_utc)
12     with open(reach_file, 'a') as target: target.write(current_utc + ',0\n')
13     os._exit(-1)

15 ## The server have no data available
16 elif(rtr_socket.state == rtr_socket.state.ERROR_NO_DATA_AVAILABLE):
17     logging.warning(rtr_socket.state)
18 os._exit(-1)

```

Quellcode 7.5: Ausführung des RTR-Clients des Python-Bindings der RTRlib

Analog zum Web-Sync wird in periodischen Abständen eine Methode aufgerufen, die einen ROA-Dump des aktuellen Datensatzes erzeugt. Diese Methode erzeugt zunächst ein Abbild des aktuellen ROA-Datensatzes (Präfix-Tabelle des RTR-Clients) und prüft, ob Teile oder der gesamte ROA-Datensatz inhaltslos sind. Falls dies zutrifft, wird die Erzeugung des ROA-Dumps unterbrochen und bei der nächsten periodischen Ausführung erneut durchgeführt. Bei einem vollständigen Datensatz wird gemäß der Spezifikation ein ROA-Dump erzeugt und entsprechend archiviert.

```

1  ## Get all records of the prefix table and count IPv4 and IPv6 records
2  pfx_tbl = []

```

```

3  ipv4_count, ipv6_count = (0 for i in range(2))
4  for recv4 in mgr.ipv4_records():
5      ipv4_count += 1
6      pfx_tbl.append(str(recv4.asn) + ',' + str(recv4.prefix) + '/' + str(recv4.
7          min_len) + ',' + str(recv4.max_len))
7  for recv6 in mgr.ipv6_records():
8      ipv6_count += 1
9      pfx_tbl.append(str(recv6.asn) + ',' + str(recv6.prefix) + '/' + str(recv6.
10         min_len) + ',' + str(recv6.max_len))

