Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Dipl.-Ing. Stephan Günther, M.Sc.
Johannes Naab, M.Sc.

Programmieraufgaben zur Vorlesung Grundlagen Rechnernetze und Verteilte Systeme

Programmieraufgabe 3 – IPv6 Traceroute (3 Punkte)
(9. Juni – 28. Juni 2015)

Abgabe via SVN bis Sonntag, 28. Juni 2015, 23:59:59.99 Uhr (MESZ)

1 Traceroute

Traceroute ermöglicht es, den Weg eines Paketes zu einem Zielrechner zu verfolgen, und Knotenpunkte, über welche das Paket weitergeleitet wird, zu erkennen. Dazu werden *ICMP Echo Requests* [1] an die entsprechende Zieladresse mit aufsteigendem HL (Hop Limit)[2], beginnend mit einem HL von 1, versendet. Wenn ein Knoten im Netzwerk ein Paket mit einem HL von 0 empfängt oder auf 0 dekrementiert, sendet er ein *ICMP Time Exceeded* an den Absender des Pakets zurück. Durch sukzessives Inkrementieren des HL erhält der Absender auf diese Weise Informationen über alle antwortenden Knoten. Für jeden Hop können mehrere Echo Requests versendet werden, um somit z. B. Paketverlust zu kompensieren oder parallele Pfade zu erkennen. Ziel dieser Programmieraufgabe ist es, *Traceroute* für IPv6 zu implementieren. Dazu müssen Sie Echo Requests mit aufsteigendem HL versenden und die jeweiligen Antworten interpretieren und ausgeben.
Hierbei sollen der Timeout je Versuch (`-t <timeout in sec>`, default 5), die Anzahl der Versuche je Hop (`-q <attempts>`, default 3) und die maximale Anzahl der Hops (`-m <max hops>`, default 15) dem Programm als Parameter übergeben werden können. Auf einen versendeten ICMPv6 Echo Request können verschiedene Antworten erfolgen, die entsprechend zu interpretieren sind. Beachten Sie, dass Sie eine empfangene Antwort auf ihre Plausibilität und Validität überprüfen. Dazu zählt insbesondere auch die Verifizierung der Checksum. Weiterhin sollte Ihr Programm mit dem Empfang von fehlerhaften bzw. unvollständigen Paketen sinnvoll umgehen können.
Folgende Antworten müssen verarbeitet werden:

- Time Exceeded. Falls valide, handelt es sich um eine Rückmeldung eines Knotens auf der Route zum Zielrechner.
- Echo Reply. Falls valide und vom Zielrechner gesendet, ist die Route zum Ziel nun vollständig bekannt. Es sollen nach Vervollständigung des aktuellen Hops keine weiteren Probes mit höherem *Hop Limit* mehr gesendet werden.
- Destination Unreachable. Falls valide, konnte der Zielrechner nicht erreicht werden. Es sollen nach Vervollständigung des aktuellen Hops keine Probes mit höherem *Hop Limit* mehr gesendet werden.
- Timeout. Keine andere gültige Antwort wurde erhalten. Es muss der Echo Request für den nächsten Versuch bzw. das nächste Hop Limit versendet werden. Ein Timeout kann auftreten, da nicht alle Router Fehlernachrichten versenden bzw. diese verloren gehen können.

Im Gegensatz zu den vorherigen beiden Programmieraufgaben arbeiten wir direkt auf der Netzwerkschicht (Layer 3). Die Layer 2 Funktionalität wird vom Betriebssystem bereit gestellt.
Wie in den letzten Programmieraufgaben werden auch dieses Mal wieder Rahmenprogramme für C und Java, über das SVN Repository unter *pub/assignment3/* zur Verfügung gestellt. Es bietet sich an, die Rahmenprogramme als Grundlage für die Abgabe zu verwenden. Stellen, an denen die Programme modifiziert werden müssen, sind mit “TODO” gekennzeichnet.
Um Ihnen eine Rückmeldung zu Ihrer Abgabe vor der Deadline zu ermöglichen, können Sie wieder das Test-Framework nutzen, welches die eingecheckte Programmversion auf ihre Funktionsfähigkeit testet. Wenn noch Unklarheiten zur Benutzung von Makefiles und Testumgebung bestehen, halten Sie sich bitte an die Angaben vom Aufgabenblatt *assignment1*.

