HAW HAMBURG

GRUNDSEMINAR
Isabell Egloff
Matrikelnummer: 2655897

Die Erweiterung des reaktiven Netzwerk-Teleskops Spoki für das Messen von Scan-Aktivitäten im IPv6-Adressraum

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Betreuung durch: Prof. Dr. Thomas C. Schmidt / Raphael Hiesgen
Eingereicht am: 17.04.2023

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Inhaltsverzeichnis

1 Einleitung

Internetweites Scannen [1] ist eine Technik, die zur Untersuchung und Vermessung des Internets eingesetzt wird. Diese Technik wird häufig dafür verwendet, Hosts oder Dienste zu finden, die für bekannte Schwachstellen anfällig sind. Dabei werden unterschiedliche Ziele verfolgt. Neben den Angriffen durch Scanning-Methoden werden mit dieser Technik auch mögliche Opfer ausfindig gemacht, um sie zukünftig vor diesen Angriffen zu schützen [1].
Das bestehende Werkzeug Spoki [2] ist ein reaktives Netzwerk-Teleskop, das bereits das Scanverhalten im IPv4-Adressraum untersucht. Zukünftig soll es möglich sein, dass Spoki neben den IPv4-Paketen auch IPv6-Pakete erkennen und verarbeiten kann. Um dies zu ermöglichen, erfolgt eine Bearbeitung des Werkzeugs Spoki. Außerdem wird es im Nachgang auf fehlerhaftes Verhalten getestet und es wird die Leistung des Werkzeugs gemessen.
Diese Arbeit ist wie folgt gegliedert. In Abschnitt 2 wird erläutert, wie Spoki funktioniert. Daraus wird deutlich, welche Anpassungen durchgeführt werden müssen, um Spoki so zu erweitern, dass IPv6-Scans erfasst und verarbeitet werden können. In Abschnitt 3 werden die Probleme aufgezeigt, die beim Scannen und Messen im IPv6-Adressraum auftreten. Die Implementierung, die für die erweiterte Messeigenschaft von Spoki benötigt wird, wird im Abschnitt 4 erklärt. Außerdem wird Spoki auf die neuen und bisherigen Funktionen getestet. Die Testdurchführung und die Ergebnisse werden ebenfalls im Abschnitt 4 erläutert. Im Abschnitt 5 wird die Leistungsmessung von Spoki beschrieben. In Abschnitt 6 wird die Vorbereitung des Einsatzes von Spoki wiedergegeben und in Abschnitt 7 ist eine Zusammenfassung und ein Ausblick auf zukünftige Arbeiten formuliert.

2 Spoki: Das reaktive Netzwerk-Teleskop

Raphael Hiesgen et al. [2] haben Spoki entwickelt. Spoki ist ein reaktives Netzwerk-Teleskop, das auf TCP-SYN-Pakete reagiert und an TCP-Handshakes teilnimmt. Somit kann es unter anderem den Scan-Verkehr im IPv4-Adressraum aufzeichnen. Um die folgenden Erläuterungen besser nachvollziehen zu können, sollten die Kernfunktionen von Spoki vorgestellt werden. Daher wird ein Überblick über die Architektur des Netzwerk-Teleskops gegeben. Die Architektur kann in vier Komponenten aufgeteilt werden. Diese Komponenten werden Ingestion, Core, Logging und Probing genannt.

In der Komponente Ingestion werden Pakete vom Interface eingelesen und in eine datenrepräsentative Form umgewandelt. In der Komponente Core wird entschieden, wie Spoki mit den eingehenden Paketen umgeht. Die Pakete werden dabei entweder verworfen oder es wird mit ihnen weitergearbeitet. Relevant sind aktuell nur TCP-SYN-Pakete und TCP-ACK-Pakete aus dem IPv4-Adressraum. Der Rest der ankommenden Pakete wird ausgefiltert. Bisher werden noch alle IPv6-Pakete verworfen. Daraus lässt sich schließen, dass eine Bearbeitung dieses Prozesses erforderlich ist, um auch TCP-SYN-Pakete in IPv6 verarbeiten zu können. In der Komponente Logging werden diese Pakete und Attribute, die Auskunft über das Verhalten des Gesprächspartners geben, für den weiteren Gebrauch gespeichert. Wichtig dabei sind die Header-Felder, um Irregularitäten zu erkennen und die Nutzdaten, um mehr über die Absichten des Gesprächspartners zu lernen. Die Pakete werden an einen Pool von Shards weitergeleitet. Die Shards entscheiden dabei, welche Ziele geprüft werden sollen und leiten die Daten an den Broker weiter. Der Broker ist Teil der Probing Komponente. Er stellt eine Verbindung zu Scamper her. Scamper ist eine Open-Source-Anwendung, die für das Versenden von Probes zuständig ist. Diese werden von der Gegenseite als gültige Antworten akzeptiert. Daraus lässt sich schließen, dass neben der Erkennung von relevanten IPv6-Paketen auch eine Implementierung für die Unterscheidung von IPv4 und IPv6 notwendig ist. Darüber hinaus müssen IPv4- und IPv6-Pakete verarbeitet werden können und es müssen korrekte Antwortpakete versendet werden. Als Nächstes ist es wichtig festzustellen, welche Probleme sich daraus ableiten, um eine möglichst fehlerfreie Implementierung zu gewährleisten.