11 ## Check for consistency of the prefix table
12 logging.info('Ratio of the prefix tables (IPv4 - IPv6): ' + str(ipv4_count) + ' -
13     ' + str(ipv6_count))
14 if not pfx_tbl:
15     logging.warning('Prefix table is empty')
16     date_utc = datetime.utcnow().replace(second=0).strftime('%Y%m%d.%H%M')
17     fill_reach(date_utc)
18     with open(reach_file, 'a') as target: target.write(date_utc + ',,-1\n')
19     raise Exception
20 if not (ipv4_count and ipv6_count):
21     logging.warning('Parts of the prefix table are empty')
22     date_utc = datetime.utcnow().replace(second=0).strftime('%Y%m%d.%H%M')
23     fill_reach(date_utc)
24     with open(reach_file, 'a') as target: target.write(date_utc + ',,-1\n')
25     raise Exception

```

Quellcode 7.6: Fetching der ROA-Daten aus der Präfix-Tabelle und Konsistenzüberprüfung

Anschließend wird je nach Scheduling-Parameter das entsprechende (Re-)Scheduling ausgeführt, um bei einer möglichen Asynchronität den Zeitpunkt der nächsten Ausführung zu beeinflussen (nähtere Erläuterungen in 7.2.4). Erneut wird der Logging-Mechanismus verwendet, um Anomalien und RPKI-Inkonsistenzen strikt trennen zu können. Eine Ausnahme für den Logging-Mechanismus tritt ein, wenn während des Neuaufbaus des ROA-Datensatzes, aufgrund der Nicht-Unterstützung des inkrementellen Updates, ein ROA-Dump erzeugt wird. Da dieses Event trotz fehlender Einträge als reguläre Speicherung angesehen wird, muss diese Anomalie identifiziert, als Messartefakt deklariert und anschließend normiert werden.

Wegen der Vielzahl an Kollektoren, wurde eine Abstraktion in Form eines All-Sync-Moduls erstellt, um alle Kollektoren – je nach Typ – mit einer entsprechenden Crawler-Instanz zeitgleich zu starten. Dieses Modul ermöglicht eine einfachere Handhabung der verschiedenen Crawler-Instanzen und initiiert ein gemeinsames Scheduling-Verhalten.

7.2.4 (Re-)Scheduling

Trotz zeitgleichem Ausführungsstart verschiedener Crawler-Instanzen kann es während der Durchführung zu vereinzelten Ausnahmen und Asynchronität kommen. Da für die Inkonsistenz-Analyse der Cache Server absolute Synchronität essentiell ist, muss gewährleistet sein, dass alle Instanzen zeitgleich und periodisch Dumps erzeugen und archivieren. Um dies sicherzustellen, wurden drei verschiedene (Re-)Scheduling-Methoden erzeugt:

- Next-Own-Timestamp (NOTS),
- Next-Common-Timestamp (NCTS),
- Restart-Scheduling.

Die Methoden können je nach Parameter und Ziel einzeln oder in Kombination verwendet werden.

Next-Own-Timestamp (NOTS)

Bei der Next-Own-Timestamp-Methode handelt es sich um einen Vorgang zum (Re-) Scheduling einer einzelner Crawler-Instanz, die ausschließlich auf der Startzeit und dem Intervall dieser Instanz beruht. Im Falle eines Fehlers wird der nächste Zeitpunkt für eine Dump-Erzeugung so berechnet, dass der Startzeitpunkt durch das in der Konfiguration angegebene Intervall bis zur nächsten Iteration erweitert wird. Das Scheduling beruht ausschließlich auf der eigenen Crawler-Instanz und ist unabhängig von anderen Instanzen. Der Vorgang wird in Abbildung 7.3 anhand eines Beispiels dargestellt.

Status	Timestamp	CC01(Web)	CC02(Web)	CC03(Web)	CC04(Web)	CC07(Web)
Start time		00:00	00:00	00:00	00:00	00:00
Sync	00:00	D	D	D	D	D
Sync	00:03	D	D	D	D	D
Out of Sync						
NOTS(00:00,3m) = 00:00 → 00:09	00:06	D	Error	D	D	D
Sync	00:09	D	D	D	D	D

D = Dump

Abbildung 7.3: Beispiel für den Ablauf des Next-Own-Timestamp-Scheduling

Im dargestellten Fall wird die Crawler-Instanz des Kollektoren “CC02“ im Zeitpunkt 00.06 durch eine Ausnahme oder einen Fehler unterbrochen. Daraufhin wird der Startzeitpunkt 00.00 durch das Intervall mehrfach erweitert, so dass der nächste Dump-Zeitpunkt mit 00.09 feststeht. Im Anschluss läuft die Crawler-Instanz des Kollektoren wieder ordnungsgemäß.

Diese (Re-)Scheduling-Methode findet ausschließlich Anwendung bei autarken Crawler-Instanzen mit unterschiedlichen Intervallen und Startzeitpunkten, so dass diese in den hier gezeigten Analysen keine Anwendung findet.

Next-Common-Timestamp (NCTS)

Bei der Next-Common-Timestamp-Methode handelt es sich, um eine (Re-) Scheduling-Methode, die auf allen aktiven Crawler-Instanzen beruht. Um eine Synchronität zwischen allen Crawler-Instanzen zu erzeugen, wird der meist verwendete Zeitpunkt für den nächsten Dump (Next-Common-Timestamp) und die Anzahl der beteiligten Instanzen berechnet. Weicht der Zeitpunkt des nächsten Dumps einer Instanz von diesem NCTS ab, wird dieser Zeitpunkt auf den NCTS korrigiert. Der Vorgang wird in Abbildung 7.4 verdeutlicht.

Status	Timestamp	CC01(Web)	CC02(Web)	CC03(Web)	CC04(Web)	CC07(Web)
Start time		00:00	00:00	00:00	00:00	00:00
Sync	00:00	D	D	D	D	D
Out of Sync NCTS = (00:06,3) Majority of 3 Collectors	00:03 00:04 00:05	D	(Delay)	D	D	(Delay)
Sync	00:06	D	D	D	D	D

D = Dump

Abbildung 7.4: Beispiel für den Vorgang des Next-Common-Timestamp-Scheduling

Im dargestellten Fall kommt es bei den Crawler-Instanzen der Kollektoren "CC02" und "CC07" wegen Verzögerung zu einer Asynchronität, wobei "CC02" erst zum Zeitpunkt 00.04 und "CC07" erst zum Zeitpunkt 00.05 einen Dump erzeugt. Da somit kein konsistenter ROA-Datensatz aller Kollektoren zum Zeitpunkt 00.03 vorliegt, wäre eine Inkonsistenz-Analyse dieser Kollektoren für diesen Zeitpunkt nicht möglich.

Während der Dump-Erzeugung berechnen alle Crawler-Instanzen ihren nächsten Dump-Zeitpunkt, für "CC02": $00.04 + 00.03 = 00.07$ und für "CC07": $00.05 + 00.03 = 00.08$. Da der NCTS jedoch durch eine Mehrheit von drei Instanzen (CC01, CC03, CC04) bei 00.06 liegt, wird diese Asynchronität bei den beiden fehlerhaften Instanzen festgestellt und deren nächster Dump-Zeitpunkt ebenfalls auf den NCTS 00.06 gesetzt. Danach laufen alle Crawler-Instanz wieder ordnungsgemäß und synchron.

Im Falle einer Anomalie oder eines schwerwiegenden Fehlers, der eine Crawler-Instanz vollständig beendet, würde diese Methode keinen Erfolg erzielen. Dieser Fall wird durch das Restart-Scheduling abgefangen.

Restart-Scheduling

Die Restart-Scheduling-Methode ist ein Fallback-Mechanismus, der eine Crawler-Instanz im Falle eines schwerwiegenden Fehlers vollständig neu startet.

Status	Timestamp	CC01(Web)	CC02(Web)	CC03(Web)	CC04(Web)	CC07(Web)
Start time		00:00	00:00	00:00	00:00	00:00
Sync	00:00	D	D	D	D	D
Out of Sync NCTS = (00:06, 3) Majority of 3 Collectors	00:03 00:04 00:05	D	(Delay)	D	D	(Delay)
Out of Sync	00:06	D	D	D	D	
Restart CC07(Web)	00:07					(Restart)
Sync	00:09	D	D	D	D	D

D = Dump

Abbildung 7.5: Beispiel für Restart Scheduling

Falls eine Crawler-Instanz für zwei aufeinanderfolgende Dump-Zeitpunkte asynchron zum NCTS war, findet die Restart-Methode ebenfalls Anwendung. Ein einmaliges Ausfallen der Dump-Erzeugung kann durch die NCTS-Methode korrigiert werden. Bei einem weiteren Ausfall wird die Instanz vollständig neu gestartet. Dafür wird der Prozess beendet, die Startzeit in der Konfiguration auf den Next Common Timestamp gesetzt und der Prozess gestartet.

Wie in Abbildung 7.5 verdeutlicht, würde die Crawler-Instanz von "CC07" bei einem erneuten Ausfall trotz einmaligem NCTS-Rescheduling durch das Restart-Scheduling beendet und zum NCTS neu gestartet werden. Da die Kombination der NCTS- und Restart-Methode die schnellstmögliche Korrektur der Asynchronität hin zur Synchronität gewährleistet, wird diese in den Analysen dieser Arbeit verwendet.

Analog zu den Crawler-Typen wurde die Scheduling-Methode um einen Logging-Mechanismus erweitert, der die Gesamtheit aller Crawler-Instanzen und deren Synchronitätszustände protokolliert. Zu Monitoring-Zwecken wurde der Scheduling-Methode ein Feature zum periodischen Anzeigen der aktuellen Synchronitätszustände aller Crawler-Instanzen hinzugefügt.

Neben dem (Re-)Scheduling muss das Zeit-Management beachtet werden, da die ROA-Datensätze in verschiedenen Analysen mit externen Daten mit Zeitangaben zusammengeführt werden. Damit dies ohne Zeitverschiebungen erfolgt, wird ein konsistentes Zeit-Management verwendet.

7.2.5 Zeit-Management

Das Zeit-Management umfasst verschiedene Teile der Messinfrastruktur: Scheduling, ROA-Crawler und ROA-Dump-Archiv.

Das Scheduling verwendet grundsätzlich aufgrund von Systemaufrufen und Kompatibilität die lokale Zeit des Systems. Somit werden die einzelnen Crawler-Instanzen stets nach der aktuellen Lokalzeit ausgeführt und geplant. Die Crawler-Instanzen per se verwenden aufgrund dieser Scheduling-Aufrufe und der Dump-Erzeugung einen hybriden Modus, in dem sie – je nach Anwendung – zwischen UTC und lokaler Systemzeit wechseln. Für die Dump-Erzeugung wird die UTC-Version der aktuellen Systemzeit berechnet und als Bezeichnung verwendet. Daher wird das ROA-Dump-Archiv gemäß der Spezifikation stets in UTC gehalten, um Komplikationen mit externen Daten, wie beispielsweise BGP-Dumps, zu vermeiden.

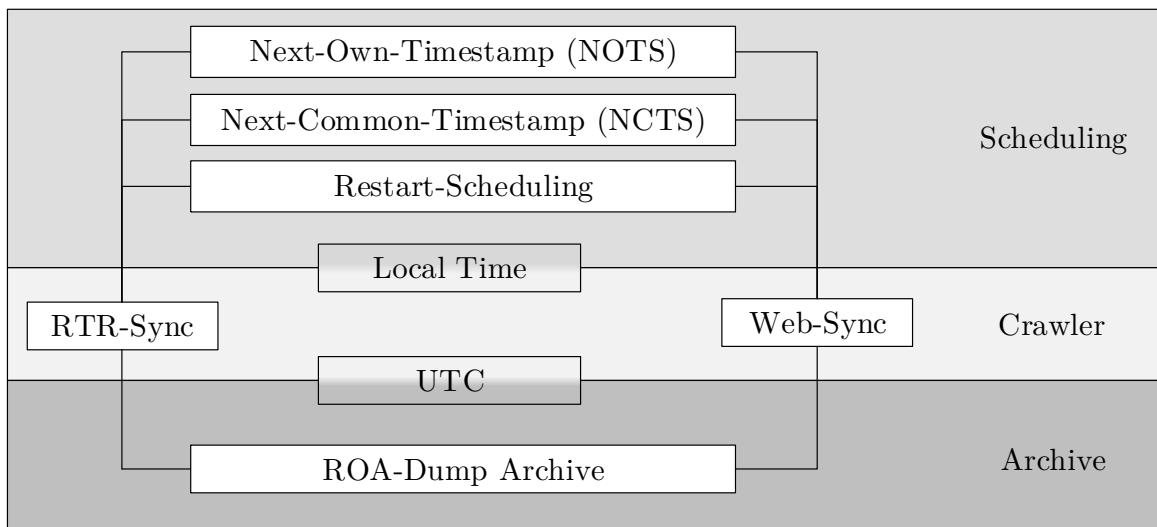


Abbildung 7.6: Zeit-Management

7.3 Entwurf des ROA-Dump-Archivs

Das ROA-Dump-Archiv ist eine Sammlung von ASCII-basierten ROA-Dumps. Wie die Abbildung 7.7 zeigt, ist das Archiv zunächst nach einzelnen Projekten gruppiert, so dass es möglich ist, verschiedene Projekt-Instanzen zu führen. Unterschiedliche Crawler könnten als verteiltes System gemeinsam das ROA-Dump-Archiv je nach Projekt befüllen.

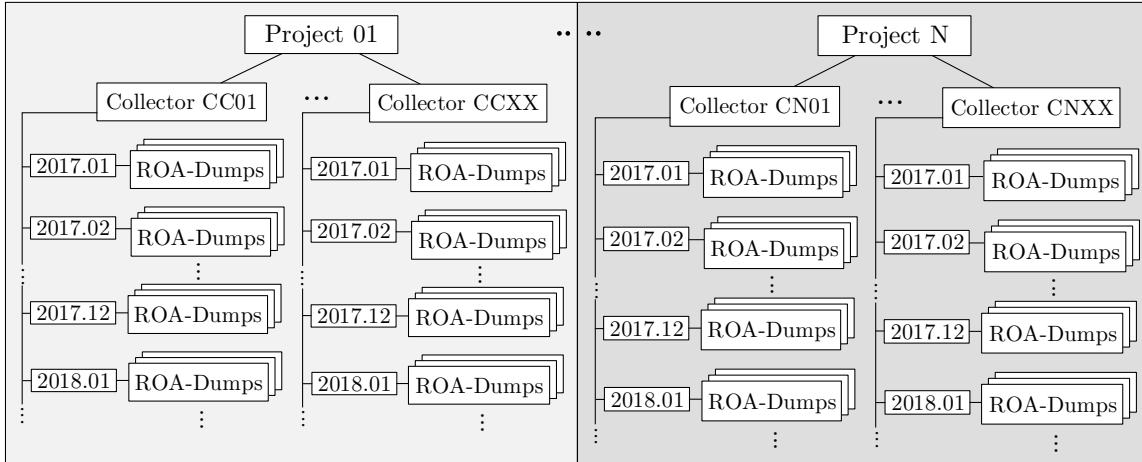


Abbildung 7.7: Entwurf des ROA-Dump-Archivs

Jedes Projekt ist aufgeteilt in die einzelnen Kollektoren, die in der jeweiligen Konfiguration spezifiziert sind. Die Bezeichnung der Kollektoren entspricht der Konfiguration und sollte eindeutig vergeben werden. Gemäß der Konfiguration werden die Bezeichnungen dem Cache-Server-Typ angepasst.

Die ROA-Dumps jedes Kollektors werden in die einzelnen Jahresmonate in der Form “%Y%M“ gruppiert. Diese Unterteilung hat den Vorteil, dass jeder Archiv-Ordner eine maximale File-Anzahl von $(60\text{min}/\text{Intervall}) * 24\text{h} * 31\text{d}$ besitzt und so im Hinblick auf die Langlebigkeit des Archivs die Grenzen gängiger File-Systeme nicht überschreiten kann. Für das in dieser Arbeit spezifizierte Intervall ergibt sich eine maximale Anzahl von 14880 Dateien.

Die Spezifikation der ROA-Dumps selbst sieht vor, dass die File-Bezeichnung direkt aus der Archivierungszeit folgt und die Form “vpr.%Y%m%d.%H%M.csv.gz“ hat. Somit würde ein ROA-Dump, das zum Zeitpunkt des 03.08.2017 um 18:24 Uhr archiviert wurde, die File-Bezeichnung “vpr.20170803.1824.csv.gz“ tragen.