Wichtig:

- Es dürfen Standard-Bibliotheksfunktionen zur Umwandlung von IP-Adressen zwischen Byte- und Textrepräsentation verwendet werden.
- (Adaptierte) Code Snippets (z. B. von *Stackoverflow*, Umfang von mehr als einer Zeile) *müssen* mit der Quelle (URL) versehen sein.

[1] Auch als *Ping* bekannt.
[2] bei IPv4 TTL (Time to Live)

- Die Test-Ergebnisse sind *nicht* die alleinige Bewertungsgrundlage für die Abgaben sondern stellen lediglich eine Orientierung dar, ob die Abgabe den Erwartungen entspricht. Kriterien, die nicht von dem Test-Framework berücksichtigt werden können, sind z. B. die korrekte Berücksichtigung der Byte-Order oder graceful Termination ohne Exceptions. Auch die Qualität des Source Codes wird bei der Bewertung berücksichtigt.

2 Paketformate

Ihnen werden dazu im Folgenden erneut die Grundstrukturen von IPv6[3] Header, möglichen Extension Headers sowie den ICMPv6[4] Nachrichten gegeben. Für eine detaillierte Beschreibung halten Sie sich an die referenzierten RFCs bzw. die Vorlesung.
Da in der letzten Programmieraufgabe einige Unklarheiten zu der Verwendung von IPv6 Extension Headers bestanden, halten Sie sich bitte für diese Aufgabe an die folgenden Vorgaben.
Ihre Implementierung sollte die folgenden Extension Header behandeln können:

1. Hop-By-Hop Options (`0x00`)
2. Routing (`0x2b`)
3. Destination Options (`0x3c`)

Diese entsprechen dem in der Vorlesung und in Abbildung 2 gegebenen Format. Der Inhalt und die Reihenfolge der Header müssen ignoriert werden. Pakete mit anderen Next-Header Typen müssen verworfen werden. Wichtig ist die Behandlung einer anschließenden ICMPv6 Nachricht.

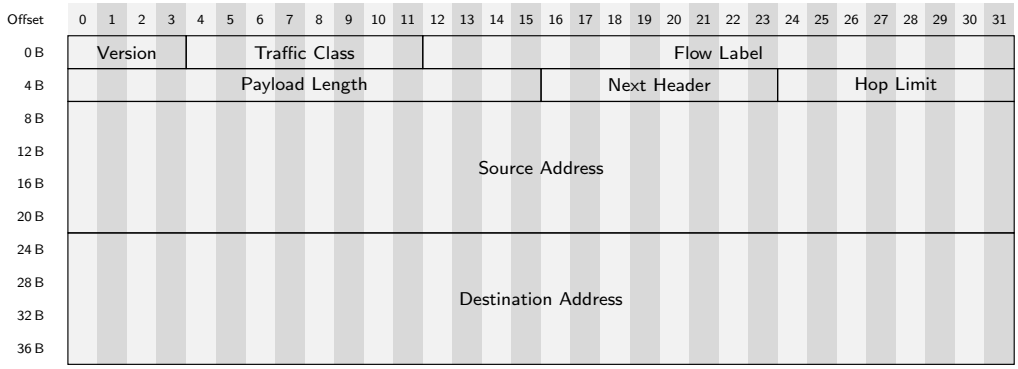

Abbildung 1: IPv6-Header[5]. *Next Header* identifiziert den Typ des nächsten (eingekapselten) Headers bzw. Layer 4 Protokolls. Im Falle von Traceroute enthält das IPv6 Paket eine ICMPv6 Nachricht vom Header-Typ `0x3a`. *Hop Limit* ist das IPv6-Äquivalent zur TTL von IPv4. Es soll für Traceroute mit 1 beginnen und anschließend sukzessive inkrementiert werden. Gefolgt werden die Felder von Extension Headers oder der gekapselten Payload – im Falle von Traceroute mit der ICMPv6 Nachricht.

