

# MASTERARBEIT

Evaluierung und Vergleich eines emulierten WireGuard-Multihop-Netzwerkes mit  
Tor

DANIEL URBAN

Hochschule Hof  
Fakultät Informatik  
Matrikelnr.: 01569916  
Dozent: Prof. Dr. Florian Adamsky



# INHALTSVERZEICHNIS

1	EINLEITUNG	9
2	PROBLEMSTELLUNG	13
2.1	Onion-Routing	13
2.2	Tor	15
2.3	VPN	17
2.3.1	Kaskadierende VPNs	18
2.3.2	VPN-Onion-Routing	19
3	WIREGUARD	21
3.1	Kommunikation	21
3.2	Kryptographie und Sicherheit	22
4	VERSUCHSAUFBAU	25
4.1	Methodik und Verfahren	25
4.1.1	Technische Daten	25
4.1.2	Allgemeines Test-Setup	25
4.1.3	WireGuard vs. Tor vs. Multihop-Netzwerk	26
4.1.4	UDP vs. TCP	27
4.1.5	Gemessene Kennzahlen	27
4.1.6	Lokale Bedingungen vs. Internet-Bedingungen	28
4.1.7	Logging	28
4.2	Verwendete Technologien	29
4.3	WireGuard-Netzwerk	30
4.4	Tor-Netzwerk	32
4.5	Multihop-Netzwerk	34
5	AUSWERTUNG UND DISKUSSION DER ERGEBNISSE	37
5.1	Auswertung	37
5.1.1	Übertragungsrate	37
5.1.2	Goodput	39
5.1.3	Retransmits	41
5.1.4	Round-Trip-Time	43
5.1.5	Jitter	45
5.1.6	Paketverlust	47
5.2	Diskussion	49
6	VERWANDTE ARBEITEN	51
7	FAZIT UND ZUKÜNFTIGE ARBEIT	53
A	ANHANG	55
A.1	WireGuard-Interface-Konfiguration	55
A.1.1	Interfaces bei zwei Peers	55
	LITERATUR	56

## ABBILDUNGSVERZEICHNIS

---

Abbildung 1	Datenübertragung durch ein Onion-Routing-Netzwerk. Der Client verschlüsselt das Paket schichtweise mit den Schlüsseln der traversierten Onion-Router in umgekehrter Reihenfolge (Hier erst Router 3, dann 2, dann 1). Jeder Onion-Router entschlüsselt anschließend wieder eine Schicht. Der letzte Router leitet das Paket an das eigentliche Ziel weiter. . . . .	14
Abbildung 2	Bei einem kaskadierenden VPN werden mehrere VPN-Server hintereinander geschaltet. . .	19
Abbildung 3	Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 2 Peers. .	30
Abbildung 4	Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 3 Peers. .	31
Abbildung 5	Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 4 Peers. .	32
Abbildung 6	Topologie des Tor Netzwerkes mit Beispiel eines möglichen Circuits. . . . .	33
Abbildung 7	Topologie und Kommunikationsablauf des Multihop-Netzwerks ohne WireGuard mit drei Knoten. .	34
Abbildung 8	Vergleich der Übertragungsraten in MBit/Sekunde bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L)) . . . . .	38
Abbildung 9	Vergleich des Goodputs bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L)) . . . . .	40

Abbildung 10	Vergleich der Anzahl an Retransmits bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L))	42
Abbildung 11	Vergleich der Round-Trip-Time in Millisekunden bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L)) . . . . .	44
Abbildung 12	Vergleich des Jitters in Millisekunden bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L))	46
Abbildung 13	Vergleich der Paketverlustquote in Prozent bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D W Tor (Direkt (D), WireGuard (W) oder Tor), T U (TCP (T) oder UDP (U)), 2 3 4P (Anzahl der Peers), I L (Internet-Bedingungen (I) oder lokale Bedingungen (L))	48

## TABELLENVERZEICHNIS

---

Tabelle 1	Beispiel einer Cryptokey-Routing-Tabelle. . . .	22
Tabelle 2	Die gewählten TCP-Puffergrößen für das Senden und Empfangen von Daten über Netzwerk.	26
Tabelle 3	Die Parameter für das Gilbert-Elliot-Modell zur Emulation von Paketverlusten in einem simulierten Netzwerk. . . . .	28
Tabelle 4	Interface-Konfiguration von Interface wgo im Client bei zwei Peers. . . . .	55
Tabelle 5	Interface-Konfiguration von Interface wg-peer1 in Peer 1 bei zwei Peers. . . . .	55