3 Problemstellung

Zakir Durumeric et al. [3] entwickelten mit ZMap einen Netzwerkscanner der in der Lage ist den IPv4-Adressbereich in weniger als 45 Minuten vollständig zu scannen. Im IPv6-Bereich ist das Scannen eine viel anspruchsvollere Aufgabe, da der Adressbereich so groß ist, dass er nicht vollständig durchscannt werden kann. Das Problem ist, dass Spoki erst auf Reaktionen von außerhalb warten muss, um diese Ereignisse erfassen zu können. Während viele IPv4-Scans den ganzen Adressbereich durchlaufen sind IPv6-Scans auf Teil-Adressbereiche beschränkt. Aus diesem Grund können IPv6-Scans auch nicht immer erfasst werden. Mit Spoki wurde bereits der Scan-Verkehr im IPv4-Adressraum untersucht. Eine Erweiterung des Werkzeuges ist daher erforderlich, um auch Scan-Verkehr im IPv6-Adressraum zu untersuchen. Dafür muss zusätzlich zur Implementierung der Erweiterung getestet werden, ob Spoki die bisherigen und neuen Funktionen korrekt ausführt.

Spoki ist so konzipiert, dass es in Echtzeit TCP-Anfragen beantworten kann [2], da die Gegenseite die Verbindung bei einer zu langen Antwortverzögerungen abbrechen würde. Brauchen die Antworten länger als 5 Sekunden, gehen über die Hälfte der kompletten Handshakes verloren, da die Scanner bei einer Zeitüberschreitung RSTs statt ACKs senden. Aus diesem Grund ist es wichtig, dass es durch die Erweiterung nicht zu hohen Leistungseinbußen bei einem Einsatz von Spoki kommt, damit alle ankommenden TCP-Anfragen in Echtzeit verarbeitet werden können und somit keine Daten verloren gehen.

4 Messeigenschaften von Spoki auf IPv6 erweitern

Bis zu diesem Zeitpunkt wird Spoki ausschließlich für das Observieren von Scans im IPv4-Adressraum verwendet. Zukünftig soll es auch möglich sein, Scan-Aktivitäten im IPv6-Adressraum beobachten zu können. Dabei ist es notwendig, dass TCP-Anfragen von IPv6-Paketen nicht verworfen werden, dass Spoki IPv6- von IPv4-Paketen unterscheiden kann, diese dementsprechend verarbeitet und anschließend korrekte Antworten verschickt. Die Implementierung der neuen Funktionen von Spoki wird im folgenden Abschnitt erläutert. In dem darauffolgenden Abschnitt werden Tests beschrieben, die durchgeführt werden, um festzustellen, ob Spoki die notwendigen Anforderungen erfüllt. Schließlich wird noch getestet, wie Spoki auf inkorrekte Pakete reagiert und was passiert, wenn mehr Pakete empfangen werden, als verarbeitet werden können.