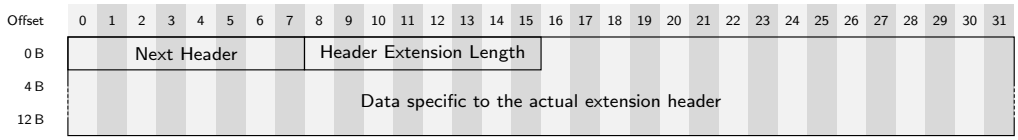

Abbildung 2: „Generischer“ IPv6 Extension-Header. IPv6 Pakete können mehrere Extension Header besitzen. Der Typ eines solchen Headers wird im *Next Header* Feld des vorangehenden IPv6-Headers bzw. Extension-Headers definiert. Im Extension-Header selbst, wird erneut der Typ des nachfolgenden Headers (*Next Header*), sowie die Länge der Extension (*Header Extension Length)* definiert. Die *Header Extension Length* wird in Vielfachen von 8 B angegeben und zählt die ersten 8 B nicht mit. In dieser Programmieraufgabe sollen nur Extension Header mit Typen `0x00`, `0x2b` und `0x3c` behandelt werden. Ein Next-Header mit Wert `0x3a` zeigt eine nachfolgenden ICMPv6-Payload an.

[3] `https://tools.ietf.org/html/rfc2460`
[4] `https://tools.ietf.org/html/rfc4443`
[4] `https://tools.ietf.org/html/rfc2460#section-3`

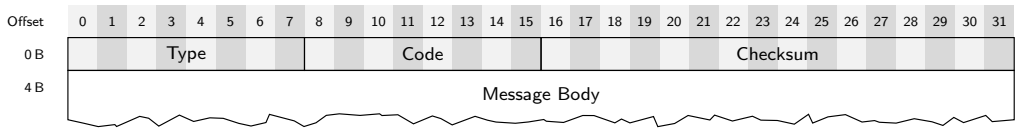

Abbildung 3: ICMPv6-Nachricht[6]. Zur Berechung der *Checksum* wird ein Pseudo-Header verwendet.

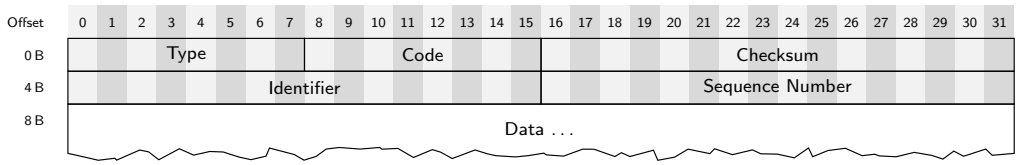

Abbildung 4: Echo Request[7] und Echo Reply[8]. Der *Identifier* und die*Sequence Number* werden verwendet, um auch bei parallelen Anfragen die Antwort eindeutig der Anfrage zuordnen zu können. Für Traceroute soll der *Identifier* anfangs zufällig gewählt werden. Nachfolgende Nachrichten sollen den selben *Identifier* verwenden. Die erste Nachricht soll die *Sequence Number* `0` verwenden, alle nachfolgenden Nachrichten sollen eine fortlaufend um 1 inkrementierte *Sequence Number* verwenden. Der Echo Request kann in *Data ...* beliebige Daten enthalten. Der zugehörige Echo Reply muss die Daten aus dem Echo Request enthalten. In dieser Programmieraufgabe sollen *keine* Daten an die Nachricht angehängt werden.

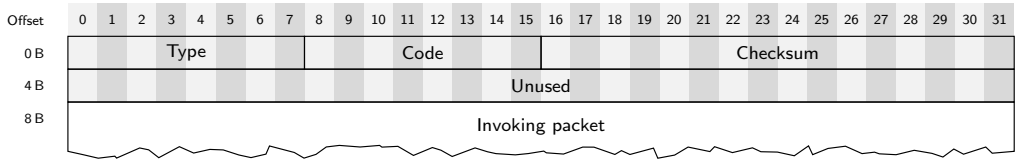

Abbildung 5: Destination Unreachable[9] und Time Exceeded[10]. *Invoking Packet* enthält das Paket, welches den Fehler ausgelöst hat, und ermöglicht somit eine Zuordnung. Mittels Time Exceeded informiert ein Router, dass das *HL* im IPv6-Header `0` erreicht hat, und deshalb verworfen wurde. Das verworfene Paket wird dann der Nachricht angehängt.
Mittels Destination Unreachable kann ein Router den Sender informieren, dass das Ziel (unabhängig vom HL) nicht erreicht werden kann. Auslöser hierfür können z. B. fehlende Routen oder eine Firewall sein.
Für das *Invoking Packet* muss berücksichtigt werden, dass sich *Hop Limit*, *Traffic Class* und *Flow Label* im Vergleich zum gesendeten Paket verändern können.