Wie bereits in Abschnitt 7.2.5 erwähnt, muss die Uhrzeit stets als UTC-Zeit angegeben sein. Um Speicherplatz zu sparen, werden alle ROA-Dumps mithilfe der gzip-Kompression komprimiert. Im Gegensatz zu anderen Kompressionsarten bietet die gzip-Kompression für die vorliegenden Daten einen Tradeoff zwischen Speichereinsparung und (De-)Kompressionszeit. Dies ist notwendig, da die ROA-Dumps für einige Analysen sehr häufig und in großer Stückzahl dekomprimiert und gelesen werden müssen, so dass dieser Faktor direkt Wirkung auf die Analysezeit hat.

KAPITEL 8

Entwurf und Implementierung einer Bibliothek zur Analyse der Auswirkungen auf BGP

Für die Analyse der Auswirkungen auf BGP, müssen die BGP-Announcements untersucht und mit den passenden ROA-Datensätzen validiert werden. Dazu ist ein Framework notwendig, das zu einem bestimmten Zeitpunkt oder Zeitraum BGP-Daten (speziell BGP-Announcements) verarbeitet. Für diese Aufgabe wurde “BGPStream“ von CAIDA (UC San Diego) auserwählt, da es aktuell das meist verwendete BGP-Analyse-Framework ist.

BGPStream ermöglicht den Zugang zu Echtzeit und historischen BGP-Daten, die für die entsprechenden Analysen benötigt werden. Durch den Entwurf und die Implementierung einer geeigneten Bibliothek namens ROAFetchlib werden diese BGP-Daten nativ während der Verarbeitung validiert. Dafür werden die ROA-Datensätze des Archivs mithilfe eines ROA-Brokers öffentlich zugänglich gemacht. Da so Validierungsergebnisse für BGP-Announcements mithilfe des Archivs für eine größere Zeitspanne analysiert und ausgewertet werden können, können die Auswirkungen von RPKI-Inkonsistenzen zwischen den Cache-Servern auf BGP analysiert werden.

Nachfolgend wird BGPStream näher erläutert, um darauf aufbauend den Entwurf und die Implementierung des ROA-Brokers und der “ROAFetchlib“ Bibliothek zu erklären.

8.1 BGPStream

BGPStream [11] ist ein Open-Source Software-Framework für die Analyse von historischen und Echtzeit-basierten BGP-Daten. Obwohl BGP eine entscheidende Komponente der Internet-Infrastruktur und -Funktionalität ist, gab es bis dato keine effiziente und vergleichsweise einfache Möglichkeit, große Mengen an verteilten historischen und Echtzeit-basierten BGP-Daten zu verarbeiten und analysieren. BGPStream füllt diese Lücke und ermöglicht somit eine effiziente Analyse von BGP-Announcements von real gemessenen BGP-Daten.

BGPStream besteht, wie in Abbildung 8.1 dargestellt, aus verschiedenen Komponenten:

- Meta-Daten-Provider,
- Daten-Provider,
- LibBGPStream,
- Analyse- und Ausgabe-Programme.

In dieser Arbeit sind davon ausschließlich die öffentlichen Provider (BGPStream-Broker und die verschiedenen Daten-Provider), die Bibliothek und der BGPReader von Interesse.

Um für einen bestimmten Zeitraum die entsprechenden BGP-Daten zu erhalten, kann das Shell-Tool BGPReader mit den entsprechenden Parametern für die Daten-Provider, Zeitraum und BGPStream-Broker gestartet werden. Daraufhin wird eine Anfrage an den von CAIDA betriebenen BGPStream-Broker gestellt, um die Metadaten für die durch die Parameter spezifizierten BGP-Daten zu erhalten. Der BGPStream-Broker antwortet mit einer JSON-Datei, die einen Stream an Informationen über den Speicherort der aufeinanderfolgenden BGP-Daten und deren Messzeitraum beinhaltet. Diese BGP-Daten sind vom BGP-Kollektor aufgenommene reale Messdaten. Dafür gibt es unterschiedliche BGP-Knotenpunkte (Vantage Points), die in regelmäßigen Abständen ihre gesamte Routing-Table (RIB) oder Updates (BGP-Announcements) für den ausgehenden Verkehr an diese BGP-Kollektoren senden.

Mithilfe der Metadaten des BGPStream-Broker kann die Bibliothek "LibBGPStream" Zugang zu diesen BGP-Daten erhalten. Anschließend werden diese chronologisch sortiert, extrahiert und in einzelne Elemente aufgeteilt. Dies ist notwendig, sofern ein BGP-Eintrag mehrere atomare Elemente enthält. Jedes atomare BGP-Element wird letztendlich in einem definierten Schema ausgegeben [11].

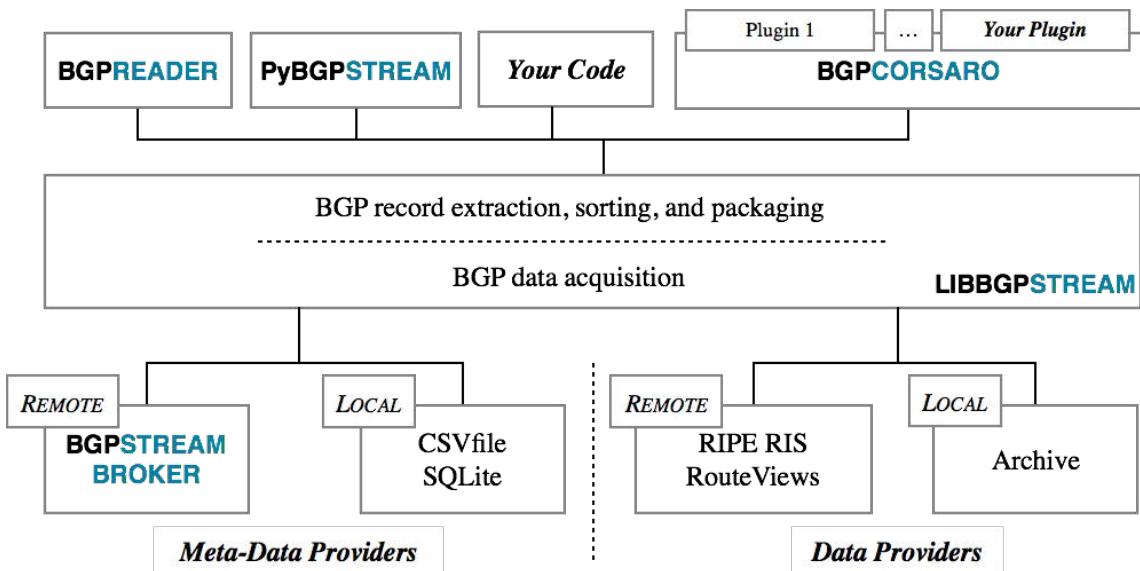


Abbildung 8.1: BGPStream – Entwurf

8.2 ROA-Broker

Ähnlich zum BGPSStream-Broker ermöglicht der ROA-Broker den Erhalt von Metadaten zu den im ROA-Dump-Archiv existierenden Dumps. So kann der ROA-Broker analog als Metadaten-Provider und das ROA-Dump-Archiv als Daten-Provider angesehen werden. Der ROA-Broker übernimmt die Aufgabe, zu einem bestimmten Zeitraum und einigen weiteren Parametern die passenden ROA-Datensätze zu ermitteln und der Bibliothek zur Verfügung zu stellen.

Der ROA-Broker bietet – im Gegensatz zu einem lokalen Meta-Daten-Provider – eine weitere Abstraktion, so dass die ROAFetchlib und die entsprechenden Analysen auf einem unabhängigen Gerät mit Internetverbindung ausgeführt werden können. Dieses muss keine Verbindung zur eigentlichen Messinfrastruktur haben.

Nachfolgend wird das Design näher erläutert.

8.2.1 Design

Das Design des ROA-Brokers und die Beziehungen zu anderen Entitäten der erweiterten Messinfrastruktur, wie zum Beispiel dem ROA-Dump-Archiv, der ROAFetchlib und weiteren Metadaten-Providern, ist für die Funktionalität und Effizienz der Bibliothek von äußerster Wichtigkeit. Um diesen Anforderungen so effizient wie möglich gerecht zu werden, wurden zwei verschiedene Ansätze ausgearbeitet und schließlich der effizientere ausgewählt.

Nachfolgend werden beide Ansätze vorgestellt, deren Vor- und Nachteile aufgezeigt und die getroffene Entscheidung deutlich gemacht.

Sequenzielles Broker-Design

Das sequenzielle Broker-Design beruht auf der Beziehung der Metadaten-Provider, dem BGPSStream-Broker und dem ROA-Broker zueinander. Der ROA-Broker ist – wie die Abbildung 8.2 zeigt – vor den BGPSStream-Broker „geschaltet“, so dass dieser zwischen dem BGPSStream-Framework und dessen BGPSStream-Broker liegt.

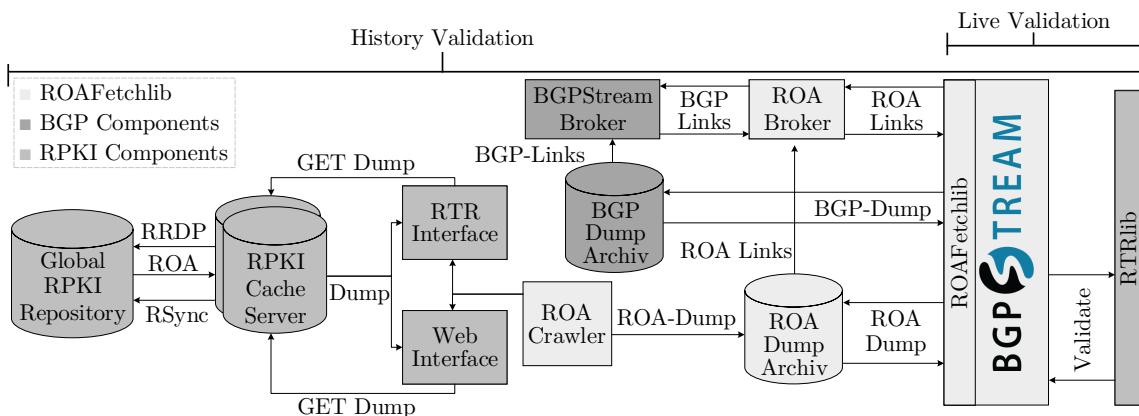


Abbildung 8.2: Erweiterte Messinfrastruktur mit sequenziellem Broker-Design

Um Metadaten zu erhalten, schickt die BGPStream-Instanz die Anfrage für die BGP-Daten zunächst an den ROA-Broker. Dieser leitet diese Anfrage an den BGP-Broker weiter, um die entsprechende JSON-Datei samt BGP-Metadaten zu erhalten. Um der BGPStream-Instanz neben den herkömmlichen BGP-Metadaten nun ebenfalls die Metadaten zu den ROA-Datensätzen zur Verfügung zu stellen, wird die vom ROA-Broker empfangene JSON-Datei mit den jeweiligen Informationen über die ROA-Dumps erweitert. Die Informationen beinhalten für jeden in der JSON-Datei vorkommenden BGP-Dump den chronologisch passenden ROA-Dump-Link zum ROA-Dump im Archiv. Die BGPStream-Instanz erhält darauf die erweiterte JSON-Datei und kann nachfolgend die BGP- und ROA-Daten vom dem jeweiligen Archiv laden und verarbeiten.

Aus diesem sequenziellen Design ergeben sich folgende Vor- und Nachteile:

Vorteile:

- Sofortige Broker-seitige Zusammenführung der BGP- und ROA-Dumps
- Einheitliches Time-Slicing Schema (abhängig vom BGPStream-Broker)

Nachteile:

- Starke gegenseitige Abhängigkeit der Metadaten-Provider (Broker)
- ROA-Broker bildet Performance-Bottleneck
- Weitere Metadaten-Provider müssten stets vorgeschaltet werden und könnten wiederum einen Performance-Bottleneck bilden
- Semantisch unterschiedliche Daten teilen sich eine Datenstruktur
- Erweiterung des BGPStream-Framework statt eigenständiger Bibliothek

Paralleles Broker-Design

Entgegen dem sequenziellen Design werden beim parallelen Broker-Design die unterschiedlichen Metadaten-Provider vollkommen getrennt und somit autark betrieben und angefragt. Wie in Abbildung 8.3 dargestellt, existieren beide Broker unabhängig voneinander und können zeitgleich angefragt werden.

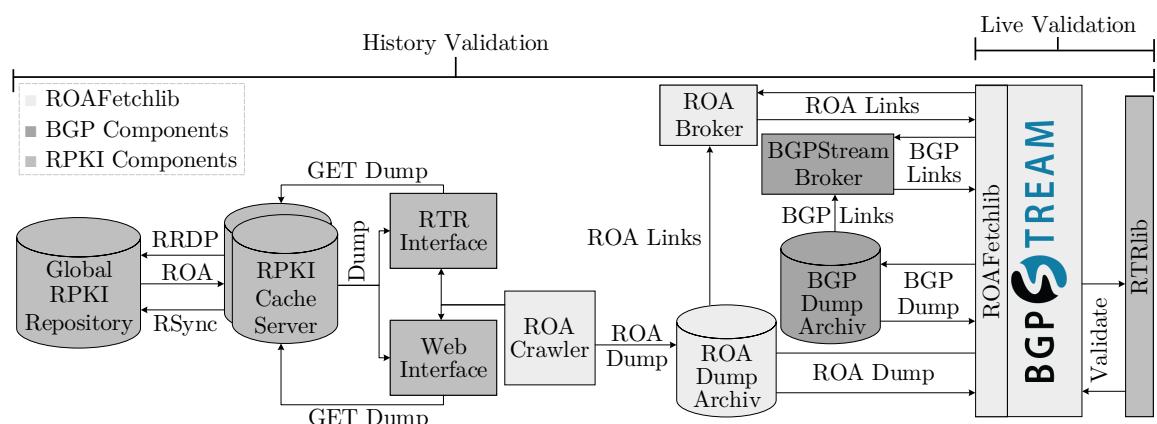


Abbildung 8.3: Erweiterte Messinfrastruktur mit parallelem Broker-Design

Um Metadaten für einen bestimmten Zeitraum zu erhalten, schickt die BGPStream-Instanz unverändert die Anfrage für die BGP-Daten an den entsprechenden BGP-Broker. Für den identischen Zeitraum sendet die ROAFetchlib, die von der BGP-Instanz gestartet wird, eine Anfrage an den ROA-Broker. Der ROA-Broker antwortet daraufhin mit einer eigenständigen JSON-Datei, die die passenden URLs zu den ROA-Datensätzen samt Meta-Informationen, beinhaltet. Analog zum sequenziellem Design können die jeweiligen URLs genutzt werden, um die ROA-Daten vom ROA-Dump-Archiv zu laden. Die jeweils von den Bibliotheken erhaltene JSON-Datei und die dahinter stehenden BGP- und ROA-Datensätze können nun unabhängig voneinander von den beiden Bibliotheken eingelesen und verarbeitet werden.

Somit ergeben sich die folgenden Vor- und Nachteile für das parallele Broker-Design:

Vorteile:

- Die Metadaten-Provider (Broker) sind unabhängig voneinander und laufen autark.
- Einzelne Metadaten werden voneinander getrennt verschickt und verarbeitet.
- Die Performance der einzelnen Broker sind unabhängig → kein Bottleneck.
- Auswahl von Teilmengen der Metadaten-Provider ist möglich, ohne redundante Daten zu versenden.
- Ermöglicht eine Modularisierung der ROA-Validierung als verteilte Bibliothek (*Shared Library*).

Nachteile:

- Bibliothek muss Zusammenführung der ROA- und BGP-Datensätze unterstützen (Client-basiert).
- Broker-abhängiges Time-Slicing Schema

Da die Vorteile die Nachteile überwiegen, wurde für das Broker-Design der parallele Ansatz gewählt. Das parallele Broker-Design bietet insgesamt mehr Flexibilität hinsichtlich der Konstellation und Auswahl der Metadaten-Provider, ermöglicht eine Modularisierung der einzelnen Aufgabenbereiche und verhindert mögliche Bottlenecks in der Messinfrastruktur.

Aufgrund der Bedeutung des Designs für beide Bibliotheken, wurde diese Entwurfsentscheidung mit den Entwicklern bei CAIDA besprochen.

8.2.2 Funktionsweise und Implementierung

Die Funktionsweise des ROA-Brokers bietet die Möglichkeit, die Metadaten zu ROA-Datensätzen unabhängig vom Endgerät über einen HTTP-Aufruf anzufragen. Dazu hat der ROA-Broker direkten Zugang zum ROA-Dump-Archiv und ermöglicht die Verteilung der entsprechenden ROA-Datensätzen über einen File-Server. Der ROA-Broker besitzt dafür eine URI, wodurch für die Metadaten entscheidende Parameter übergeben werden können.

Die URI des ROA-Brokers hat folgende Syntax:

```
http://roa-broker.realmv6.org/broker?  
project = <(Projekte[,])*>&  
collector = <(Kollektoren[,])*>&  
interval = <(utcstart1-utcend1,utcstartn-utcendn)*>
```

Ein Beispielaufruf für die Kollektoren „CC01“ und „CC02“ aus dem Projekt „FU-Berlin“ für das Zeitintervall 26.07.2017,01:30 Uhr bis 01:46 Uhr, würde somit wie folgt aussehen:

`http://roa-broker.realmv6.org/broker?project=FU-Berlin,FU-Berlin&collector=CC01, CC02&interval=1501032600-1501033600`

Die Parameter für die Projekte, Kollektoren und Zeit-Intervalle können so übergeben werden. Projekte und Kollektoren werden dabei in einer Komma-getrennten Liste übergeben, wobei eine Eins-zu-eins-Beziehung zwischen den Projekten und den Kollektoren herrscht. Die Zeitintervalle werden im UTC-Unix-Zeitformat zugeleitet, wobei jedes Intervall mit einem Bindestrich-getrennten Start- und Endzeitpunkt definiert wird, die ebenfalls in der Gesamtheit als Komma-getrennte Liste übergeben werden.

Nachdem die URI mit den entsprechenden Parametern aufgerufen wurde, werden diese an eine Broker-Instanz übergeben. Diese liest zunächst die Konfigurationsdatei des ROA-Brokers ein, um den Pfad des ROA-Dump-Archivs, die Adresse des Datei-Servers und den Pfad des für den Aufbau des Archivs zuständigen ROA-Crawlers zu bekommen. Nachdem diese Konstanten eingelesen wurden, wird ein Parsing und eine Validierung durchgeführt zur Integritätsprüfung durchgeführt.

Die Prüfung wird durch reguläre Ausdrücke vorgenommen:

- Projekte: $[a - zA - Z() - _]_+ (, [a - zA - Z() - _]_+)^*$
- Kollektoren: $CC[0 - 9]\{2\}[0 - 9]?((RTR))?(, CC[0 - 9]\{2\}[0 - 9]?((RTR))?)^*$
- Intervalle: $[0 - 9]\{10\} - (0|[0 - 9]\{10\})(, [0 - 9]\{10\} - (0|[0 - 9]\{10\}))$

Im Falle eines Verstoßes gegen die regulären Ausdrücke wird eine entsprechende Fehlermeldung als Antwort auf die Anfrage versendet. Nach dieser Prüfung folgt die semantische Validierung der einzelnen Parameter, so dass die Anzahl an Projekten und Kollektoren identisch sein muss. Des Weiteren wird geprüft, ob die übergebenen Projekte und Kollektoren im ROA-Dump-Archiv verfügbar sind. Falls dies nicht der Fall ist, wird erneut mit einer Fehlermeldung geantwortet.

Bei der Validierung der Intervalle, wird zunächst geprüft, ob jeder Zeitpunkt jedes Intervalls ein valider UTC-Unix-Zeitstempel ist. Anschließend wird getestet, ob die einzelnen Intervalle chronologisch korrekt sind, so dass jeder Startzeitpunkt kleiner als der jeweilige Endzeitpunkt und nicht größer als der aktuelle UTC-Zeitpunkt ist. Falls die Validierung fehlschlagen sollte, erfolgt erneut eine Fehlermeldung in Form einer Antwort an den Client.

Anschließend folgt das Befüllen eines geordneten „Dictionary“ mithilfe der bereits bekannten übergebenen Parameter und des letzten Zeitstempels aller Zeit-Intervalle, die letztendlich die Meta-Informationen der JSON-Datei formen. Die Projekte und Kollektoren werden dabei lexikografisch sortiert, um eine eindeutige Zuordnung von ROA-Dump-URL und entsprechendem Kollektor garantieren zu können. Um die passenden ROA-Dumps ausfindig zu machen, wird für jedes Projekt bzw. jeden Kollektor und jedes Zeit-Intervall des Kollektors zunächst das Intervall des Kollektors aus der Konfiguration entnommen und alle möglichen ROA-Dumps unabhängig des Intervalls bestimmt. Mithilfe des jeweiligen Zeit-Intervalls und des Kollektor-Messintervalls können nun alle ROA-Dumps des Kollektors nach diesen Zeitangaben gefiltert werden, so dass ausschließlich ROA-Dumps identifiziert werden, die in dieses Intervall passen.

Die Zeit-Intervalle können dabei in drei verschiedenen Formen auftreten:

- ein geschlossenes Intervall mit zwei expliziten Zeitstempeln,
- ein halb-offenes Intervall mit einem expliziten Zeitstempel,
- ein offenes Intervall mit keinem expliziten Zeitstempel.

Ein geschlossenes Intervall kann die Verbindung zweier Zeitstempel, wie beispielsweise 1491004800 – 1491005000, sein. Ein halb-offenes Intervall kann durch den Wert 0 und einen Zeitstempel definiert werden, wodurch alle ROA-Dumps bis bzw. ab einem bestimmten Zeitstempel ausgewählt werden.

Ein Beispiel wären die beiden folgenden Zeit-Intervalle:

- 1491004800–0, womit alle ROA-Dumps verwendet werden, die neuer als der Zeitpunkt 01.04.2017, 00:00 Uhr sind
- 0 – 1491004800, womit alle ROA-Dumps verwendet werden, die älter als der Zeitpunkt 01.04.2017, 00:00 Uhr sind

Ein offenes Intervall kann durch die Kombination beider Fälle 0 – 0 definiert werden, wodurch alle verfügbaren ROA-Dumps des Kollektors verwendet werden.

Nicht-fortlaufende Zeitfenster können durch die Definition mehrerer Zeit-Intervalle angegeben werden, so dass zwischen den einzelnen Zeit-Intervallen Lücken entstehen. Für diese Lücken werden entsprechend keine ROA-Dumps verwendet. Im Kontrast dazu können zwischen den Zeit-Intervallen auch Schnittmengen existieren, die Duplikate von ROA-Dumps zur Folge hätten. Um dies zu verhindern, werden entsprechende Duplikate aus den ROA-Dump-Mengen entfernt, so dass jeder ROA-Dump nur einmal bei einer Anfrage vorkommen kann. Um nun alle benötigten ROA-Dumps im Archiv ausfindig zu machen, wird eine Filtering-Methode verwendet. Diese Methode prüft, welche ROA-Dumps der Gesamtmenge in das vorgegebene Intervall passen. Es werden nur ROA-Dumps verwendet, falls der Zeitstempel des Dumps größer gleich dem Start und kleiner gleich dem Ende des Intervalls entspricht, wobei der Start des Zeit-Intervalls kein direktes Matching mit dem Zeitstempel des ROA-Dumps ergeben muss. So können ebenfalls Zeitpunkte gewählt werden kann, die innerhalb des Intervalls des ROA-Dumps liegen. Um diesen Fall zu ermöglichen, wird das Messintervall des Kollektors verwendet, so dass zusätzlich das ROA-Dump ausgewählt wird, das zwischen dem letzten Messintervall vor dem Start des definierten Zeit-Intervalls und dem Start selbst liegt.

Anhand des folgenden Beispiels wird dieser Sachverhalt verdeutlicht:

Zeit-Intervall: 1491004800 – 1491005000

verfügbare ROA-Dumps: 1491004660 – 1491004840, 1491004840 – 1491005020

Für dieses Intervall werden beide ROA-Dumps gewählt, obwohl der Beginn des ersten ROA-Dumps außerhalb des eigentlichen Zeit-Intervalls liegt. Dies ist notwendig, um das Zeitfenster 1491004800 – 1491004840 mit dem semantisch korrekten ROA-Dump validieren zu können. Anschließend werden alle restlichen ROA-Dumps des Kollektors, die in das vorgegebene Zeit-Intervall passen hinzugefügt und dem entsprechenden Pfad der File-Server-URL an gehangen. Alle in die Zeit-Intervalle passenden ROA-Dumps der einzelnen Kollektoren werden anschließend nach Zeitstempel gruppiert und als Kombination aus Zeitstempel und String-Array dem *Dictionary* hinzugefügt und anschließend in ein JSON-Objekt konvertiert.

Im Falle von fehlenden ROA-Dumps einiger Kollektoren werden die entsprechenden ROA-Dump-URLs frei gehalten. Das dabei verwendete String-Array der einzelnen ROA-Dump-URLs ist dabei lexikografisch sortiert. Dies ist notwendig, da es sonst keine direkte Zuordnung von Kollektor zu fehlendem ROA-Dump gibt (dieser Fall wird näher erläutert in Abschnitt 8.3.4).

Es ergibt sich somit das folgende JSON-Format für die Antwort des ROA-Broker:

JSON Format	
Meta Informations	
Projects:	Comma-separated sorted list of all projects
Collectors:	Comma-separated sorted list of all collectors
Interval:	Comma-separated list of all time intervals
Max_End:	Last timestamp of all time intervals
ROA Data	
Timestamp #1:	Sorted list of all ROA-Dump URLs of the collectors at timestamp #1
Timestamp #2:	Sorted list of all ROA-Dump URLs of the collectors at timestamp #2
:	:
Timestamp #N:	Sorted list of all ROA-Dump URLs of the collectors at timestamp #N

Abbildung 8.4: JSON-Format des ROA-Brokers

Das JSON-Format wurde mittels einer Entwurfsentscheidung aufgrund der Performance als Austauschmedium ausgewählt. Eine Alternative war das direkte Versenden der einzelnen ROA-Dumps vom ROA-Broker an den entsprechenden Client. Dies hätte allerdings hohe Performance- und Netzwerklast-Einbüße verursacht. Des Weiteren hätte der entsprechende Client alle abgefragten ROA-Dumps zwischenspeichern müssen, was eine unnötige Belastung des Clients gewesen wäre.

Eine weitere Alternative wäre das Übertragen der Pfade zu den jeweiligen Kollektor-Ordnern des öffentlichen Archivs an den Client. Dies hätte zur Folge gehabt, dass der Client die Zuordnung zwischen ROA-Dumps und den vorgegebenen Parametern hätte vornehmen müssen. Dies hätte neben einer erneut starken Client-Last auch eine geringere Flexibilität zur Folge gehabt. Das JSON-Format des ROA-Brokers ermöglicht somit eine flexible, Netzwerk- und Performance-schonende Art dem Client die entsprechenden Meta-Daten zur Verfügung zu stellen.

Nachfolgend wird die ROAFetchlib näher erläutert, die diese Meta-Daten verwendet, um BGP-Announcements mit den passenden ROA-Dumps validieren zu können.

8.3 Bibliothek – ROAFetchlib

Die ROAFetchlib entstand aufgrund einer Entwurfsentscheidung zwischen zwei verschiedenen Ansätzen, um das Problem der nativen RPKI-Validierung von BGP-Announcements zu lösen. Der erste Ansatz war die Erweiterung des bereits bestehenden LibBGPStream-Quellcodes um die RPKI-Funktionalität. Aufgrund der Komplexität und dem Fakt, dass die Daten der BGP-Announcements nativ verarbeitet werden sollten, hätte es zu einer hohen Konvergenz und Abhängigkeit geführt. Der zweite Ansatz war die Implementierung einer *Shared Library*, die ausschließlich die native RPKI-Validierung von BGP-Announcements realisiert. Mithilfe der API der Bibliothek wäre es somit möglich, bestehende BGP-Analyse-Tools um die RPKI-Funktionalität zu erweitern.

Die Vor- und Nachteile beider Ansätze sind nachfolgend in der Gegenüberstellung zu sehen.

Erweiterungsansatz

Nachteile:

- Abhängigkeit beider Funktionalitäten
- Hohe Konvergenz der Quellcodes
- Aufwendige Wartung
- Keine semantische Modularisierung

Bibliotheksansatz

Vorteile:

- Funktionalitäten sind strikt getrennt
- Keine Code-Konvergenz
- Eigenständige Wartung
- Verwendbar von BGP-Analyse-Tools

Abbildung 8.5: Gegenüberstellung der Erweiterungsansätze für eine RPKI-Validierung

Aufgrund der deutlichen Unterschiede der beiden Ansätzen hinsichtlich der Vor- und Nachteile wurde sich bei der Umsetzung für den Bibliotheksansatz und somit für eine Implementierung einer *Shared Library* entschieden. Nachfolgend wird das Design der Bibliothek vorgestellt.

8.3.1 Design

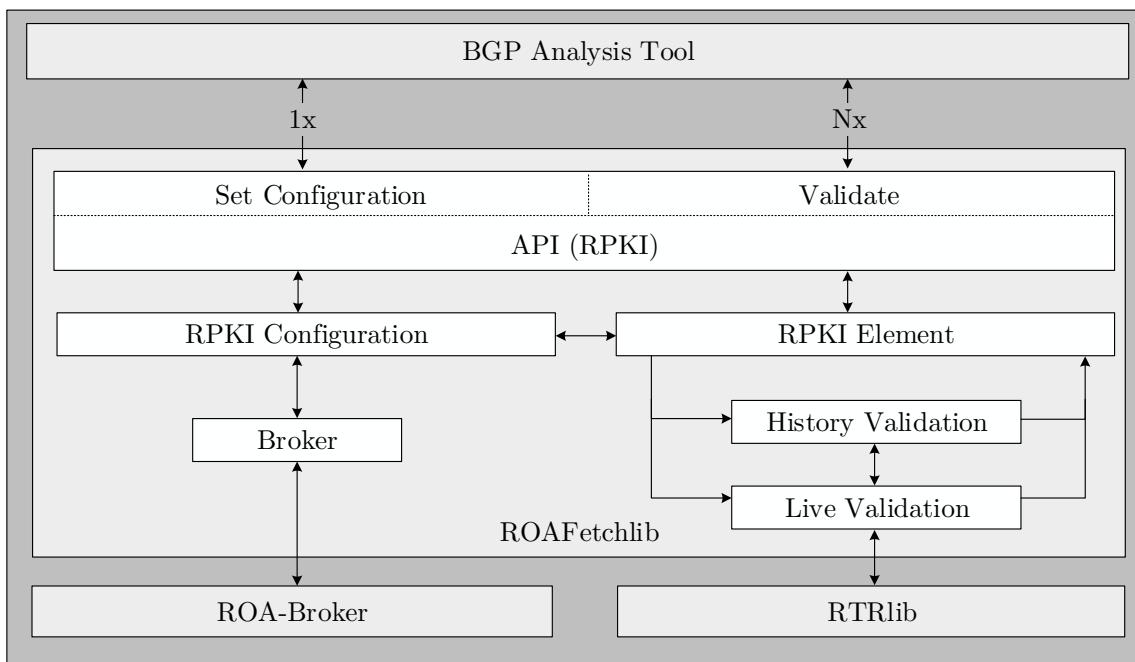


Abbildung 8.6: ROAFetchlib – Design

Das Design der Bibliothek beruht, wie in Abbildung 8.6 dargestellt, auf zwei geteilten Funktionalität. Der erste Teil besteht aus der Konfiguration der RPKI-Umgebung, die eine nachfolgende Validierung ermöglicht. Dafür werden die Meta-Daten des ROA-Broker angefragt und für die native Validierung der BGP-Announcements vorbereitet. Dieser Vorgang wird anfänglich einmalig durchgeführt. Der zweite Teil besteht aus der Validierung der BGP-Daten mithilfe der RTRlib [12] und kann in zwei verschiedene Modi ausgeführt werden: Der historischen und der Echtzeit-basierten Validierung. Dabei werden die Daten aus der RPKI-Konfiguration entnommen und für jedes Element eines BGP-Announcements verwendet.

8.3.2 API

Wie in Abbildung 8.6 verdeutlicht, gibt es zwei verschiedene API-Aufrufe, die es einem Analyse-Tool (BGPStream) ermöglichen, eine native RPKI-Validierung zu unterstützen.

Der erste API-Aufruf (`rpk_set_config`) ermöglicht das einmalige Setzen der RPKI-Konfiguration. Dafür werden, wie in Abbildung 8.7 dargestellt, alle für die Validierung notwendigen Parameter der ROAFetchlib übergeben:

- Komma-getrennte Liste aller Projekte, Kollektoren und Zeit-Intervalle (Strings)
- Parameter für die verbundene oder getrennte Validierung (Integer)
- Parameter der den Modus der Validierung festlegt (Integer)
- URL, die die Adresse des ROA-Brokers definiert (String, optional)
- Komma-getrennte Liste von SSH-Optionen, die eine SSH-Verbindung zum RTR-Server ermöglichen (String, optional)

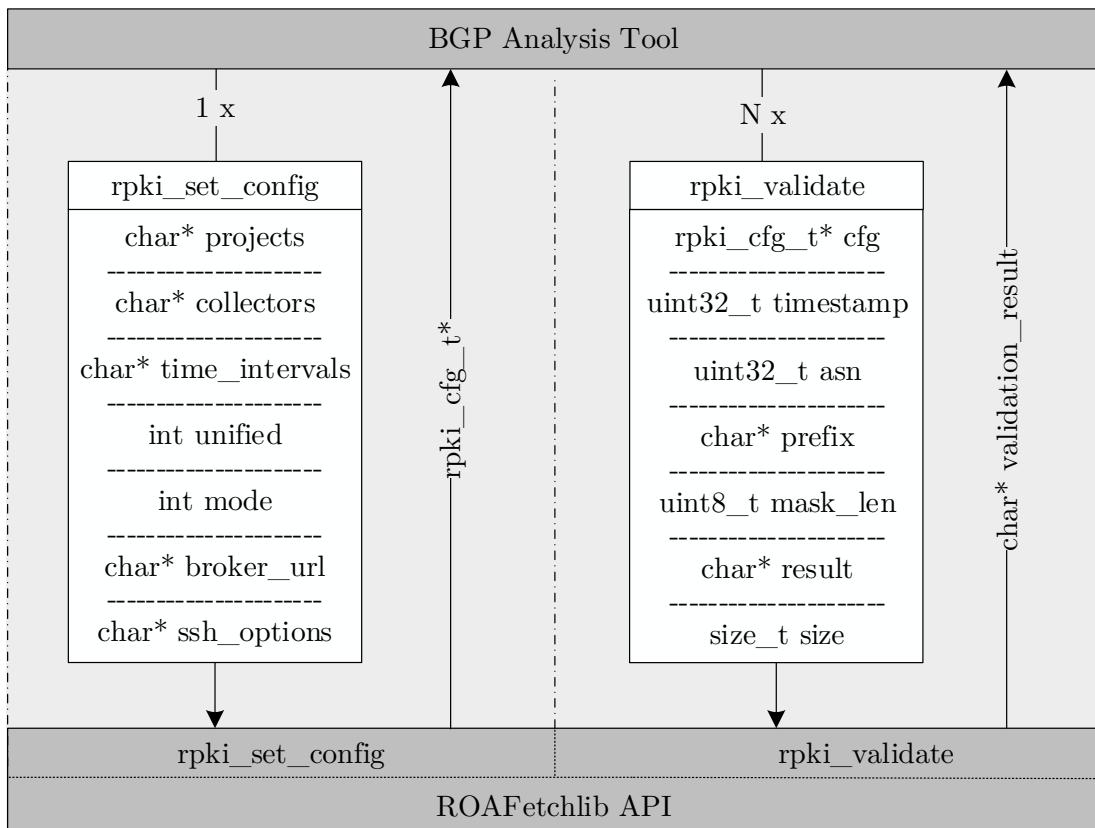


Abbildung 8.7: ROAFetchlib – API

Der API-Aufruf wird von der Bibliothek mit einem Zeiger auf das RPKI-Konfiguration-Konstrukt beantwortet. Mithilfe dieser Konfiguration ist es der ROAFetchlib möglich alle kommenden BGP-Announcements zustandslos zu validieren. Die Validierung per se wird durch den zweiten API-Aufruf möglich, der neben anderen Parametern auch die RPKI-Konfiguration übergibt:

- Zeiger auf das RPKI-Konfiguration-Konstrukt
- Zeitstempel des BGP-Announcement-Elements (`rpki_cfg_t*`)
- Origin-ASN des BGP-Announcement-Elements(`uint32_t`)
- Präfix des BGP-Announcement-Elements (`char*`)
- Subnetz-Maske des Präfixes (`char*`)
- Zeiger auf einen Ergebnis-Buffer (`String`)
- Größe des Ergebnis-Buffers (`size_t`)

Die Bibliothek antwortet auf den Validierungs-API-Aufruf mit einem Validierungsergebnis, das in den übergebenen Ergebnis-Buffer geschrieben wird. Dieser Aufruf wird für jedes folgende BGP-Announcement-Element wiederholt.

Durch die beschriebene API kann BGPSstream nun die RPKI-Funktionalität integrieren und unterstützen. Nachfolgend wird die Integration in BGPSstream näher erläutert.

8.3.3 Integration in BGPSstream

Durch die einfache API der ROAFetchlib ist es möglich, die Integration in BGPSstream auf den Input und auf den Output zu beschränken, wodurch die Bibliotheken sehr flexibel miteinander verbunden sind. Nachdem BGPSstream mit der RPKI-Unterstützung kompliert wurde, kann das Tool mithilfe des BGPReaders (Input) gestartet werden. Neben den herkömmlichen Optionen zum Spezifizieren der BGP-Daten existiert eine weitere Option `-H` mit der die Parameter für die RPKI-Validierung übergeben werden können.

Die ersten drei Parameter spezifizieren den Modus der Validierung, die Vereinigung der ROA-Datensätze und die Möglichkeit der SSH-Verbindung zu einem der RTR unterstützenden Cache-Server. Der Modus der Validierung legt fest, ob die BGP-Daten durch historische ROA-Datensätze oder in Echtzeit, mit einer direkten Verbindung zu einem Cache Server, validiert werden sollen (Abschnitt 8.3.4). Der Parameter, der die Vereinigung der ROA-Datensätze spezifiziert, beschreibt die Art der Validierung der einzelnen Kollektoren. So können die BGP-Announcements mit den ROA-Datensätzen jedes Kollektors getrennt von einander oder durch eine Kombination aller ROA-Datensätze validiert werden. Die SSH-Option definiert, ob bei einer Live-Validierung eine SSH-Verbindung zum ausgewählten Cache Server verwendet werden soll.

Die darauffolgenden drei optionalen Parameter legen die Einstellung zu der SSH-Verbindung fest, falls diese aktiviert wurde. So muss der Benutzername, der Pfad zum Host-Schlüssel und der Pfad zum privaten Schlüssel übergeben werden. Daraufhin folgt eine Liste von Projekt-Kollektor-Paaren definieren. Die Liste spezifiziert welche Kollektoren für die native RPKI-Validierung herangezogen werden sollen.

Nachdem die Parameter eingelesen und auf Korrektheit und Integrität geprüft wurden, kön-

nen die Zeit-Intervalle der BGP-Parameter eingelesen werden. Mithilfe aller Parameter ist es nun möglich, den in 8.3.2 definierten API-Aufruf `rpkiset_config` zu tätigen. Somit wird für die eingegebenen Parameter die entsprechenden Konfiguration erzeugt und zurückgegeben.