4.1 Implementierung

Spoki verwendet für eine möglichst effektive Verarbeitung von Paketen das C++ Actor Framework (CAF) [4]. Bisher wurde nur die IPv4-Adressimplementierung von CAF verwendet. Damit zukünftig auch noch TCP-Pakete aus dem IPv6-Adressraum verarbeitet werden können, wird die Adressimplementierung von CAF für IPv4 und IPv6 im Programmcode integriert. Somit ist es möglich, IPv4- von IPv6-Adressen unterscheiden zu können, um mit den jeweiligen TCP-Paketinformationen weiter zu arbeiten.
Um IPv6-Pakete erkennen und verarbeiten zu können, wird ein Input Parameter implementiert. Dieser wird network6 genannt. Ein solcher Input Parameter wird bereits für IPv4 verwendet und nennt sich network. Die Implementierung für IPv6 orientiert sich an der IPv4-Implementierung und wird so angepasst, dass Spoki IPv4 von IPv6 unterscheiden kann. Ein Beispiel dazu ist der Abbildung 1 zu entnehmen.

```
...
case 0x86DD: {
  ipv6_packets += 1;
  auto ip_hdr = reinterpret_cast<libtrace_ip6_t*>(layer3);
  auto ts = spoki::to_time_point(trace_get_timeval(pkt));

  caf::ipv6_address saddr;
  caf::ipv6_address daddr;

  //Kopieren der Bytes in die IPv6-Adress-Struktur
  std::memcpy(saddr.bytes().data(), ip_hdr->ip_src.s6_addr, caf::ipv6_address
    ::num_bytes);
  std::memcpy(daddr.bytes().data(), ip_hdr->ip_dst.s6_addr, caf::ipv6_address
    ::num_bytes);

  //Abfrage, ob eingehende Pakete in das konfig. Präfix geschickt werden
  if (network6.contains(saddr)) {
    std::cerr << "source is in our prefix\n";
    return;
  }

  //Abfrage, ob eingehende Pakete nicht aus dem konfigurierten Präfix kommen
  if (!network6.contains(daddr)) {
    std::cerr << "destination is not in our prefix\n";
    return;
  }

  auto proto = extract_protocol(pkt);
  if (!proto) {
    std::cerr << "couldn't identify protocol\n";
    break;
  }
  ...
}
...
```

Abbildung 1: Verarbeitung eines neuen IPv6-Pakets für die Extraktion der Paketinformationen

Die Repräsentation einer IPv6-Adresse, die 128-Bit groß ist, lässt sich nicht so einfach abbilden wie eine IPv4-Adresse, die nur eine Größe von 32-Bit hat. Die 32-Bit lassen sich einfach auf einen Typen reproduzieren. Die Komplexität von einer IPv6-Adresse benötigt in diesem Fall einen Container-Typen. Die Bytes einer IPv6-Adresse werden daher nicht

in eine Variable kopiert, sondern in die IPv6-Adress-Struktur caf::ipv6_address. Dies ist in Abbildung 1 zu erkennen.

Damit zukünftig alle relevanten Informationen gespeichert werden können, wird in der Konfigurationsdatei noch der Pfad zu einem Ordner angegeben, der diese Informationen als Logdateien zukünftig speichern soll.

4.2 Testen der Funktionalität

Nachdem die Implementierung von IPv6 abgeschlossen ist und Spoki erfolgreich ausgeführt werden kann, muss überprüft werden, ob IPv6-Pakete erkannt, verarbeitet und beantwortet werden können. Eine wichtige Rolle bei der Verarbeitung der ankommenden Pakete spielt hierbei libtrace[1]. Dabei handelt es sich um eine Bibliothek, die zur Verarbeitung von Netzwerkverkehrserfassungen verwendet wird. Teile des Analysierens von Paketen werden von libtrace übernommen, die über eine Schnittstelle ausgelesen werden können.

Es werden Tests zur Überprüfung der IPv6-Funktionalität durchgeführt und es werden die einzelnen Komponenten auf Fehlverhalten getestet. Darüber hinaus wird überprüft, ob alle Komponenten zusammen korrekt arbeiten. Damit ein erfolgreicher Einsatz von Spoki stattfinden kann, müssen folgende Punkte erfüllt sein:

1. Die Shards bekommen die richtigen Pakete.
2. Der Probe Request wird richtig gebaut.
3. Spoki verschickt die richtigen Antwortpakete.