3 Ausgabe

Der Tester versucht die Ausgabe des Programms mit der Netzwerkaktivität zu synchronisieren. Deshalb ist es essentiell, dass das Ausgabeformat eingehalten wird. Die Ausgabe soll auf *stdout* erfolgen, für etwaigen Debug Output muss *stderr* verwendet werden. Die IP-Adressen etc. sollen direkt ausgegeben werden, bevor das nächste Paket versendet wird.
Bevor die IP Adressen der Hops ausgegeben werden, soll am Beginn der Zeile, noch bevor ein Paket mit diesem Hop Limit versendet wird, das aktuelle Hop Limit ausgegeben werden. Für jeden Versuch wird dann danach mit 2(!) Leerzeichen getrennt die IP-Addresse bzw. der Timeout-Indikator `*` (Zeile 5) ausgegeben. Diese Ausgabe muss erfolgen, bevor das nächste Paket versendet wird. Wenn das Hop Limit inkrementiert wird, wird die aktuelle Zeile mit einem Zeilenumbruch (`\n`) abgeschlossen.
Der Empfang eines *Destination Unreachable* soll durch `!X` an der IP-Adresse signalisiert werden: `2001:4ba0:dead:fefe::2!X`.

```
1 $ ./trace6 2001:4ba0:ffec:193::
2 1  2a00:4700:0:3::1  2a00:4700:0:3::1  2a00:4700:0:3::1
3 <snip>
```

[6] https://tools.ietf.org/html/rfc4443#section-2.1
[7] https://tools.ietf.org/html/rfc4443#section-4.1
[8] https://tools.ietf.org/html/rfc4443#section-4.2
[9] https://tools.ietf.org/html/rfc4443#section-3.1
[10] https://tools.ietf.org/html/rfc4443#section-3.3

```
8  2001:1900:5:2:2::21b  2001:1900:5:2:2::21b  2001:1900:5:2:2::21b
9  *  *  *
10  2001:4ba0:dead:fefe::2  2001:4ba0:dead:fefe::2  2001:4ba0:dead:fefe::2
11  2001:4ba0:ffec:0193::  2001:4ba0:ffec:0193::  2001:4ba0:ffec:0193::
```

Sollten Sie auf *stderr* zusätzliche Debug-Ausgaben haben, können diese, zur Verbesserung der Lesbarkeit, über Output Redirection umgeleitet und ignoriert werden. Hierzu muss an den Programmaufruf `2>/dev/null` angehängt werden: `./trace6 2001:4ba0:ffec:0193:: 2>/dev/null`. Zusätzlicher Output auf *stderr* hat keinen Einfluss auf den Tester. Einzig der Output auf *stdout* wird berücksichtigt. Bitte überprüfen Sie, dass diese Output den gegeben Anforderungen (Format und Synchronisation mit dem Netzwerk) entspricht.

4 Benutzung der Programmierumgebung

Nach dem Auschecken ihres SVN-Repositories erhalten Sie nun die folgende Ordnerstruktur.

```
assignment3
|-- C
|   |-- libraw
|   |   |-- include
|   |   |   |-- checksums.h
|   |   |   |-- hexdump.h
|   |   |   '-- raw.h
|   |   |-- Makefile
|   |   '-- src
|   |       |-- checksum.c
|   |       |-- hexdump.c
|   |       |-- raw.c
|   |       '-- timespec.h
|   |-- Makefile
|   '-- src
|       |-- assignment3.c
|       |-- traceroute.c
|       '-- traceroute.h
'-- java
    |-- deps
    |   |-- GRNVS_RAW.c
    |   |-- GRNVS_RAW.h
    |   |-- libraw
    |   |   |-- include
    |   |   |   |-- checksums.h
    |   |   |   |-- hexdump.h
    |   |   |   '-- raw.h
    |   |   |-- Makefile
    |   |   '-- src
    |   |       |-- checksum.c
    |   |       |-- hexdump.c
    |   |       |-- raw.c
    |   |       '-- timespec.h
    |   |-- manifest.txt
    |   '-- run
    |-- Makefile
    '-- src
        |-- Arguments.java
        |-- Assignment3.java
        |-- GRNVS_RAW.java
        '-- Timeout.java
```

In den Unterordnern `C` und `Java` finden sich die Umgebungen, die Sie für die Bearbeitung der Aufgabe benötigen. Entscheiden Sie sich für *eine* Programmiersprache und kopieren Sie ***den Inhalt eines dieser Ordner*** in ihr Arbeitsverzeichnis:

`/users/<LRZ-Kennung>/assignment3/abgabe/`

Es sollten sich danach die Ordner `src, deps` und die Datei `Makefile` in `abgabe` befinden. Nachdem das gewünschte Rahmenprogramm nach `abgabe` kopiert wurde, muss dort nun `src/assignment3.c` bzw. `src/Assignment3.java` bearbeitet werden.