```

1  /* Configure RPKI Validation */
2  rpkicfg_t *cfg = NULL;
3  if(rpkactive){
4      char rpkindows[WINDOW_CMD_CNT * RPKI_WINDOW_LEN] = "";
5      char rpk>window[RPKI_WINDOW_LEN];
6      for (i = 0; i < windows_cnt; i++) {
7          snprintf(rpk>window, sizeof(rpk>window), "%" PRIu32 "-%" PRIu32 ",",
8                  windows[i].start, windows[i].end);
9          strncat(rpkindows, rpk>window);
10     }
11     rpkindows[strlen(rpkindows) - 1] = '\0';
12     cfg = rpkiset_config(rpkiprojects, rpkicollectors, rpkindows,
13                           rpkounified, rpkihistorical, RPKI_BROKER,
14                           (rpk ssh && !rpkihistorical) ? rpk ssh_arg : NULL);

```

Quellcode 8.1: API-Aufruf (`rpkiset_config`) für die Erstellung einer RPKI-Konfiguration

Während des Extrahierens der BGP-Announcements durch BGPSstream in Elemente kommt es zum fortlaufenden Output jedes Elements (Output). Jedes dieser Elemente steht für einen atomaren BGP-Eintrag bestehend aus einem Unix-Zeitstempel, Origin-ASN, Präfix und Mask-Länge. Mithilfe dieser Daten, dem bereits erzeugtem Konfiguration Konstrukt und einem Ergebnis-Buffer kann nun der zweite API-Aufruf zur Validierung dieses BGP-Elements getätigert werden. Nachdem die Validierung mittels der ROAFetchlib vollständig ist, wird das Validierungsergebnis in Form eines Strings in den Ergebnis-Buffer gespeichert. Dieser wird als Erweiterung des herkömmlichen BGPSstream-Output hinzugefügt und ausgegeben. Dadurch unterstützt die BGPSstream-Bibliothek eine native RPKI-Validierung der BGP-Announcements.