Treffen diese Aussagen zu, kann bestätigt werden, dass Spoki nun die gewünschten Funktionen erfüllt. Für die Tests werden pcap-Dateien erstellt, um SYN- und ACK-Pakete an Spoki zu schicken. Somit kann die Reaktion von Spoki auf die ankommenden Pakete überprüft werden. Mit dem Werkzeug Wireshark[2] kann Netzwerkverkehr mitgeschnitten werden. Aus diesem Mitschnitt können daraufhin pcap-Dateien für die Tests erstellt und an Spoki verschickt werden. Der Pfad zu der zu testenden pcap-Datei wird in der

[1] https://github.com/LibtraceTeam/libtrace
[2] https://www.wireshark.org/docs/wsug_html_chunked/

Konfigurationsdatei eingetragen, sodass Spoki beim Testen die vorgegebenen Pakete verwenden kann. Es wird kontrolliert, ob die SYN- und ACK-Pakete korrekt bei den Shards ankommen. Dafür werden diese im Programmcode an der entsprechenden Stelle ausgegeben und der Paketinhalt kann bei der Ausführung von Spoki überprüft werden. Mit den Ausgaben der Shard-Pakete und Probe-Requests kann nun festgestellt werden, dass die Shards die richtigen Pakete bekommen und die Probe-Requests richtig gebaut werden. Dieser Vorgang wird für IPv4-Adressen wiederholt, um zu überprüfen, ob die Modifikation des Codes Fehler in der vorherigen Implementierung auslösen. Dabei lassen sich jedoch keine Fehler feststellen. Die IPv4-Implementierung funktioniert weiterhin wie bisher.

Im nächsten Schritt wird getestet, ob Spoki die richtigen Antwortpakete verschickt. Es wird nun wieder ein SYN-Paket an Spoki verschickt und der Inhalt des Antwortpakets von Spoki wird wieder auf Korrektheit überprüft. Nachdem festgestellt werden kann, dass das Antwortpaket die richtigen Angaben enthält, kann nun auch bestätigt werden, dass Spoki korrekte Antwortpakete verschickt.

Schließlich wird noch einmal der vollständige Handshake untersucht. Es ist dafür nicht mehr notwendig, pcap-Dateien zu erstellen. Stattdessen wird nun auf einem ausgewählten Interface gelauscht. Um den Handshake verfolgen zu können, muss ein SYN den Handshake anstoßen. Dafür wird mit der hinzugefügten IPv6-Adresse im Interface eine Netzwerkverbindung mit Hilfe des Werkzeugs Netcat[3] initiiert. Netcat ist ein Standard-Unix-Utility, also ein Kommandozeilenwerkzeug. Es ist in diesem Fall sehr nützlich, da mit Netcat eine TCP-Verbindung neu initiiert werden kann, die Spoki daraufhin annimmt. Bei Überprüfung des Handshakes kann nachgewiesen werden, dass Spoki IPv6-Pakete erkennt, diese von IPv4-Paketen unterscheiden kann und korrekt verarbeitet. Außerdem kann bestätigt werden, dass zusätzlich korrekte Antwortpakete verschickt werden. Trotz der Modifikation bleiben die bisherigen Funktionen von Spoki erhalten.

4.3 Testen der Robustheit

Um die Robustheit zu testen, wird hierzu die Reaktion von Spoki auf syntaktisch fehlerhafte Pakete untersucht und es wird außerdem überprüft, was passiert, wenn bei Spoki mehr Pakete ankommen, als verarbeitet werden können.

[3] https://www.commandlinux.com/man-page/man1/netcat.1.html

Als Erstes wird ein SYN an Spoki verschickt, dass die Version vier statt sechs eingetragen bekommt, obwohl es sich um ein IPv6-Paket handelt. Anschließend verarbeitet Spoki dieses Paket als IPv6-Paket und sendet ein SYN-ACK mit der Angabe Version sechs zurück. Daher wird festgestellt, dass libtrace das Paket trotzdem als Version sechs anerkennt und es dementsprechend von Spoki verarbeitet werden kann.

Es werden mehrere Pakete mit unterschiedlichen Angaben der IP-Payload-Länge verschickt. Dabei wird getestet, wie Spoki reagiert, wenn die Angabe der Payload-Länge größer ist als die tatsächliche Länge der Nutzlast. Es fällt auf, dass diese Angaben nicht vorher von libtrace oder Spoki auf Korrektheit überprüft werden. Die Pakete werden angenommen, verarbeitet und es werden Antwortpakete zurückgeschickt. Ein Problem dabei ist, dass Spoki die Nutzlast eines empfangenen Pakets kopiert, um diese zu speichern. Dabei wird auch die Länge der Nutzlast ausgelesen. Wird im Header angegeben, dass die Nutzlast beispielsweise viel größer ist, kann das zu Problemen führen, da es sich um zwei unterschiedliche Angaben handelt. Voraussichtlich könnte es auch passieren, dass Spoki dabei abstürzt.