## ABKÜRZUNGEN

---

CIA Confidentiality Integrity Availability

VPN Virtual Private Network

TOR The Onion Router

TCP Transmission Control Protocol

UDP User Datagram Protocol





## EINLEITUNG

---

Computer spielen in unserer modernen Gesellschaft eine essentielle Rolle, denn sie kommen mittlerweile in allen möglichen Anwendungsgebieten zum Einsatz. Manche dieser Anwendungsbereiche sind sehr sicherheitskritisch, weil sie z. B. Zugriff auf wichtige bzw. sensible Funktionen oder Daten gewähren. Unberechtigte oder böswillige Angreifer, in der Informatik auch *Black Hats* genannt, können sich dieser Funktionen oder Daten bemächtigen, um sie für ihre eigenen Zwecke zu missbrauchen. Gerade deshalb ist es umso wichtiger, dass IT-Systeme ausreichend geschützt werden. Zu diesem Zwecke existiert in der Informatik das Konzept der so genannten CIA-Triade. Es umfasst die drei wichtigste Grundwerte der Informationssicherheit: Vertraulichkeit (engl. : *confidentiality*) , Integrität (engl. : *integrity*) und Verfügbarkeit (engl. : *availability*) [27]:

**VERTRAULICHKEIT** Informationen sollten nur berechtigten Personen zur Verfügung stehen. Unberechtigter Zugriff soll verhindert werden.

**INTEGRITÄT** Informationen sollen unversehrt sein. Das heißt, sie dürfen nicht durch Unberechtigte verändert werden.

**VERFÜGBARKEIT** Informationen sollen Berechtigten innerhalb einer angemessenen Zeit zur Verfügung stehen.

Obwohl die Bewahrung dieser drei Hauptziele in der Informationssicherheit oberste Priorität hat, gibt es neben der CIA-Triade noch weitere Ziele, die ebenfalls einen wichtigen Stellenwert einnehmen. Hierbei handelt es sich um *Privatsphäre* und *Anonymität*. Wie wichtig diese Punkte sind und wie stark sie vor allem im privaten Bereich gefährdet sind, ist spätestens seit den Veröffentlichungen des Whistleblowers Edward Snowden rund um die so genannte „NSA-Affäre“ bekannt. Es kam dabei heraus, dass die US-Regierung im Zuge einer Massenüberwachung nicht nur persönliche Daten wie Anrufe, Nachrichten und E-Mails von Privatpersonen, sondern sogar von hochrangigen Politikern weltweit gesammelt hatte. [26]

Seit diesem Ereignis hat sich der Blick auf die Privatsphäre im Internet und den Schutz unserer Daten schlagartig verändert. Umso wichtiger werden Organisationen und Projekte, welche sich für mehr Privatsphäre und Datenschutz im Internet einsetzen. Ein Beispiel für eine solche Organisation ist der *Chaos Computer Club* <sup>1</sup> Dabei handelt

---

<sup>1</sup> <https://www.ccc.de/>

es sich um die größte europäische Hackervereinigung, welche sich für mehr Sicherheit in der Informationstechnik einsetzt, indem sie beispielsweise die Politik beraten, Kampagnen und Veranstaltungen organisieren oder auch selbst technische Forschung betreiben sowie Endanwendern Tipps geben und Verhaltensweisen nahelegen, mit denen man sich und seine Daten besser schützen kann.