Wichtig: Die Verzeichnisstruktur muss exakt eingehalten werden. Lediglich Quelltext im Abgabeverzeichnis wird bewertet.

4.1 Ausführung

Nachdem im `abgabe` Ordner `make` ausgeführt wurde, finden Sie nun die ausführbare Binary *trace6* in Ihrem Verzeichnis. Diese kann im Ordner `abgabe` mit dem Befehl `./trace6 <target addr>` ausgeführt werden.
Hierzu wird eine Ziel-IP-Adresse benötigt. Diese kann z. B. über das Tool `host` abgefragt werden. Das `-t AAAA` gibt an, dass die IPv6-Adresse per DNS aufgelöst werden soll.

```
$ host -t AAAA traceroute.grnvs.net
traceroute.grnvs.net has IPv6 address 2001:4ba0:ffec:0193::
```

Die vollen Parameter sind `./trace6 -i <network interface> -t <probe timeout in sec> -q <attempts> -m <max hops> <target addr>`. Der *timeout*, welcher als Ganzzahl zu übergeben ist, spezifiziert die Anzahl an Sekunden, die auf den Empfang einer der obigen ICMPv6-Antworten gewartet werden soll. Das Argument *attempts* gibt die Anzahl an Versuchen, einen Hop zu erreichen, als Ganzzahl an. Die *max hops* geben eine Obergrenze des Hop Limit an, bis zu der Echo Requests versendet werden.

Ohne zusätzliche Parameter wird durch die Rahmenprogramme das Netzwerkinterface `eth0` verwendet. Soll Traceroute über ein anderes Interface ausgeführt werden, so kann dieses über den Switch `-i` angegeben werden.

4.2 Abgabetests

Wenn Sie der Meinung sind, dass Ihr Programm die Anforderungen erfüllt, können sie die Funktionalität durch die bereitgestellte Testumgebung überprüfen lassen. Im Ordner `assignment3` befindet sich seit dem letzten SVN Update die Datei `test-`

`bitte.txt`. Um einen Test anzufordern, verändern Sie diese Datei und committen sie in das SVN Repository. Führen Sie die Testumgebung erst aus, nachdem Sie das Programm selbst ausreichend getestet haben.

Um die korrekte Ausführung des Testers zu gewährleisten, dürfen die Ordnerstrukturen sowie die Makefile nicht verändert werden. Geben Sie außerdem Logging-/Debugging-Output nur auf **stderr** aus (`fprintf(stderr, ...)` bzw. `System.err.print(...)`) und stellen Sie sicher, dass auf **stdout** nur das erwartete Ergebnis ausgebeben wird.

Sie können nur einen Test alle 6 Stunden ausführen lassen. Falls Sie vor Ablauf der 6 Stunden einen zweiten Test beantragen, wird dieser verzögert, bis die 6 Stunden vergangen sind. Die Testumgebung wird nach dem Commit die Ergebnisse des Tests in den Ordner `results/<revision nr>` schreiben, welchen Sie nach einem `svn update` erhalten.

Status	Beschreibung
Success	Die getestete Abgabe hat den Test erfolgreich bestanden.
Failed	Die getestete Abgabe hat den Test nicht bestanden.
Precondition failed	Der Test wurde nicht ausgeführt, da ein hierfür vorhergehender notwendiger Test nicht erfolgreich war.