Um auf anderem Wege syntaktisch fehlerhafte Pakete zu erstellen, werden inkorrekte Werte testweise in TCP-Header-Felder [5] eingetragen. Mit fehlerhaften Angaben der Empfangsfenstergröße und der Prüfsumme wird getestet, wie Spoki auf die Pakete reagiert.

Um die Reaktion auf fehlerhafte Angaben der Empfangsfenstergröße zu testen, werden hierzu inkorrekte Einträge im TCP-Header vorgenommen. Die Pakete werden wieder von Spoki angenommen und es werden Antwortpakete zurückgeschickt. Die Empfangsfenstergröße [6] gibt die Menge der Daten an, die der Sender des Pakets beim Empfang noch zwischenspeichern kann. Damit gibt sie die maximale Datenmenge an, die empfangen werden kann, bevor es zu einem Pufferüberlauf kommt und weiter eingehende Pakete verworfen werden müssen. Bei einer zu kleinen Größe könnte es aber wiederum zu Verzögerungen kommen. Bei einer syntaktisch fehlerhaften Eingabe könnte damit der Nachrichtenaustausch gestört werden und wichtige Daten könnten dadurch verloren gehen.

Die Prüfsumme [7] wird über den TCP-Header, die Daten und einen Pseudo-Header berechnet und wird verwendet, um Übertragungsfehler zu erkennen. Gibt es eine Abweichung der Angabe mit der tatsächlich berechneten Prüfsumme, sollte das Paket im

Normalfall verworfen werden. Mit einer fehlerhaften Angabe der Prüfsumme wird getestet, wie Spoki auf ein solches Paket reagiert. In diesem Fall wird als falsche Angabe 12345 gewählt. Spoki nimmt das Paket an und antwortet mit einem SYN-ACK. Das SYN-ACK würde im Normalfall mit Angabe der fehlerhaften Prüfsumme bei Ankunft verworfen werden.

Es wird außerdem noch getestet, wie Spoki auf Pakete reagiert, die mehrere Flags gesetzt haben, die in der Kombination fehlerhaft sind. Das zu testende Paket hat die Flags SYN, ACK und FIN gesetzt. Spoki überprüft die gesetzten Flags der ankommenden Pakete. Alle Pakete, die nicht ausschließlich das SYN-Flag oder das ACK-Flag gesetzt haben, werden ignoriert. Beim Testen kann diese Aussage bestätigt werden, da Spoki das Paket mit den drei gesetzten Flags ignoriert und verwirft.

Mit der Möglichkeit, IPv6-Pakete zu verarbeiten, können nun auch Pakete mit Erweiterungsheadern [8] bei Spoki ankommen. Um die Reaktion auf Pakete mit Erweiterungsheadern zu testen, wird ein SYN-Paket mit einem Hop-by-Hop-Option-Header, einem Destination-Option-Header und einem Routing-Header an Spoki verschickt. Das Paket wird von Spoki als SYN erkannt und es wird ein SYN-ACK zurückgeschickt. Da keine Implementierung für Erweiterungsheader integriert wurde, werden diese nicht verarbeitet und werden ignoriert. Das Paket wird wie ein SYN ohne Erweiterungsheader behandelt.

In einem weiteren Test wird für einen Zielport der Wert null angegeben und das Paket wird anschließend an Spoki verschickt. Daraufhin wird das Paket angenommen und es wird ein SYN-ACK mit der Angabe null als Quellport zurückgeschickt. Dieser Port ist reserviert und sollte nicht verwendet werden dürfen. Es handelt sich hierbei wieder um inkorrekte Paketangaben, die Spoki nicht herausfiltert. Die Pakete werden daher standardmäßig verarbeitet.

Schließlich ist es noch wichtig zu wissen, wie Spoki reagiert, wenn mehr Pakete ankommen, als Spoki selber verarbeiten kann. Aus den bisherigen Messungen ist bekannt, dass die Komponente Ingestion 250.000 Pakete pro Sekunde verarbeiten kann, ohne dass dabei mehrere Threads verwendet werden. Werden jedoch mehr Pakete von Spoki erfasst, kann nur noch ein geringer Teil davon in Echtzeit verarbeitet werden. Werden von der Komponente Ingestion 270.000 Pakete pro Sekunde angenommen, können höchstens 250.000 Pakete pro Sekunde verarbeitet werden. Die Pakete werden daher nicht mehr alle in Echtzeit verarbeitet. Werden mehr Pakete an Spoki verschickt als verarbeitet werden

können, stauen sich die Pakete und dies führt zu Verzögerungen bei der Verarbeitung. Eine weitere Erhöhung der Paketrate verlängert zusätzlich noch die Dauer der Verarbeitung.