Verschlüsselung kann zwar die Privatsphäre verbessern, jedoch nicht die Anonymität. Denn auch wenn eine Kommunikation verschlüsselt ist, ist immer noch ersichtlich, wer mit wem kommuniziert. Der Lösung dieses Problems hat sich „The Tor Project“<sup>2</sup> verschrieben. Dabei handelt es sich um die Entwickler des so genannten *Tor-Netzwerks*. Tor ist die Abkürzung für „The Onion Router“, und stellt ein Netzwerk dar, welches das Konzept des Onion-Routings anwendet. Dabei werden vor einer Übertragung die einzelnen Datenpakete in mehrere Verschlüsselungsschichten eingepackt (wie eine Zwiebel) und über ein Netzwerk von *Onion Routern* wieder schichtweise entpackt, bevor sie an das endgültige Ziel weitergeleitet werden. Dabei ist jeder Hop für das Entfernen von genau einer Verschlüsselungsschicht verantwortlich. Da jede Schicht nur die IP-Adressen des Vorgängers und des Nachfolgers beinhaltet, gibt es kein Gerät im Netzwerk, das sowohl den ursprünglichen Absender als auch das endgültige Ziel kennt. Das Prinzip des Onion Routings wurde bereits in den 1990er Jahren von David Goldschlag, Mike Reed, und Paul Syverson am US. Naval Research Lab entwickelt. [11]

The Tor Project griff dieses Konzept auf und entwickelte ein Netzwerk, das mittlerweile aus mehr als 2 Millionen Nutzern und über 6000 Knoten, den so genannten *Relays*, besteht<sup>2</sup>. Dabei merzten sie zeitgleich viele Schwächen des ursprünglichen Ansatzes aus, was Tor zu hervorragender Anonymisierung verhalf. Jedoch hat auch Tor seine eigenen Schwächen. Wenn es auch im Verhältnis zu damaligen alternativen Lösungen noch als Design mit geringer Latenz zählt, kann die gedrosselte Übertragungsrate in Zeiten von High-Speed-Internet sehr stark ins Gewicht fallen. Außerdem unterstützt Tor kein UDP, wodurch Live-Übertragungen wie VoIP oder Videochat von Grund auf ausgeschlossen sind. [8]

Dies führt unweigerlich zur Frage, ob es eine Alternative zu Tor gibt, die eine bessere Effizienz bei ähnlichen Anonymitäts-Merkmalen anbietet. Eine Möglichkeit wäre der Einsatz von *Virtual Private Networks*, (VPNs). Allgemein gesagt ist ein VPN ein Netzwerk, das ein anderes öffentliches Netzwerk verwendet, um private Daten zu transportieren [20]. Konkret wird dabei im Regelfall ein VPN-Dienst als Proxy, also als Stellvertreter, für die Datenübertragung verwendet, wodurch die eigene IP-Adresse hinter der des VPN-Dienstes versteckt wird. Dieser Ansatz ist im Vergleich zu Tor wesentlich effizienter,

<sup>2</sup> Siehe <https://metrics.torproject.org/networksize.html>

bietet jedoch nicht das gleiche Maß an Anonymität, da der VPN-Dienst die originale Quell-IP-Adresse kennt. Wird dieser kompromittiert, kann keine Anonymität mehr gewährleistet werden.

Ein Kompromiss könnte eine hybride Lösung aus Onion-Routing und VPN sein. Um dabei aber eine möglichst hohe Übertragungsrate und Sicherheit zu gewährleisten, sollte dabei auch auf eine möglichst effiziente und sichere VPN-Lösung zurückgegriffen werden. Eine solche Lösung stellt *WireGuard* dar. *WireGuard* ist ein sicherer VPN-Tunnel, der ursprünglich für Linux entwickelt wurde, um eine Alternative zum standardmäßig vorinstallierten IPsec anzubieten. [9] Dabei operiert *WireGuard* vollständig kernelseitig auf der dritten Schicht des OSI-Modells, was es zu einem sehr performanten VPN-Dienst macht. Außerdem nutzt es für Verschlüsselung, Schlüsselaustausch und Übertragung modernste Kryptografie-Verfahren, was es zu einem sehr sicheren VPN-Protokoll macht.

Ziel dieser Arbeit soll es sein, eine Hybrid-Lösung aus VPN und Onion-Routing zu testen, die *WireGuard* als VPN-Protokoll nutzt. Die Hypothese lautet dabei, dass dieser VPN-Onion-Routing-Ansatz eine viel performantere Alternative zu Tor darstellt, bei ähnlichem Anonymitätsfaktor. Um diese These zu überprüfen, wurden mehrere Multihop-Netzwerke aus *WireGuard*-Knoten eingerichtet, welche sich in der Anzahl der Knoten unterscheiden. Analog dazu wurde ebenfalls ein kleines Netzwerk aus Tor-Relays aufgebaut. Beide Netzwerke wurden mithilfe von Linux-Network-Namespaces simuliert. Um die Performanz zu testen, wurden diverse Netzwerk-Messungen durchgeführt und typische Kennzahlen ermittelt. Die Ergebnisse dieser Messungen sollen ebenfalls in dieser Arbeit vorgestellt, evaluiert und diskutiert werden. Abschließend wird final die Frage geklärt, ob es sich bei dem vorgestellten VPN-Onion-Routing-Ansatz tatsächlich um eine effizientere Alternative zu Tor handelt.