```

1  if(elem->annotations.rpkactive){
2      char prefix[INET6_ADDRSTRLEN];
3      char result[B_REMAIN];
4      bgpstream_addr_ntop(prefix, INET6_ADDRSTRLEN,
5                            &(((bgpstream_pfx_t *)(&(elem->prefix))->address));
6      uint32_t asn = ((bgpstream_as_path_seg_asn_t *)
7                       bgpstream_as_path_get_origin_seg(elem->aspath))->asn;
8      if(!rpkivalidate(elem->annotations.cfg, elem->timestamp, asn, prefix,
9                        elem->prefix.mask_len, result, sizeof(result))) {
10         c = snprintf(buf_p, B_REMAIN, "%s", result);
11         written += c;
12         buf_p += c;
13     }
14 }

```

Quellcode 8.2: API-Aufruf (`rpkivalidate`) für RPKI-Validierung eines BGP-Elements

Nachfolgend wird die genaue Funktionsweise und die Implementierung der ROAFetchlib vorgestellt und erläutert.

8.3.4 Funktionsweise und Implementierung

Erstellung der RPKI-Konfiguration

Nachdem der erste API-Aufruf zur Erstellung der RPKI-Konfiguration getätigter wurde, wird zunächst eine Konfiguration mithilfe der übergebenen Parameter erzeugt. Dafür wird zunächst eine leere Instanz des RPKI-Konfiguration-Structs (`rpki_cfg_t`) erstellt. Wie in Abbildung 8.8 dargestellt, besteht das RPKI-Konfiguration-Konstrukt wiederum aus einzelnen Konstrukten für die jeweiligen Module.

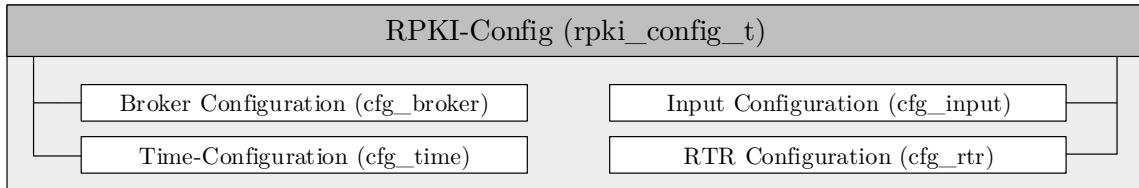


Abbildung 8.8: ROAFetchlib – RPKI-Konfiguration-Struct

Während der Erstellung der Konfiguration wird zunächst die Broker-URL festgelegt, falls eine vom Standardwert abweichende Broker-URL übergeben wurde. Anschließend wird diese in der Broker-Konfiguration gespeichert. Durch die API ist es somit möglich einen anderen ROA-Broker zu verwenden ohne den Standard-Broker zu verändern. Des Weiteren wird für die ROA-URLs, die durch die Metadaten übertragen werden, bereits Speicher auf dem Heap alloziert und der entsprechende Zeiger in der Broker-Konfiguration gespeichert. Da die RPKI-Validierung durch die RTRlib realisiert wird, müssen die ROA-Datensätze später in einer von der RTRlib zur Verfügung gestellten Datenstruktur, der Präfix-Tabelle, gespeichert werden. Diese werden durch ein weiteres Speichersegment verfügbar gemacht, initialisiert und in der RTR-Konfiguration gespeichert.

Alle weiteren übergebenen Parameter werden in der Input- bzw. Time-Konfiguration gespeichert, um sie für einen späteren Zeitpunkt zur Verfügung zu stellen.

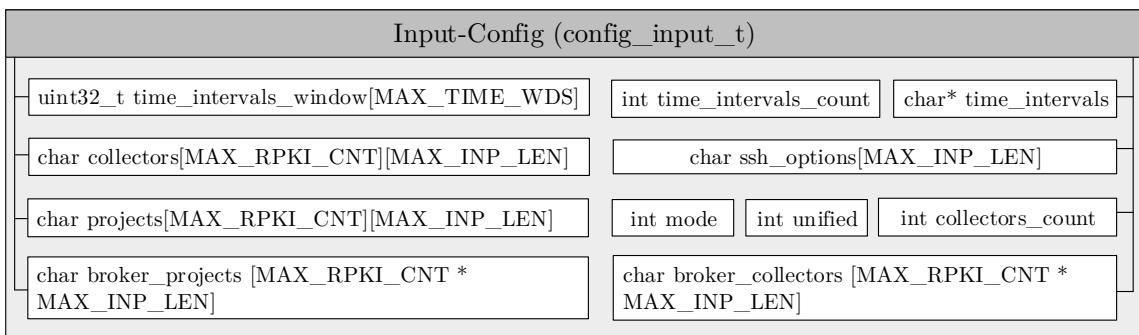


Abbildung 8.9: ROAFetchlib – Input-Konfiguration-Struct

Die entstandene Konfiguration ist nun initialisiert und kann je nach Modus weiter mit Meta-Daten befüllt werden.

Im Falle des Live-Modus werden zunächst die entsprechenden Parameter überprüft, um die

Verfügbarkeit des eingegebenen RTR-Servers sicherzustellen. Für den Live-Modus können somit ausschließlich RTR-Kollektoren verwendet werden. Falls mehrere Kollektoren angegeben werden, wird stets der erste verwendet. Je nach SSH-Optionen wird die Verbindung mit den entsprechenden Parametern verschlüsselt, so dass alle ROA-Datensätze auf einem sicheren Kanal übertragen werden. Alle dafür notwendigen Parameter wie die Konfiguration des RTR-Managers und einer Datenstruktur der RTRlib zur Konfiguration der einzelnen RTR-Sockets des Clients werden in der RTR-Konfiguration gespeichert. Somit besteht ab diesem Zeitpunkt eine Live-Verbindung zum RTR-Cache-Server, wodurch eine Validierung der BGP-Announcements mittels des aktuellen ROA-Datensatzes des Cache Servers möglich ist.

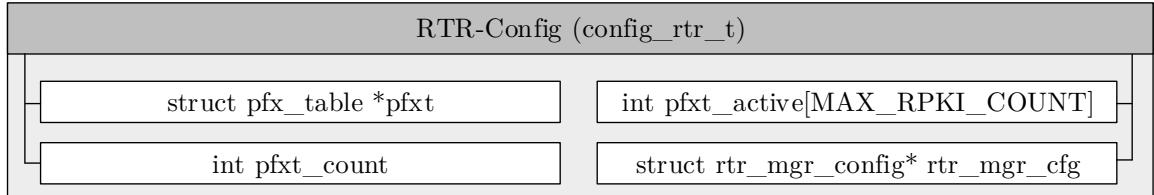


Abbildung 8.10: ROAFetchlib – RTR-Konfiguration-Struct

Im Falle einer historischen Validierung wird zunächst die URI des ROA-Broker mithilfe der übergebenen Parameter für Projekte, Kollektoren und Zeitintervalle erstellt und aufgerufen. Falls es dabei zu einem Fehler kommt oder der ROA-Broker für die eingegebenen Parameter einen Fehler meldet, kann keine korrekte historische Validierung erfolgen, woraufhin der Prozess mit einer entsprechenden Fehlermeldung beendet wird. Bei einer validen Antwort des ROA-Brokers in Form einer JSON-Datei wird diese zunächst geladen und in einen entsprechenden Buffer geladen.

Anschließend erfolgt das Parsing des JSON-Strings durch die Open-Source Bibliothek “JSMN” [<https://github.com/zserge/jsonn>]. Dabei handelt es sich um einen JSON-Parser, der es ermöglicht einen JSON-String auf unterschiedliche Art und Weise zu interpretieren. Für das Parsing der vom ROA-Broker erhaltenen JSON-Datei wurde der Parser dahingehend angepasst, dass zunächst die einzelnen Felder der Meta-Informationen statisch eingelesen und in der Input-Konfiguration abgespeichert werden. Die erneute Speicherung der Parameter ist notwendig, da der ROA-Broker die Reihenfolge der Kollektoren verändert haben kann, um eine eindeutige Zuordnung von Kollektor und ROA-URL zu ermöglichen.

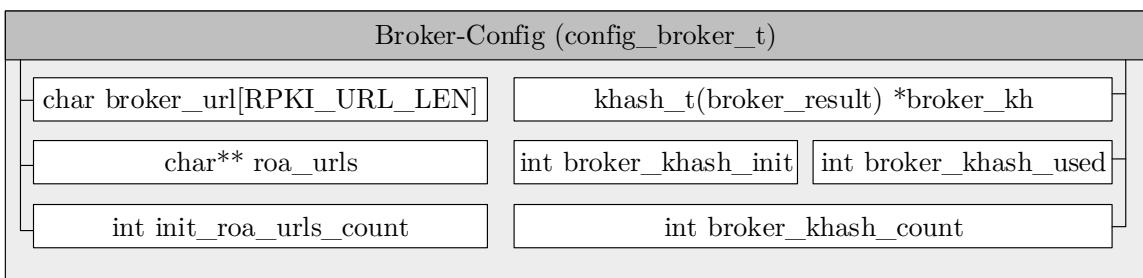


Abbildung 8.11: ROAFetchlib – Broker-Konfiguration-Struct

Anschließend wird die Hash-Tabelle “broker_result“, wie in Abbildung 8.11 verdeutlicht,

initialisiert und mit den ROA-URLs aus den Metadaten des Brokers befüllt. Dafür wurde die generische Hash-Tabelle der Open-Source Bibliothek “Klib” [<https://github.com/attractivechaos/klib>] verwendet. Diese realisiert eine leichtgewichtige und effiziente Implementierung einer Hash-Tabelle. Da die Hash-Tabelle ausschließlich die Organisation der ROA-URLs jedoch nicht die Speicherung der eigentlichen Strings übernimmt, muss das Array, das anfangs auf dem Speicher vorbereitet wurde, je nach Größe der Metadaten realloziert werden, um Buffer-Overflows zu verhindern.

Nachfolgend können die einzelnen ROA-URLs und deren Zeitstempel ausgelesen und in das Array innerhalb der Broker-Konfiguration gespeichert werden. Somit sind die Metadaten und die ROA-URLs in der Konfiguration für jede Iteration der Validierung verfügbar. Das RPKI-Konfiguration-Konstrukt wurde nun für den jeweiligen Modus vorbereitet und mit den entsprechenden Metadaten befüllt, wodurch der erste API-Aufruf mit der Rückgabe der Konfiguration beendet werden kann.

```

1  rpk_i_cfg_t* rpk_i_set_config(char* projects, char* collectors, char*
2   time_intervals, int unified, int mode, char* broker_url, char* ssh_options){
3
4      rpk_i_cfg_t *cfg;
5      if((cfg = cfg_create(projects, collectors, time_intervals, unified, mode,
6          broker_url, ssh_options)) == NULL){
7          debug_err_print("%s", "Error: Could not create RPKI config\n");
8          exit(-1);
9      }
10
11     // Configuration of live mode
12     if(!mode){
13         debug_print("%s", "Info: For Live RPKI Validation the first collector will be
14             taken only\n");
15         live_validation_set_config(cfg->cfg_input.collectors[0], cfg, ssh_options);
16         return cfg;
17     }
18
19     // Configuration of historical mode
20     config_input_t *input = &cfg->cfg_input;
21     broker_connect(cfg, input->broker_projects, input->broker_collectors, input->
22         time_intervals);
23     print_config_debug(cfg);
24     return cfg;
25 }
```

Quellcode 8.3: API-Methode (rpk_i_set_config) zum Setzen der RPKI-Konfiguration

Live-Validierung

Nachdem der zweite API-Aufruf für die Methode – rpk_i_validate – aufgerufen wurde, wird zunächst ein RPKI-Element erzeugt. Ein RPKI-Element spiegelt dabei eine Instanz der RPKI-Validierung eines BGP-Announcements wider.

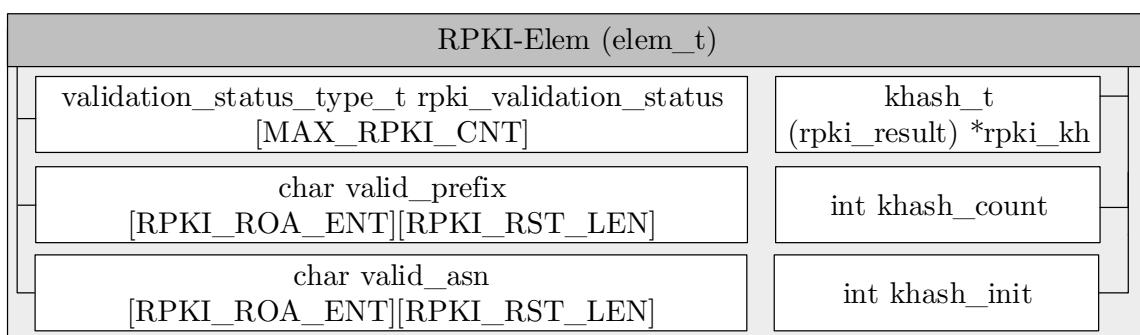


Abbildung 8.12: ROAFetchlib – RPKI-Element

Dafür wird zunächst ausschließlich das Array für die Validierungsergebnisse vorbereitet. Im Falle der Live-Validierung wird daraufhin die Validierung für das RTR-Protokoll mithilfe der Parameter des BGP-Announcements gestartet. Da bereits eine Live-Verbindung zum Cache Server besteht, kann diese mithilfe des RTR-Managers aus der RTR-Konfiguration genutzt werden. Der Cache Server validiert daraufhin die Kombination aus ASN, BGP-Präfix und Subnetz-Maske und antwortet mit einem Validierungsergebnis, einem oder mehreren Gründen und der Anzahl an Gründen. Die Gründe sind dabei Einträge aus dem aktuellen ROA-Datensatz des Cache Servers, die zeigen, dass das spezifizierte BGP-Announcement *valid*, *invalid* oder *notfound* ist. Das Validierungsergebnis wird daraufhin in dem Array *-rpki_validation_status* gespeichert.

Für die Organisation wurde ähnlich wie bereits bei der Broker-Konfiguration eine Hash-Tabelle – *rpki_result* – genutzt, die alle Validierungsgründe nach den Kollektoren gruppiert. Diese organisiert somit das Projekt, den Kollektor, das Ergebnis und den Grund der Validierung für jeden Kollektor.