Nach Durchführung der Tests kann festgestellt werden, dass Spoki viele syntaktisch fehlerhafte Pakete annimmt, verarbeitet und beantwortet. Dies kann unterschiedliche Auswirkungen haben. Es könnte passieren, dass Pakete im Anschluss verloren gehen, dass Pakete nicht in Echtzeit verschickt werden können oder es könnte passieren, dass Spoki abstürzt. Aus diesem Grund könnte es vom Vorteil sein, nach offensichtlich falschen Paketen zu filtern und sie zu verwerfen. Es könnte aber auch Nachteile mit sich ziehen, wenn solche Pakete verworfen werden. Einzelne Fälle könnten sich zukünftig noch als interessantes Beobachtungsziel herausstellen. Das Herausfiltern solcher Pakete könnte daher Folgen haben. Es wäre auch möglich, diese zu protokollieren oder nicht auf diese zu reagieren.

5 Messen der Leistung

Das Messen der Leistung sollte vor dem Einsatz von Spoki nicht vernachlässigt werden, denn damit wird die Leistungsevaluation für die Einsatzfähigkeit bestimmt. Dabei wird überprüft, wie viele Pakete Spoki verarbeiten kann. Damit festgestellt werden kann, an welcher Stelle es zu Leistungseinbußen kommt, werden die Komponenten von Spoki einzeln gemessen. Dafür werden Ethernet-Interfaces angelegt, mit denen mit Hilfe von ZMap Pakete an Spoki geschickt werden. Gemessen werden daraufhin die Pakete, die von Spoki verarbeitet werden können. Dafür werden die Raten der Pakete pro Sekunde für jede Messung immer weiter erhöht, bis die Anzahl der Paketrate nicht mehr verarbeitet werden kann. Für einen besseren Überblick der erwarteten Leistung werden die Ergebnisse der Messungen vor der Implementierung des IPv6-Supports für einen Vergleich hinzugezogen.

5.1 Vorbereitung der Leistungsmessungen

Zur Vorbereitung wird die Umgebung für die Leistungsmessungen angepasst. Für die Messungen werden IPv6-Adressen benötigt, um Nachrichtenverkehr mit ZMap zu generieren. Damit es zu keinen Konflikten mit bereits verwendeten oder reservierten Adressen

kommt, werden privat einsetzbare und einmalige IPv6-Adressen unter anderem aus dem RFC 4193 [9] recherchiert. Somit wird verhindert, dass die Quell- und die Zieladresse der TCP-Verbindung bei der Messung identisch sind.

ZMap wird benötigt, um die TCP-SYN-Pakete zu generieren, die an Spoki verschickt werden. Für die Leistungsmessungen muss ZMap nun auch in der Lage sein, IPv6-Pakete zu verschicken. Aus diesem Grund wird ein ZMap Projekt verwendet, das eine IPv6-Implementierung integriert und mit einem speziellen Modul SYN-Scans in IPv6 erzeugen kann[4]. Das benötigte Modul muss dabei nur im ZMap Aufruf angegeben werden. Es werden zwei virtuelle Ethernet-Interfaces angelegt, die beide freie Adressen als Schnittstelle benötigen und für den Nachrichtenaustausch zwischen der Paketquelle und Spoki verwendet werden. In der Konfigurationsdatei sollte nun noch das passende Interface verwendet werden, welches eines der virtuellen Ethernet-Interfaces ist. Außerdem muss noch das Präfix, das beim Messen angezielt wird, in der Konfigurationsdatei angegeben werden. Es wird darauf geachtet, die Messungen unter möglichst ähnlichen Bedingungen durchzuführen, wie die Messungen vor der Implementierung des IPv6-Supports. Somit können genauere Aussagen über die Veränderung der Leistung von Spoki getroffen werden.