## PROBLEMSTELLUNG

---

Privatsphäre und Anonymität beim Surfen im Web zu gewährleisten ist nicht immer einfach. Viele Webseiten nutzen Cookies nicht nur, um ihren Nutzern für sie optimierte Dienste anzubieten, sondern auch, um deren Verhalten und Gewohnheiten im Web zu überwachen. Dieses Vorgehen wird gemeinhin als „Tracking“ bezeichnet [6]. Dabei können die dadurch gewonnen Informationen nicht nur für ein benutzeroptimierteres Marketing verwendet, sondern auch an Dritte weiterverkauft werden. Zwar müssen Webseiten im Europäischen Raum seit 2018 <sup>1</sup> die Erlaubnis zum Sammeln der Daten ihrer Nutzer einholen, sowie offenkundig machen, für welche Zwecke die gesammelten Daten verwendet werden. Jedoch sind die mittlerweile allseits bekannten „Cookie-Banner“, die beim Öffnen einer Webseite die Erlaubnis des Nutzers einholen, oft sehr unübersichtlich gehalten. Auch verleiten sie durch ihren Aufbau und ihr Design unbedachte Nutzer oft dazu, sämtliche von der Seite verwendeten Cookies zu akzeptieren. Doch auch andere, dritte Parteien können versuchen, das persönliche Surf-Verhalten im Web zu überwachen. Vor allem in Ländern mit eingeschränkter Pressefreiheit oder durch Zensur gefilterten Internetzugriff ist das Problem der staatlichen Überwachung besonders groß. Aus diesem Grund ist es wichtig, Techniken anzubieten, mit denen man einer solchen Überwachung entgehen kann.

### 2.1 ONION-ROUTING

Anonymes Surfen im Web bedeutet in erster Linie, die eigene Identität zu verschleiern. Und diese ist im Internet eng mit der persönlichen *IP-Adresse* verbunden. Ähnlich wie reale Adressen, sind IP-Adressen notwendig, damit ein Endgerät über das Netzwerk erreichbar ist. Da diese Adressen deshalb eindeutig sein müssen, hat das jedoch den Nebeneffekt, dass man anhand seiner IP-Adresse auch eindeutig identifiziert werden kann. Will man also im Web anonym bleiben, muss man einen Weg finden, seine eigene IP-Adresse zu verschleiern.

*Onion Routing* bietet diese Möglichkeit. Goldschlag et. al. beschrieben Onion-Routing als „eine Infrastruktur zur privaten Kommunikation über ein öffentliches Netz“ [11, 24]. Es werden hierbei anonyme Routen zwischen Sender und Empfänger aufgebaut, die sehr resistent gegen Lauschangriffe und Analysen des Netzwerkverkehrs sind. Die Verbindungen sind bidirektional, können also Daten in beide

---

<sup>1</sup> Vgl. Datenschutzgrundverordnung (DSGVO)