Tabelle 1: Beschreibung der Test Status Codes

Testcase	Beschreibung
Launchable	trace6 wurde kompiliert und kann gestartet werden.
Sends data on interface	Das Programm sendet (irgendwelche) Daten an das, per `-i <interface>` angegebene, Interface.
Rejects invalid address	Ungültige IPv6-Adressen werden erkannt, das Programm terminiert direkt.
Sends correct IPv6 header	Das gesendete Paket hat den erwarteten IPv6-Header.
Test Sends correct ICMPv6 message	Das gesendete Paket hat die erwartete ICMPv6 Payload.
Sends correct Echo Request	Es wird der erwartete Echo Request gesendet.
Timeout without answer	Das Programm erkennt Timeouts.
Supports timeout switch	Das Programm unterstützt die `-t <timeout in sec>` Option.
Performs 3 attempts by default	Das Programm führt 3 Versuche je Hop durch.
Supports attempts switch	Das Programm unterstützt die `-q <attempts>` Option.
Tries 15 hops by default	Das Programm versucht ohne Option 15 Hops.
Supports max hops switch	Das Programm unterstützt die `-m <max hops>` Option.
Increments sequence number	Das Programm inkrementiert die Sequenznummer und hält den Identifier im Echo Request konstant.
Accepts Time Exceeded	Das Programm erkennt Time Exceeded Nachrichten.
Rejects invalid IPv6 header	Das Programm überprüft den IPv6 Header der Antworten.
Ignores Flow Label and Traffic Class	Das Programm akzeptiert beliebige Flow Label und Traffic Classes.
Handles IPv6 Extension Headers	Das Programm kann mit den geforderten IPv6 Extension Headers umgehen.
Detects ICMPv6 checksum mismatch	Das Programm überprüft die ICMP Checksum.
Reject invalid Time Exceeded	Das Programm verwirft „falsche“ Time Exceeded Nachrichten.
Accepts fuzzy Time Exceeded	Das Programm ignoriert unnötige Felder in Time Exceeded.
Accepts Echo Reply	Das Programm erkennt Echo Replies.
Rejects invalid Echo Reply	Das Programm verwirft „falsche“ Echo Reply Nachrichten.
Rejects Echo Reply from other host	Das Programm überprüft den Absender der Echo Reply Nachrichten.
Accepts Destination Unreachable	Das Programm erkennt Destination Unreachable Nachrichten.
Rejects invalid Destination Unreachable	Das Programm verwirft „falsche“ Destination Unreachable Nachrichten.
Accepts fuzzy Destination Unreachable	Das Programm ignoriert unnötige Felder in Destination Unreachable.
Timeout handling after Time Exceeded	Timeout funktioniert auch nach dem ersten Time Exceeded noch.
Resets timeout handling	Timeout Handling wird neu initialisiert.
Handles multiple IPs on single hop	Das Programm kann mit mehreren IP Adressen auf einem Hop umgehen.
Terminates after receiving Echo Reply	Das Programm terminiert nachdem ein Echo Reply empfangen wurde.
Terminates after receiving Destination Unreachable	Das Programm terminiert nachdem ein Destination Unreachable empfangen wurde.

Tabelle 2: Beschreibung der Abgabetests

4.3 Hinweise, falls Sie nicht die Rahmenprogramme verwenden

Es ist möglich, die Aufgaben in einer Sprache Ihrer Wahl zu lösen. Es gibt ein paar Dinge, die dabei aber beachtet werden müssen:

- Der Arbeitsaufwand muss vergleichbar sein mit dem der Rahmenprogramme (magic.do_tracert()) währe **nicht** ok. Falls Sie sich unsicher sind, fragen Sie die Übungsleitung.
- Die ausführbare Datei **muss** *trace6* heißen.
- Die ausführbare Datei **muss** die Parameter der Rahmenprogramme unterstützen: `./trace6 -i <network interface> -t <probe timeout in sec> -q <attempts> -m <max hops> <target addr>`
- Es **muss** eine Makefile existieren. Falls Ihre Lösung nicht kompiliert werden muss, da Sie z.B. eine Scriptsprache verwendet haben, muss dennoch eine Dummy-Makefile vorhanden sein, welche ggf. einfach nichts tut.
- Um Pakete vor dem Test zu installieren müssen die Paketnamen in einer Datei *deps.txt* in dem Ordner *abgabe* stehen.
- Pakete dürfen nur aus Debian Jessie *main* und *contrib* installiert werden.
- Die Ausgabe muss dem Format in Abschnitt 3 entsprechen. Zur Synchronisation von Ausgabe und Netzwerkaktivität kann es notwendig sein, dass *stdout* nach den *print* Aufrufen geflushed wird.
- Sollte die übergebene Adresse keine IPv6-Adresse sein, so soll das Programm ohne Ausgabe auf stdout direkt terminieren.
- Weitere Nachrichten, Logs, etc. sind **nicht** auf *stdout* sondern auf *stderr* auszugeben.