5.2 Durchführung und Ergebnisse

Nun werden die Leistungen der einzelnen Komponenten Ingestion, Core und Logging gemessen. Daraufhin wird ZMap gestartet. Neben der Angabe der Paketrate pro Sekunde wird außerdem die Mac-Adresse der Zielschnittstelle und eine IP-Adresse als Quelle angegeben. Beim Ausführen von ZMap wird nun die Angabe der Paketrate immer weiter erhöht und es wird überprüft, wie viele Pakete von Spoki verarbeitet werden können. Sobald die Paketrate nicht mehr vollständig verarbeitet werden kann, werden die Prozesse durch die Angabe von Threads parallelisiert und es wird erneut die Verarbeitungsrate überprüft. Bei Angabe der Threads wird darauf geachtet, dass Spoki die Verarbeitungsrate hochskalieren kann. Das bedeutet, die Paketrate, die ein Thread vollständig verarbeiten kann, können auch zwei Threads gleichzeitig erledigen. Die Leistung eines Threads wird bei zwei parallelen Prozessen daher verdoppelt.

Es wird erst mit der Komponente Ingestion gestartet, danach folgt die Komponente Core und schließlich wird noch die Leistung der Komponente Logging gemessen. Das

[4] https://github.com/tumi8/zmap

erste Ziel für Ingestion und Core, das sich aus den Messungen mit einem Thread vor der Implementierung des IPv6-Supports erschließt, ist die Verarbeitung von 250.000 Paketen pro Sekunde. Danach wird bis zu einer Million Paketen pro Sekunde hochskaliert, die in den beiden Messungen mit 4 Threads verarbeitet werden können. Bei der Messung von der Logging Komponente ist zu beachten, dass die Shards zusätzlich belastet werden, da sie noch Pakete versenden müssen. Es lässt sich daher feststellen, dass durch die Mehrarbeit nicht mehr so viele Pakete verarbeitet werden können. Die Anzahl der Threads hat sich nach Angabe der bisherigen Messungen verdoppelt. Die erwartete Leistung steht somit nun fest. Anschließend werden die Leistungsmessungen für den Einsatz im IPv6-Adressraum durchgeführt.

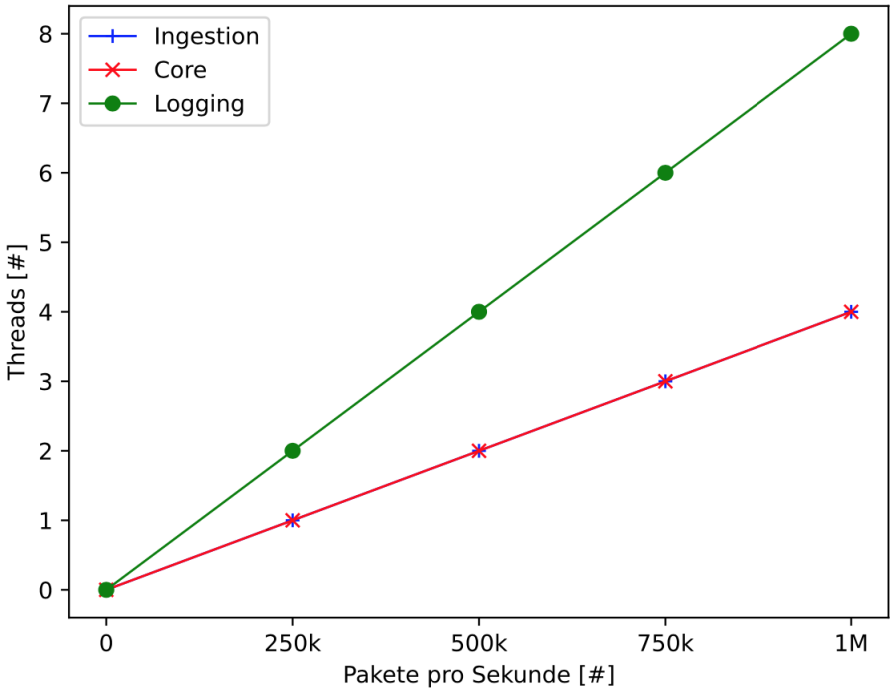

Abbildung 2: Die Messung der Leistung und Überprüfung der Skalierbarkeit von Spoki

Die Anzahl der verarbeiteten Pakete pro Sekunde sind auf der x-Achse dargestellt, während die benötigten Threads auf der y-Achse dargestellt werden.