```

1 int *k_c = &elem->khash_count;
2 khash_t(rpki_result) *rpki_kh = elem->rpki_kh;
3 config_input_t *input = &cfg->cfg_input;
4 for (int i = 0; i < res_reasoned.reason_len; i++) {
5     snprintf(elem->valid_asn[*k_c], sizeof(elem->valid_asn[*k_c]), "%s,%s,%s,%"PRIu8
6             , input->projects[pfxt_count], input->collectors[pfxt_count], elem->
7                 rpki_validation_status[pfxt_count] == ELEM_RPKI_VALIDATION_STATUS_INVALID ?
8                     "invalid" : "valid", res_reasoned.reason[i].asn);
9
10    if(kh_get(rpki_result, rpki_kh, elem->valid_asn[*k_c]) == kh_end(rpki_kh)){
11        k = kh_put(rpki_result, rpki_kh, elem->valid_asn[*k_c], &ret);
12        kh_val(rpki_kh, k) = '\0';
13        lrtr_ip_addr_to_str(&(res_reasoned.reason[i].prefix), reason_prefix, sizeof(
14            reason_prefix));
15        sprintf(elem->valid_prefix[*k_c], RPKI_RST_MAX_LEN, "%s/%" PRIu8 "-%"PRIu8,
16                reason_prefix, res_reasoned.reason[i].min_len, res_reasoned.reason[i].
17                    max_len);
18        kh_val(rpki_kh, k) = elem->valid_prefix[*k_c];
19        elem->khash_count++;
20    } else {
21        char v_prefix[RPKI_RST_MAX_LEN];
22        k = kh_get(rpki_result, rpki_kh, elem->valid_asn[*k_c]);
23        for(int i = 0; i < *k_c; ++i) {if(!strcmp(elem->valid_asn[i], elem->valid_asn[*
24            k_c])) {ret = i;}}
25        lrtr_ip_addr_to_str(&(res_reasoned.reason[i].prefix), reason_prefix, sizeof(
26            reason_prefix));
27        sprintf(v_prefix, sizeof(v_prefix), "%s %s/%" PRIu8 "-%"PRIu8, kh_val(rpki_kh,
28            k), reason_prefix,
29            res_reasoned.reason[i].min_len, res_reasoned.reason[i].max_len);
30        strncpy(elem->valid_prefix[ret], v_prefix, sizeof(elem->valid_prefix[ret]));
31        kh_val(rpki_kh, k) = elem->valid_prefix[ret];
32    }
33}

```

Quellcode 8.4: Befüllung der Hash-Tabelle “*rpki_result*“ mit Validierungsergebnissen

Da während der Live-Validierung ausschließlich ein Kollektor aktiv ist, beschränkt sich entsprechend die Hash-Tabelle auf diesen Kollektor. Analog zur Broker-Hash-Tabelle werden weitere String-Arrays für die eigentliche Speicherung der Validierungs-Ergebnisse und -Gründe benötigt (*valid_asn* und *valid_prefix* in Abbildung 8.12). Nachdem somit die Validierungsergebnisse gespeichert wurden, können diese anschließend ausgegeben werden. Die Ausgabe der RPKI-Elemente unterscheidet sich je nach Modus, so dass das Validierungs-

ergebnis einer vereinigten Präfix-Tabelle einen entsprechend anderen Aufbau hat als der Output von diskret validierten Präfix-Tabellen.

Je nach übergebenen Modus wird somit einer der beiden Output-Varianten verwendet:

- diskret:Project,Collector,Validation_status[,ASN_1,Prefix_1,ASN_2,Prefix]*; || not-found;
- vereinigt: Project_01\Collector_01 Project_02\Collector_02,Validation_status[, ASN_1,Prefix1 ASN_2,Prefix]*; || notfound;

Im diskreten Modus werden die einzelnen Einträge der Hash-Tabelle konkateniert und als Validierungsergebnis in Form eines Strings gespeichert. Im Falle einer vereinigten Validierung werden zunächst alle Projekte und Kollektoren konkateniert, das Validierungsergebnis und die einzelnen Gründe hinzugefügt und ebenfalls in Form eines Strings gespeichert. Abschließend wird das Validierungsergebnis in den übergebenen Ergebnis-Buffer geschrieben und dem API-Aufruf nach der Lösung des Elements zurückgegeben.

Ein Beispiel-Aufruf für die Live-Validierung ist der folgende Befehl:

```
$ bgpreader -p ris -c rrc00 -w 1502378500 -H "0,0,0,FU-Berlin,CC06(RTR)"
```

Historische Validierung

Im Falle einer historischen Validierung wird zunächst geprüft, ob der übergebene Zeitstempel innerhalb der konfigurierten Zeitintervalle liegt. Falls dies nicht der Fall sein sollte, wird die Validierung des Intervalls abgebrochen. Andernfalls muss das zutreffende Zeitintervall bestehend aus Start- und Endzeitstempel ausfindig gemacht werden. Dafür wird zunächst der Eintrag der Broker Hash-Tabelle gesucht, der den jüngsten Zeitstempel vor dem aktuellen BGP-Zeitstempel trägt. Dieser wird, wie in Abbildung 8.13 verdeutlicht, in der Time-Konfiguration als “current_roa_timestamp“ gespeichert.

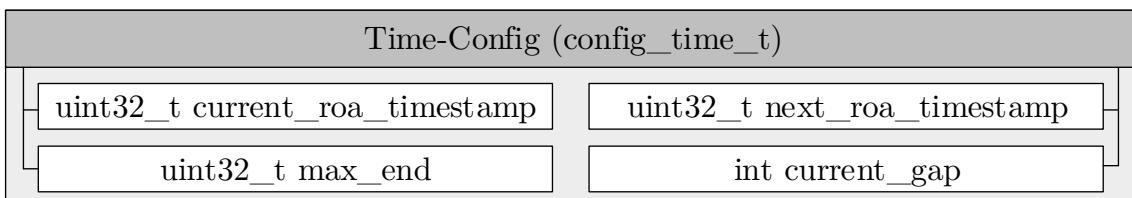


Abbildung 8.13: ROAFetchlib – Time-Konfiguration

Nachfolgend wird der nächst bekannte Zeitstempel, der auf den “current_roa_timestamp“ folgt, ausfindig gemacht und als “next_roa_timestamp“ ebenfalls in der Konfiguration gespeichert. Falls der “current_roa_timestamp“ der letzte verfügbare Zeitstempel der Hash-Tabelle ist, wird “next_roa_timestamp“ mit dem Wert 0 als offenes Zeitintervall definiert.

Zeitgleich werden die ROA-URLs in einen entsprechenden Buffer zwischengespeichert, um anschließend ein Parsing durchzuführen. Da nun der aktuelle Zeitstempel und das zutreffende ROA-Zeitintervall bekannt ist, können die entsprechenden ROA-URLs interpretiert, aufgerufen und als Präfix-Tabelle gespeichert werden. Dafür werden zunächst alle aus der vorherigen Iteration verfügbaren Daten gelöscht und die Datenstrukturen gesäubert.

Anschließend kann die Zuordnung von Kollektor und ROA-URL aus den Metadaten in die RTR-Konfiguration übernommen werden. Je nach Validierungsmodus werden alle ROA-Datensätze aus den ROA-URLs in eine (diskret) oder in unterschiedliche Präfix-Tabelle (vereinigt) importiert. Während des Imports eines ROA-Dumps wird zunächst die Datei geladen und in einen Buffer geschrieben. Anschließend kann jede Zeile des ROA-Dumps in die einzelnen Bestandteile: ASN, Präfix, Max-Länge und Trust Anchor aufgeteilt werden. Falls die Zeile zulässig ist, können die einzelnen Bestandteile in die jeweiligen Datenstrukturen der RTRlib überführt und der zum aktuellen Kollektor gehörigen Präfix-Tabelle hinzugefügt werden.

Nachdem jede Zeile eines ROA-Dumps in die entsprechende Präfix-Tabelle hinzugefügt und somit jedes ROA-Dump aller Kollektoren importiert wurde, ist das Parsing der ROA-URLs abgeschlossen und die historische Validierung vorbereitet. Nun kann die Validierung erfolgen, wobei in diesem Fall nicht der ROA-Datensatz des Cache Servers für die Validierung verwendet wird, sondern die einzelnen (diskret) oder die kombinierte (vereinigt) Präfix-Tabelle(n).

```

1  struct reasoned_result historical_validate_reason(uint32_t asn, char* prefix,
2  {
3      struct lrtr_ip_addr pref;
4      lrtr_ip_str_to_addr(prefix, &pref);
5      enum pfxv_state result;
6      struct pfx_record *reason = NULL;
7      unsigned int reason_len = 0;
8
9      pfx_table_validate_r(pfxt, &reason, &reason_len, asn, &pref, mask_len, &result)
10     ;
11
12     struct reasoned_result reasoned_res;
13     reasoned_res.reason = reason;
14     reasoned_res.reason_len = reason_len;
15     reasoned_res.result = result;
16
17     return (reasoned_res);
18 }
```

Quellcode 8.5: Historische Validierung – Validierungsauftruf der RTRlib

Erneut wird ein Validierungsergebnis erzeugt, in der Hash-Tabelle – gruppiert nach den Kollektoren – organisiert und in den entsprechenden Arrays gespeichert. Je nach Modus werden die einzelnen Validierungs-Ergebnisse und -Gründe kombiniert oder getrennt ausgegeben. Falls nun alle BGP-Announcements des ROA-Zeitintervalls abgearbeitet wurden, erfolgt der Wechsel zu den ROA-URLs des „next_roa_timestamp“. Dafür wird der „next_roa_timestamp“ auf den „current_roa_timestamp“ gesetzt, der nächst verfügbare Zeitstempel ermittelt und das Parsing und der Import der neuen ROA-URLs des Zeitstempels durchgeführt. Mithilfe dieser ROA-Dumps kann nun das nächste Zeitintervall des BGP-Stroms validiert werden bis alle durch das Zeit-Intervall definierten BGP-Daten validiert wurden.

Ein Beispiel-Aufruf für die historische Validierung ist der folgende Befehl:

\$ bgpreader -p ris -c rrc00 -w 1502269200,1502269200 -H “1,0,0,FU-Berlin,CC01“

Hybride Validierung

Da BGPStream die Analyse von endlosen BGP-Daten durch ein offenes Zeit-Intervall erlaubt, muss dieser Modus auf die native RPKI-Validierung übertragen werden. Dieser Modus wird hybrider Modus genannt.

Der hybride Modus erlaubt es ein offenes Zeit-Intervall zu definieren, so dass es möglich ist, von einer historischen Validierung zu einer nahezu Echtzeit-basierten Validierung mithilfe des Brokers oder einem RTR-Cache-Server während der Laufzeit zu wechseln. Im Falle eines offenen Zeit-Intervalls werden zunächst alle historisch abgeschlossenen BGP-Zeit-Intervalle mit den zutreffenden ROA-Dumps validiert bis die Metadaten des ROA-Broker-Aufrufs nicht ausreichen, um die BGP-Daten zu validieren. Indiziert wird dies durch den zuvor beschriebenen Fall, dass kein weiterer Zeitstempel in der Broker Hash-Tabelle existiert (“next_roa_timestamp“ besitzt den Wert 0).

Nachfolgend wird geprüft, ob es sich um eine nahezu Echtzeit-basierte oder eine Echtzeit-basierte Live-Validierung handelt, indem die Differenz des BGP-Zeitstempels zum aktuellen UTC-Unix-Zeitstempel berechnet wird. Befindet sich die Differenz außerhalb des ROA-Intervalls handelt es sich um eine nahezu Echtzeit-basierte Validierung. Andernfalls wird von der historischen zur Echtzeit-basierten Live-Validierung gewechselt.

Im Falle der nahezu Echtzeit-basierten Validierung wird zunächst eine erneute Broker-Anfrage für den Zeitraum nach dem zuletzt verfügbaren Zeitstempel gestellt. Der Broker beantwortet diese Anfrage mit den Metadaten in Form einer JSON-Datei mit den URLs zu den ROA-Datensätzen für den Zeitraum zwischen der letzten Anfrage und dem aktuellen UTC-Unix-Zeitstempel. Analog zur historischen Validierung werden die bereits bestehenden Daten gelöscht und die Datenstrukturen gesäubert, um mit den neuen ROA-URLs und Zeitstempeln ersetzt zu werden. Anschließend erfolgt das Parsing und der Import der neuen ROA-Dumps, wodurch die neu verfügbaren BGP-Daten validiert werden können. Dieser Vorgang wird endlos wiederholt, um alle BGP-Announcements des endlosen BGP-Stroms zu validieren, sobald diese verfügbar sind.

Im Falle der Echtzeit-basierten Live-Validierung wird zunächst die Verbindung zum Cache Server aufgebaut. Der ROA-Broker wird in diesem Fall nicht mehr verwendet, so dass alle nachfolgenden BGP-Announcements ausschließlich mit dem aktuellen ROA-Datensatz des Cache Servers validiert werden. Da jedoch die Dump-Intervalle der gängigen BGP-Kollektoren (RouteView, RIS) über dem ROA-Intervall liegt, wird der hybride Modus zum Wechsel von der historischen Validierung zur Live-Validierung für diese BGP-Kollektoren unter realen Umständen nicht zum Einsatz kommen. Stattdessen ist der ROA-Broker in der Lage für die aktuellsten BGP-Daten der BGP-Kollektoren stets die aktuell passenden ROA-Datensätze zur Verfügung zu stellen.

```
1 // Validate with hybrid mode
2 config_broker_t *broker = &cfg->cfg_broker;
3 if(input->mode && !cfg_time->max_end && timestamp >= cfg_time->
   current_roa_timestamp + ROA_INTERVAL && !cfg_time->next_roa_timestamp) {
4     debug_print("%s", "Info: Entering hybrid mode\n");
5     if(cfg_time->current_roa_timestamp < (uint32_t)time(NULL) - ROA_INTERVAL) {
6       char time_inv[MAX_INTERVAL_SIZE];
7       snprintf(time_inv, MAX_INTERVAL_SIZE, "%"PRIu32"-%"PRIu32, timestamp,
8           cfg_time->max_end);
8       broker_connect(cfg, input->broker_projects, input->broker_collectors,
9                     time_inv);
```

```

9      broker->broker_khash_used = 0;
10     rtr->pfxt_count = 0;
11     cfg_get_timestamps(cfg, timestamp, url);
12     cfg_parse_urls(cfg, url);
13 } else {
14     debug_print("%s", "Info: Entering live mode\n");
15     input->mode = 0;
16     live_validation_set_config(cfg->cfg_input.collectors[0], cfg, input->
17         ssh_options);
18     elem_get_rpki_validation_result(cfg, rtr->rtr_mgr_cfg, elem, prefix, asn,
19         mask_len, NULL, 0);
20     elem_get_rpki_validation_result_snprintf(cfg, result, size, elem);
21     elem_destroy(elem);
22     return 0;
23 }
24 } else if(input->mode && timestamp >= cfg_time->current_roa_timestamp +
25     ROA_INTERVAL &&
26     timestamp < cfg_time->next_roa_timestamp && cfg_time->next_roa_timestamp != 0) {
27     if(cfg->cfg_time.current_gap) {
28         debug_print("Info: No ROA dumps for this ROA interval %"PRIu32" - next
29         available timestamp: %"PRIu32"\n",
30         timestamp, cfg_time->next_roa_timestamp);
31     }
32     cfg->cfg_time.current_gap = 0;
33     strcpy(result, "", size);
34     elem_destroy(elem);
35     return 0;
36 }

```

Quellcode 8.6: Hybrider Modus zum Wechsel des Validierungsmodus

Ein Beispiel-Aufruf für eine hybride Validierung ist der folgende Befehl:

```
$ bgpreader -p ris -c rrc00 -w 1502378500 -H "1,0,0,FU-Berlin,CC01"
```

Validierung eines BGP-Announcements mit einem nicht-passenden ROA-Datensatz

Die verschiedenen Modi der ROAFetchlib sind notwendig, da die Wahl des ROA-Datensatzes zum passenden BGP-Announcement von großer Bedeutung für die Validierung ist. So ist ein ROA-Datensatz stets ausschließlich für das ROA-Intervall gültig, bis ein weiterer ROA-Datensatz mit möglicherweise aktualisierten Daten diesen ersetzt. Um eine semantisch korrekte Validierung durchzuführen, ist es somit essentiell, dass ein BGP-Datensatz mit einem bestimmten Zeitstempel mit dem dazu passenden ROA-Datensatz, dessen Zeit-Intervall diesen Zeitstempel einschließt, validiert wird. Nur so kann sichergestellt werden, dass das BGP-Datum mit den ROA-Daten validiert wird, die zu diesem Zeitpunkt in der Vergangenheit präsent waren.

Falls ein BGP-Announcement mit einem ROA-Datensatz validiert wird, dessen Zeit-Intervall vor bzw. nach dem Zeitstempel des BGP-Announcements liegt, wird das BGP-Datum mit zu der Zeit alten bzw. in der Zukunft liegenden ROA-Daten validiert. In diesem Fall kann es möglich sein, dass während der Zeitspanne zwischen dem ROA-Intervall und dem BGP-Zeitstempel eine Aktualisierung beispielsweise in Form eines Rückzugs eines ROAs die ROA-Daten beeinflusst und somit zu einem anderen Validierungsergebnis führt. In diesem Fall wäre die Validierung semantisch inkorrekt sein.

Dieser Fall kann unter Umständen für BGPStream-Aufrufe mit RPKI-Unterstützung vorkommen, wenn historische BGP-Daten mit dem aktuellen ROA-Datensatz eines Cache-Servers mittels Live-Validierung validiert werden. Dieser Fall wird nicht durch eine Fehlermeldung abgefangen, ist jedoch semantisch inkorrekt.

KAPITEL 9

Verwandte Arbeiten

9.1 BGP

Das Border Gateway Protocol (BGP) [1] gilt als Grundbaustein des Internets, wodurch die gesicherte Funktionsweise von äußerster Wichtigkeit für alle darauf aufbauenden Protokolle ist. Entsprechend sind Analysen und Messungen hinsichtlich des Inter-Domain-Protokolls von hohem Interesse, wodurch bereits viele Einsichten über Stabilität, Routing und Sicherheitsaspekte genommen wurden.

Ein Beispiel für Instabilitäten und divergente Routing-Verhalten sind eigene Richtlinien der Autonomen Systeme. Jedes Inter-Domain-Protokoll (BGP) ist dazu gezwungen, dass lokale Richtlinien-basierte Routing-Metriken Distanz-basierte Routing-Metriken überschreiben dürfen, wodurch Routing-Richtlinien mit kleiner oder keiner globalen Koordinierung angewendet werden. Solch AS-abhängige Richtlinien können dazu führen, dass AS eigensinnig hinsichtlich der Pfad-Wahl agieren. Um dies zu verhindern, thematisieren Analysen die Menge an Erweiterungen und Restriktionen, die dazu führen sollen, dass AS hinsichtlich deren Pfade auf Daten-Ebene nicht eigensinnig handeln oder Daten verfälschen [13]. Des Weiteren wird diskutiert, ob Sicherheitsprotokolle ausschließlich die Daten-Ebene, Steuerung-Ebene oder beide absichern sollten [13]. Eine Statistik über die aktiven unterschiedlichen Routing-Richtlinien der einzelnen AS wird in [14] vorgestellt, wobei der Bezug zu bekannten gängigen Modellen hergestellt wird. Um dieses Problem zu lösen gibt es verschiedene Ansätze, wie z.B. Verbesserungen für künftige Richtlinien mithilfe der Identifizierung des Entwurfsraums und der Charakterisierung entsprechender Kompromisse zwischen lokalen und globalen Richtlinien [15]. Weitere Lösungsansätze werden in [16, 17] vorgestellt, um stabile dennoch möglichst kurze Routen zu ermöglichen.

Ein anderes Beispiel für Instabilitäten für das Routing-Verhalten und die Sicherheit sind Misskonfigurationen der einzelnen BGP-Router in den AS. Die Anzahl an realvorkommenden Misskonfigurationen werden in [18] statistisch berechnet. Daraus wurde deutlich, dass die entstehenden Fehler tiefe Einschnitte das globale Routing hinsichtlich der Integrität von BGP-Announcement und der Menge an fälschlich erstellen Updates. Eine einheitliche Konfiguration der BGP-Router könnte die Mehrheit solcher Probleme lösen [18]. Einen möglichen Lösungsansatz wird in [19] vorgestellt, bei dem Misskonfigurationen geprüft und behoben werden können.

Um weitergehend ein besseres Verständnis über das globale Routing innerhalb von BGP zu erlangen, können dynamische BGP-Beacons verwendet werden [20]. Mithilfe aktiver Untersuchungen konnten so Unterschiede zwischen den gängigen BGP-Router-Implementierungen

– Cisco und Juniper – untersucht werden. Dabei wurde festgestellt, dass die beiden Implementierungen aufgrund anderer Umsetzungen des Standards hinsichtlich des Routings unterschiedlich sind. Eine weitere Analyse thematisiert *Route Flap Damping*, wodurch deutlich wurde, dass die beiden Implementierungen unterschiedliche Auslöser und Aggressivitäten dafür aufzeigen [20]. *Route Flap Damping* kann nicht nur einzelne Router unterbinden sondern auch Nicht-Erreichbarkeit erzeugen [21]. Analysen in [20] zu Inter-Arrival-Zeiten bzw. der Verteilung von Announcements innerhalb einer Sequenz von Updates zeigten, dass diese Verteilung einen Einfluss auf die Performance der Infrastruktur besitzen und höhere Inter-Arrival-Zeiten diese begünstigen.

Aufgrund des hohen Interesses und der großen Bedeutung von Untersuchungen um das BGP-Protokoll, wurden bereits Bibliotheken und Framework entwickelt, die das passive Analysen von enormen BGP-Routing-Daten vereinfachen und ermöglichen. Ein Beispiel dafür ist das – in dieser Arbeit erweiterte – Framework [11] für die Analyse von echtzeitbasierten und historischen BGP-Daten von Monitoren.

9.2 RPKI

Die Sicherheitsaspekte von BGP mithilfe der Ressource Public Key Infrastructure (RPKI) [3] zu sichern, ist seit Jahren Gegenstand aktueller Netzwerkforschungen. Dabei wurden unterschiedliche Analysen hinsichtlich resultierenden Ergebnisse, Verzögerungen, Verbreitung und Skalierung einer vollständig entwickelten Infrastruktur vorgenommen.

So konnte als Ursache für Routen, die als invalide validiert wurden, in [22] mehrheitlich fehlerhafte Konfigurationen der BGP-Router gegenüber zunächst annehmbare Präfix-Hijacks identifiziert werden. Dennoch werden Szenarien diskutiert und untersucht, in denen Richtlinien der BGP-Router mithilfe von verfälschten RPKI-Daten umgangen werden, um BGP-Announcements bestimmter Präfixe mutwillig zu unterbinden. So wird die Möglichkeit für solch eine mutwillige Unterbindung in [23] thematisiert und diskutiert. Weitergehend werden mögliche Lösungsansätze in [24] vorgestellt.

Des Weiteren wurde das Verhältnis von RPKI-gesicherten zu ungesicherten Webseiten wurde in [25] empirisch untersucht. Dabei wurde deutlich, dass wenig bekannte Webseiten ein höheres Potenzial haben durch RPKI gesichert zu werden als bekanntere Webseiten.

Eine weitere Analyse [26] thematisierte die deutlichen Verzögerungen für einen inkrementellen RSYNC-Fetching-Prozess basierend auf der aktuellen Anzahl an Repository und RPKI-Objekten. Die weitergehenden Problemfelder einer vollständig entwickelten RPKI aufgrund des Umfangs an Objekten hinsichtlich der Speicherung und Verteilungsdauer über die Infrastruktur wurde in [27] untersucht.

Für die Analysen wurde verschiedene Bibliotheken und Programme erstellt, die unter anderem Anwendung in der vorliegenden Arbeit fanden, wie z.B. für das Durchsuchen von RPKI Repository durch [28]. Weitergehend kann mithilfe von [29] der Status eines Web-Servers im Browser überprüft werden.

9.3 RPKI Cache Server Inkonsistenzen

In der vorliegenden Arbeit wurde das Problemfeld der Inkonsistenzen zwischen Cache Server innerhalb von RPKI untersucht. Da zu dieser Thematik bisher keine Arbeiten öffentlich existieren, kann kein direkter Vergleich hinsichtlich Methodik und Analyse-Ansätze vorgenommen werden.

KAPITEL 10

Zusammenfassung und Ausblick

10.1 Zusammenfassung

Die Resource Public Key Infrastructure (RPKI) ist Gegenstand aktueller Netzwerk-Forschungen und ein möglicher Ansatz für die Sicherung des BGP-Protokolls mittels Autorisierung von Internet-Ressourcen – den autonomen Systemen und IP-Adressblöcken – durch Zertifikate und öffentliche Schlüssel. Dadurch wird das BGP-Protokoll um eine für jede enthaltene Instanz stets validierbare Vertrauenswürdigkeit erweitert. Eine Schlüsselkomponente dieser Infrastruktur sind die RPKI-Cache-Server, die eine Abstraktionsstufe zwischen dem globalen Repository – einem verteilten Speicher für alle erforderlichen Zertifikate und Schlüssel – und den BGP-Routern darstellen. Um bei BGP-Router eine einheitliche Vertrauenswürdigkeit mittels RPKI zu erzielen, ist die Konsistenz und Integrität dieser RPKI-Cache-Server äußerster Wichtigkeit für diese Infrastruktur, da die BGP-Router ihre Funktionsweise der entstandenen Vertrauenswürdigkeit ihre Funktionsweise entsprechend anpassen.

In der vorliegenden Arbeit wurden zunächst sowohl die Architektur und Funktionsweise des Border Gateway Protocol (BGP) als auch die Resource Public Key Infrastructure (RPKI) beschrieben. Dabei wurde jede Komponente und deren Funktion innerhalb der Infrastruktur näher beleuchtet. Anschließend wurde das Problemfeld der Cache-Konsistenz hinsichtlich der genauen Definition der Konsistenz zwischen zwei Cache Servern abgesteckt. Für die geplanten Analyse wurde verifiziert, dass die aufgezeichneten Daten den realen Daten der Infrastruktur entsprechen. Im nächsten Schritt wurden die Cache-Konsistenzen für mehrere Cache Server analysiert, wobei zunächst mithilfe von Meta-Analysen die während der Messung entstandenen Messartefakte bzw. Messfehler von den zu untersuchenden Daten abgegrenzt wurden. Bei der Inkonsistenz-Analyse wurden Datensätze der einzelnen Cache Server die Inkonsistenzen quantifiziert. Die Untersuchung der Inkonsistenzen machte deutlich, dass sowohl legitime Ereignisse, wie das Signieren von ROA, als auch nicht-legitime Ereignisse, wie Synchronisierungsprobleme innerhalb des globalen Repository, Auslöser für Inkonsistenzen sein können. Ferner wurde festgestellt, dass die Inkonsistenzen erst in Kombination mit der Verzögerung innerhalb der Infrastruktur auftreten. Im Ergebnis ließ sich die Verzögerung innerhalb der RPKI als Hauptursache für die Inkonsistenzen identifizieren.

Um zu prüfen, die festgestellten RPKI-Inkonsistenzen Einfluss auf die Vertrauenswürdigkeit auf den BGP-Routern und somit auf den BGP-Verkehr haben, wurden die Auswirkungen dieser Inkonsistenzen auf die Validierung der BGP-Announcements tiefergehend untersucht. Dabei konnte festgestellt werden, dass die RPKI-Inkonsistenzen direkten Einfluss auf den BGP-Verkehr haben und somit BGP-Inkonsistenzen hinsichtlich der Validierung erzeugen. Dies hat zur Folge, dass verschiedene BGP-Router den gleichen BGP-Verkehr aufgrund ihrer Verbindung mit unterschiedlichen Cache Servern unterschiedlich validieren.

Zusammenfassend konnte festgestellt werden, dass Inkonsistenzen zwischen den RPKI-Cache-Servern, die eine Schlüsselkomponente in der Infrastruktur besitzen, einen erheblichen Einfluss auf die Funktionsweise und Integrität dieser Infrastruktur haben. Daher kann eine Verzögerung innerhalb der Infrastruktur unterschiedliche Vertrauenswürdigkeiten und in Folge dessen unterschiedliche Routing-Verhalten der BGP-Router verursachen.

In der vorliegenden Arbeit wurde ferner der Entwurf und die Implementierung eines ROA-Dump-Archivs, das die vorangegangenen Analysen und Untersuchungen ermöglichte, thematisiert. Das genannte Archiv ermöglicht es für kommende Analysen mit ähnlicher Problematik historische ROA-Datensätze zu analysieren und auszuwerten. In einem weiteren Schritt wurde in der Arbeit auf das Design und die Implementierung einer Bibliothek zur Analyse der Auswirkungen auf BGP eingegangen. Mithilfe dieser Bibliothek, die im Rahmen dieser Arbeit eigenständig erstellt wurde, ist es künftig möglich, historische BGP-Daten mit den archivierten historischen ROA-Datensätzen zu validieren. Um eine möglichst nutzerfreundliche Handhabung zu realisieren, wurde eine Einbindung in das derzeitig gängigste Tool zur BGP-Analyse (BGPStream) umgesetzt.

Die Arbeit benennt abschließend einzelne Themenbereiche, die in weiterführenden Untersuchungen detaillierter analysiert werden könnten, wie z.B. die detaillierte Analyse der Verzögerungen hinsichtlich der Bearbeitungszeiten der Cache Server, die explizite Identifizierung der Auslöser der Inkonsistenzen und die theoretische Betrachtung der Auswirkungen der RPKI-Inkonsistenzen auf den BGP-Verkehr ohne die Einbeziehung realer BGP-Daten. Für künftige Arbeiten werden der aktuelle Sachstand dargelegt, die Sinnhaftigkeit beschrieben und teilweise bereits Lösungsansätze, wie z.B. für die Behebung der Problematik der Verzögerung innerhalb der Infrastruktur, deren Effekte und mögliche Verbesserungen, angeboten.

10.2 Ausblick

Wegen des beachtlichen Umfangs des Themas und der zeitlichen Begrenzung war es nicht möglich, jeden Aspekt vollständig und in zufriedenstellendem Maße zu analysieren und auszuleuchten.

Nachfolgend werden einzelne noch offene Aspekte und künftige mögliche Lösungsansätze oder Verbesserungen vorgeschlagen.

Delay - Untersuchung der Anfragenverarbeitung des Cache Server

Ein Teil der Zeitproblematik aus Kapitel 4.1.2 war die Bearbeitung der ROA-Anfragen des Cache Server. In der vorliegenden Arbeit wurde dieser Aspekt hinsichtlich der Implementierung und genauen Befehlsabfolge des RIPE-Validators nicht weiter untersucht. So wäre es

für weitere Betrachtungen der Delay-Analyse wichtig, diesen Aspekt detaillierter zu betrachten, indem beispielsweise die Art der Bearbeitung der Anfragen untersucht wird. Hierbei könnte analysiert werden, ob der Cache Server jede Anfrage einzeln empfängt, schnellstmöglich einen ROA-Mitschnitt erzeugt, um die Verzögerung zwischen Anfrage-Zeitpunkt und Zeitpunkt des Mitschnitts gering zu halten und antwortet. Eine andere Möglichkeit wäre die Bearbeitung auf Basis einer pipeline-orientierten Implementierung.

Des Weiteren wurde bei diesem Thema festgestellt, dass die aktuellen Log-Dateien keine Informationen über den genauen Zeitpunkt des ROA-Mitschnitts angeben. Mithilfe dieser Informationen könnte jedoch die erwähnte Verzögerung zwischen den beiden Zeitpunkten genauer untersucht und hinsichtlich ihrer Auswirkung auf mögliche Mess-Ungenauigkeiten besser ausgewertet werden.

Relative Messmethodik der Trust-Anchor-Analyse

Für die Untersuchung der Präsenz von Trust Anchor wurde eine Messmethodik entworfen, die auf der absoluten Anzahl der Cache-Einträge beruht, die diesem Trust Anchor unterliegen. Daher kann festgestellt werden, ob ein Trust Anchor für einen Cache Server präsent oder nicht präsent ist. Die Verwendung eines relativen Werts zwischen der Anzahl der Cache-Einträge im ROA-Datensatz, die diesem Trust Anchor unterstehen, und der Gesamtanzahl der möglichen Cache-Einträge dieses Trust Anchor würde die Messmethode verbessern. Die Folge wäre eine genauere Untersuchung der Präsenz, da die einzelnen Trust Anchor stark abweichende Cache-Eintragsmengen besitzen.

Unterschiede pro Ressourcen Besitzer des Präfixes

Eine weitere Analyse zu Inkonsistenzen zwischen Cache Servern ist die Untersuchung der Ressourcen-Besitzer der von den Inkonsistenzen betroffenen ROA. Hier wären z.B. die einzelnen Differenzen hinsichtlich der Ressourcen Besitzer weiter zu untersuchen. Des Weiteren könnten die einzelnen Ressourcen-Besitzer im Falle einer Inkonsistenz, die auf einem Unterschied der ASN beruht, gegenübergestellt werden, um so mögliche Verteilungen weiter zu verfolgen.

ROA-Unterschiedsanalyse - Verzögerung aufgrund Signierung neuer ROAs

Während der Arbeit wurde eine Überschneidung (mit einem Offset der Verzögerung innerhalb der Infrastruktur) der beiden Zeitpunkte bezüglich des Signierens von ROA und Erscheinen von Inkonsistenzen, die auf den Präfixen der signierten ROA basieren, festgestellt. Es wurde davon ausgegangen, dass es sich beim Signieren von ROA um den Auslöser handelt, während die Verzögerung innerhalb der RPKI die Ursache für die Inkonsistenzen bildet. Diese Annahme wurde in der Arbeit nicht vertiefend untersucht, so dass nicht mit eindeutiger Bestimmtheit feststeht, die Signierung der Auslöser für die Inkonsistenzen ist. Diese Problematik sollte hinsichtlich der Kausalität genauer untersucht werden.

Des Weiteren sollte eine Analysemöglichkeit zur Untersuchung der Historie des Signierens von ROA bzw. der Validierungsperiode erzeugt werden, so dass es möglich ist, den Zeitpunkt der Signierung rückwirkend zu ermitteln und zu untersuchen.

Theoretische Betrachtung der BGP-Auswirkungen

Die Untersuchung der RPKI-Inkonsistenzen hinsichtlich der Auswirkungen auf den BGP-Verkehr wurde in dieser Arbeit mit realen Daten von BGP-Announcements analysiert. Diese Methode hat den Vorteil, dass die unmittelbare Auswirkung für den Messzeitraum – wie es in der Realität passiert wäre – gemessen wird. Der Nachteil dabei ist jedoch, dass die Untersuchung der Auswirkungen stark von den BGP-Announcements bzw. den darin enthaltenen ASN-Präfix-Kombinationen abhängt. Dadurch können nicht alle Auswirkungen der Inkonsistenzen sichtbar gemacht werden. Hier sollte eine weitere theoretische Betrachtung ansetzen.

Ein weiter Aspekt wäre die Analyse der ROA-Datensätze, die zu Inkonsistenzen führen, in Hinblick auf das Validierungsergebnis eines beliebigen Präfixes. Sie könnte helfen, die Frage nach der Höhe der Wahrscheinlichkeit, dass ein beliebiger Präfix in unterschiedlichen ROA-Datensätzen zu verschiedenen Validierungsergebnissen führt, zu beantworten. Mithilfe dieser Analyse wäre die Aussage über den Einfluss einer Inkonsistenz auf die BGP-Validierung unabhängig vom eigentlichen BGP-Verkehr.

Auswirkungen der Inkonsistenzen auf die betroffenen Subnetze eines Präfixes

Eine weitere Untersuchung für die Inkonsistenz-Analyse beschäftigt sich mit den Auswirkungen, die eine Inkonsistenz auf die betroffenen Präfixe hat. Dabei wird als Vergleichskriterium die Anzahl an betroffenen Subnetzen eines Präfixes herangezogen. Somit könnte die Frage, wie stark die Auswirkungen für Unterschiede von einem Präfix auf die darunter liegenden Subnetze sind, beantwortet werden. Diese Analyse könnte eine Klassifizierung der Cache-Einträge hinsichtlich der Anzahl der betroffenen Subnetze vornehmen, wodurch die Inkonsistenzen ebenfalls eine Klassifizierung entsprechend der betroffenen Präfixe erhalten würden.

Verteilung der Problematiken, Auslöser und Ursachen

Während der Arbeit wurden unterschiedliche Problematiken, Auslöser, die direkten Einfluss auf RPKI-Inkonsistenzen zwischen den einzelnen Cache Servern haben, festgestellt und untersucht. Eine weitere Untersuchung könnte sich der Verteilung dieser Auslöser und Problematiken hinsichtlich der Anzahl an betroffenen Cache-Einträge widmen, so dass die Ausprägung einzelner Ursachen und Auslöser analysiert würden.

Untersuchung von Lösungsansätzen zur Verbesserung der Konsistenz

Die Untersuchung der Konsistenz der Cache Server ergab, dass die Inkonsistenzen trotz der unterschiedlichen Auslöser und Problematiken auf die Ursache der Verzögerung zurückzuführen sind, so dass diese als Hauptursache für die Differenzen zwischen den Cache Servern identifiziert werden konnte. Um dieses Problem zu beheben, können aufgrund der inhaltlichen Betrachtung der Thematik unterschiedliche allgemeine Lösungsansätze zur Minderung der Verzögerung dargelegt werden, deren Untersuchung und Bewertung Gegenstand künftiger Arbeiten sein könnte.

Wie aus den Inkonsistenz-Analysen deutlich wurde, ist das asynchrone Fetching der Cache Server zum globalen Repository ein entscheidender Faktor der Verzögerung zwischen den einzelnen Cache Servern. So kam es während der Messung dazu, dass ein Cache Server ein Fetching durchgeführt hat und bereits die erhobenen Daten an die BGP-Router überträgt, obwohl ein anderer Cache Server zur gleichen Zeit noch kein Fetching durchgeführt hat. Das kann dazu führen, dass der schnellere Cache Server eine weitere Fetching-Iteration im gleichen Zeitraum durchführt. Solch ein asynchroner Prozess hat unmittelbare Auswirkungen auf die Konsistenz der Daten und kann somit durch eine verteilte Synchronisierung der Cache Server behoben werden. Ein synchronisierter Fetching-Prozess wäre ein möglicher Lösungsansatz, damit jeder Cache Server zeitgleich die gleichen Objekte vom globalen Repository erhält.

Die Prämisse für solch einen Synchronisierungsprozess ist eine einheitliche Konfiguration der Cache Server hinsichtlich Faktoren wie z.B. Fetching-Intervall oder Menge der eingebundenen Trust Anchor. Da die Wahl des Fetching-Intervalls neben der Synchronisierung für die Konsistenz der ROA-Datensätze von Bedeutung ist, da es die durchschnittliche Menge der übertragenen Objekte bestimmt, sollte der Wertebereich des Intervalls so gering wie möglich angesetzt werden, ohne eine entsprechend höhere Netzwerklast oder Performance-Einbuße zu verzeichnen. So könnte im Falle von Verzögerung oder Ausfall ein geringeres Fetching-Intervall schneller einen erneuten konsistenten Zustand wiederherstellen und somit die Anzahl der Inkonsistenzen vermindern.

Neben der Ursache der Inkonsistenzen sollten Auslöser, wie nicht-legitime Ereignisse (beispielsweise Synchronisierungsprobleme) innerhalb der globalen Repository möglichst nicht vorkommen bzw. im Falle eines Ereignisses schnell erkannt werden. Dafür sollten mögliche Mechanismen zur Detektion solcher nicht-legitimer Ereignisse entworfen und deren Auswirkungen auf die Konsistenz des globalen Repository untersucht werden.

Literaturverzeichnis

- [1] “Border Gateway Protocol (BGP),” RFC 1105, Jun. 1989. [Online]. Available: <https://rfc-editor.org/rfc/rfc1105.txt>
- [2] Y. Rekhter, S. Hares, and D. T. Li, “A Border Gateway Protocol 4 (BGP-4),” RFC 4271, Jan. 2006. [Online]. Available: <https://rfc-editor.org/rfc/rfc4271.txt>
- [3] M. Lepinski and S. Kent, “An Infrastructure to Support Secure Internet Routing,” RFC 6480, Feb. 2012. [Online]. Available: <https://rfc-editor.org/rfc/rfc6480.txt>
- [4] “Internet Protocol,” RFC 791, Sep. 1981. [Online]. Available: <https://rfc-editor.org/rfc/rfc791.txt>
- [5] J. A. Hawkinson and T. J. Bates, “Guidelines for creation, selection, and registration of an Autonomous System (AS),” RFC 1930, Mar. 1996. [Online]. Available: <https://rfc-editor.org/rfc/rfc1930.txt>
- [6] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280, May 2008. [Online]. Available: <https://rfc-editor.org/rfc/rfc5280.txt>
- [7] M. Lepinski, A. Chi, and S. Kent, “Signed Object Template for the Resource Public Key Infrastructure (RPKI),” RFC 6488, Feb. 2012. [Online]. Available: <https://rfc-editor.org/rfc/rfc6488.txt>
- [8] S. Weiler, D. Ward, and R. Housley, “The rsync URI Scheme,” RFC 5781, Feb. 2010. [Online]. Available: <https://rfc-editor.org/rfc/rfc5781.txt>
- [9] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein, “The RPKI Repository Delta Protocol (RRDP),” RFC 8182, Jul. 2017. [Online]. Available: <https://rfc-editor.org/rfc/rfc8182.txt>
- [10] R. Bush and R. Austein, “The Resource Public Key Infrastructure (RPKI) to Router Protocol,” RFC 6810, Jan. 2013. [Online]. Available: <https://rfc-editor.org/rfc/rfc6810.txt>
- [11] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, “BGPStream: a soft-

- ware framework for live and historical BGP data analysis,” in *Internet Measurement Conference (IMC)*, Nov 2016.
- [12] M. Wählisch, F. Holler, T. C. Schmidt, and J. H. Schiller, “Rtrlib: An open-source library in c for rpki-based prefix origin validation,” in *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*. Washington, D.C.: USENIX, 2013. [Online]. Available: <https://www.usenix.org/conference/cset13/workshop-program/presentation/W{ä}hlisch>
 - [13] S. Goldberg, S. Halevi, A. D. Jaggard, V. Ramachandran, and R. N. Wright, “Rationality and traffic attraction: Incentives for honest path announcements in bgp,” in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM ’08. New York, NY, USA: ACM, 2008, pp. 267–278. [Online]. Available: <http://doi.acm.org/10.1145/1402958.1402989>
 - [14] P. Gill, M. Schapira, and S. Goldberg, “A survey of interdomain routing policies,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 1, pp. 28–34, Dec. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2567561.2567566>
 - [15] T. G. Griffin, A. D. Jaggard, and V. Ramachandran, “Design principles of policy languages for path vector protocols,” in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM ’03. New York, NY, USA: ACM, 2003, pp. 61–72. [Online]. Available: <http://doi.acm.org/10.1145/863955.863964>
 - [16] T. G. Griffin, F. B. Shepherd, and G. Wilfong, “The stable paths problem and interdomain routing,” *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, Apr. 2002. [Online]. Available: <http://dl.acm.org/citation.cfm?id=508325.508332>
 - [17] L. Gao and J. Rexford, “Stable internet routing without global coordination,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, Dec. 2001. [Online]. Available: <http://dx.doi.org/10.1109/90.974523>
 - [18] R. Mahajan, D. Wetherall, and T. Anderson, “Understanding bgp misconfiguration,” *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 3–16, Aug. 2002. [Online]. Available: <http://doi.acm.org/10.1145/964725.633027>
 - [19] N. Feamster and H. Balakrishnan, “Detecting bgp configuration faults with static analysis,” in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI’05. Berkeley, CA, USA: USENIX Association, 2005, pp. 43–56. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1251203.1251207>
 - [20] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan, “Bgp beacons,” in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, ser.

- IMC '03. New York, NY, USA: ACM, 2003, pp. 1–14. [Online]. Available: <http://doi.acm.org/10.1145/948205.948207>
- [21] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz, “Route flap damping exacerbates internet routing convergence,” in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: ACM, 2002, pp. 221–233. [Online]. Available: <http://doi.acm.org/10.1145/633025.633047>
- [22] M. Wählisch, O. Maennel, and T. C. Schmidt, “Towards detecting bgp route hijacking using the rpki,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 103–104. [Online]. Available: <http://doi.acm.org/10.1145/2342356.2342381>
- [23] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg, “On the risk of misbehaving rpki authorities,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 16:1–16:7. [Online]. Available: <http://doi.acm.org/10.1145/2535771.2535787>
- [24] E. Heilman, D. Cooper, L. Reyzin, and S. Goldberg, “From the consent of the routed: Improving the transparency of the rpki,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 51–62, Aug. 2014. [Online]. Available: <http://doi.acm.org/10.1145/2740070.2626293>
- [25] M. Wählisch, R. Schmidt, T. C. Schmidt, O. Maennel, S. Uhlig, and G. Tyson, “Ripki: The tragic story of rpki deployment in the web ecosystem,” in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 11:1–11:7. [Online]. Available: <http://doi.acm.org/10.1145/2834050.2834102>
- [26] S. Kent and K. Sriram, “Rpki rsync download delay modeling,” Tech. Rep., 2013.
- [27] E. Osterweil, T. Manderson, R. White, and D. McPherson, “Sizing estimates for a fully deployed rpki,” Tech. Rep. 1120005 version 2, 2012.
- [28] A. Reuter, M. Wählisch, and T. C. Schmidt, “Rpki miro: Monitoring and inspection of rpki objects,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 107–108, Aug. 2015. [Online]. Available: <http://doi.acm.org/10.1145/2829988.2790026>
- [29] M. Wählisch and T. C. Schmidt, “See how isps care: An rpki validation extension for web browsers,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 115–116, Aug. 2015. [Online]. Available: <http://doi.acm.org/10.1145/2829988.2790034>