Ingestion. Bei einer Rate von 260.000 Paketen pro Sekunde beginnt die Rate der verarbeitenden Pakete zu schwanken. Bei einer Menge von 270.000 Paketen schafft Spoki es nur noch etwa 250.000 Pakete zu verarbeiten. Das Ergebnis der Leistung von Spoki

bleibt daher mit einer Menge von 250.000 Paketen pro Sekunde mit der bisherigen Leistung konstant.
Mit zwei Threads ist Spoki in der Lage, doppelt so viele Pakete zu verarbeiten, dass bedeutet, es können 500.000 Pakete pro Sekunde verarbeitet werden. Diese Leistung kann auf größere Paketraten skalieren, sodass Spoki mit 4 Threads eine Million Pakete pro Sekunde verarbeiten kann. Die Ergebnisse von der Komponente Ingestion stimmen somit mit der bisherigen Leistung von Spoki überein.
Core. Bei dieser Messung schafft es Spoki ebenfalls 250.000 Pakete pro Sekunde zu verarbeiten. Es ist außerdem möglich, die Verarbeitung auf eine Million hoch zu skalieren mit der Verteilung auf vier Threads.
Logging. Beim Logging wird wie erwartet die doppelte Menge an Threads benötigt, um die Paketverarbeitung auf eine Million zu skalieren.
Durch die Erweiterung wird Spoki demnach nicht auffallend belastet und kann die gleiche Leistung wie zuvor erbringen. Das Ergebnis ist in der Abbildung 2 dargestellt.

6 Vorbereitung für den Einsatz von Spoki

Für den Einsatz von Spoki werden zwei /48 Präfixe speziell für ankommende IPv6-Pakete angelegt. Es handelt sich dabei um zwei große Präfixe, bei den viel Nachrichtenverkehr ankommen könnte. Jedoch ist es wahrscheinlich, dass auf einem neu angelegten Präfix noch nicht sehr viele Pakete ankommen werden. Es wird jeweils ein Netzwerkinterface für diese Präfixe angelegt. Dabei wird sich wieder an der Vorarbeit für IPv4 orientiert. Für die vorherige Implementierung wurden bisher drei Netzwerkinterfaces hinzugefügt. Momentan werden für IPv6 nur zwei Interfaces benötigt.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde das Netzwerk-Teleskop Spoki erweitert, um neben dem Scan-Verkehr von IPv4 auch Scan-Verkehr von IPv6 aufzeichnen zu können. Dafür wurde der Programmcode so angepasst, dass ankommende SYN-Pakete auch erkannt, verarbeitet und in Echtzeit beantwortet werden können, wenn es sich um IPv6-Pakete handelt. Die bisherigen und neu implementierten Funktionen von Spoki wurden dabei auf Korrektheit untersucht. Anschließend konnten Leistungsmessungen stattfinden. Für die Messungen wurden die Leistungen der einzelnen Komponenten Ingestion, Core und das Logging

gemessen. Dabei konnte bestätigt werden, dass Spoki durch die neue Implementierung keine Leistungseinbußen aufweist und zukünftig im realen Einsatz genutzt werden kann. Schließlich wurde noch damit begonnen, den Einsatz von Spoki vorzubereiten. Zukünftig soll Spoki dafür zuständig sein, Scanner im IPv6-Adressraum zu finden, um daraus später gegebenenfalls Aussagen über das IPv6-Scanverhalten treffen zu können.

Literatur

[1] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, (USA), p. 65–78, USENIX Association, 2014.

[2] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch, "Spoki: Unveiling a New Wave of Scanners through a Reactive Network Telescope," in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), pp. 431–448, USENIX Association, 2022.

[3] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-Wide Scanning and Its Security Applications," SEC'13, (USA), p. 605–620, USENIX Association, 2013.

[4] D. Charousset, R. Hiesgen, and T. C. Schmidt, "Revisiting actor programming in C++," *Computer Languages, Systems & Structures*, vol. 45, pp. 105–131, 2016.

[5] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, IETF, August 2022.

[6] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323, IETF, September 2014.

[7] J. Zweig and D. C. Partridge, "TCP alternate checksum options," RFC 1145, IETF, February 1990.

[8] B. E. Carpenter and S. Jiang, "Transmission and Processing of IPv6 Extension Headers," RFC 7045, IETF, December 2013.

[9] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," RFC 4193, IETF, October 2005.