Richtungen übertragen. Dabei spielt es keine Rolle, ob es sich um eine verbindungsorientierte (TCP) oder verbindungslose (UDP) Übertragung handelt. Auch werden die Routen im Normalfall regelmäßig geändert, um somit Netzwerkverkehrsanalysen zusätzlich zu erschweren. Die Routen selbst bestehen aus mehreren Knotenpunkten, in früheren Entwürfen „Mixes“ [11], unter Tor [8] auch „Relays“, genannt. Zur Vereinheitlichung und zum besseren Verständnis wird im weiteren der Begriff „Onion-Router“ verwendet. Ein über dieses Netzwerk übertragendes Paket wird zuerst in mehrere Verschlüsselungsschichten gepackt, wobei jede Schicht mit dem Schlüssel eines Onion-Routers verschlüsselt wird, über den das Paket bei der Übertragung geroutet wird. Jeder passierte Onion-Router innerhalb des Pfades ist dabei wiederum für die Entschlüsselung einer Verschlüsselungsschicht und die Weiterleitung an den nächsten Onion-Router zuständig. Ein solcher Pfad wird auch als „Circuit“ bezeichnet. Das Entpacken der einzelnen Schichten wird in umgekehrter Reihenfolge zum Einpacken durchgeführt. Dabei enthält jede Schicht kryptografische Informationen, sowie den nächsten Onion-Router. [Abbildung 1](#) zeigt diesen Prozess anhand von drei Onion-Routern. Der Datenverkehr wird in so genannten „Zellen“ übertragen. Diese Zellen haben dabei immer dieselbe Größe und werden bei Bedarf auf die entsprechende Größe aufgefüllt. Ein Verbindungsaufbau erfolgt durch das Senden einer initialen „Zwiebel-Struktur“. Anhand dieser Struktur werden die ausgewählten Router konfiguriert. Für den initialen Schlüsselaustausch werden asymmetrische Verschlüsselungsverfahren verwendet. Die weitere Ver- und Entschlüsselung erfolgt unter Verwendung von symmetrischen Schlüsseln. Erhält der endgültige Empfänger die Zwiebel-Struktur, ist der Verbindungsaufbau abgeschlossen und es können Daten in beide Richtungen gesendet werden. [11, 24]

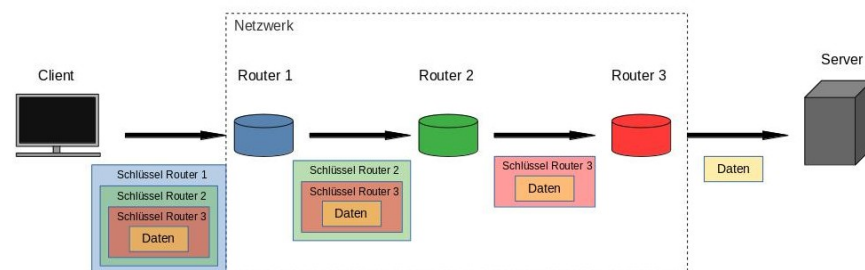


Abbildung 1: Datenübertragung durch ein Onion-Routing-Netzwerk. Der Client verschlüsselt das Paket schichtweise mit den Schlüsseln der traversierten Onion-Router in umgekehrter Reihenfolge (Hier erst Router 3, dann 2, dann 1). Jeder Onion-Router entschlüsselt anschließend wieder eine Schicht. Der letzte Router leitet das Paket an das eigentliche Ziel weiter.

Die Anonymität wird dadurch erreicht, dass jedes Gerät im Netzwerk nur die unmittelbar mit ihm verbundenen Geräte kennt. Einzige Ausnahme bildet hier der Initiator (Client), denn dieser kennt auch das endgültige Ziel. Somit gibt es kein Gerät, das sowohl den Absender, als auch den Empfänger kennt. Der letzte Onion-Router im Netzwerk, also derjenige, welcher mit dem eigentlichen Ziel kommuniziert, dient dabei als Proxy für das gesamte Netzwerk. Der Empfänger sieht folglich nur die IP-Adresse dieses letzten Knotens.

*Allerdings gibt es auch hier Ausnahmen, sog. Hidden Services [8, 21]. Mehr dazu in einem späteren Kapitel.*

## 2.2 TOR

Bisher wurde ein allgemeiner Einblick in die Funktionsweise eines Onion-Routing-Netzwerkes gegeben. Dabei handelt es sich jedoch auch nur um eine allgemeine Definition. Es gibt und gab verschiedene, konkrete Umsetzungen dieses Prinzips, wie beispielsweise Babel [12] oder Mixmaster [7]. Wirklich durchgesetzt hat sich jedoch nur eine Lösung: *Tor*. *Tor* ist eine Onion-Routing-Infrastruktur der zweiten Generation. Es beseitigt viele Schwächen der ursprünglichen Entwürfe, indem es viele neue Funktionen anbietet. Im folgenden werden die wichtigsten aufgezählt: [8]

**VERBESSERTE ANONYMISIERUNG** Eine große Schwäche des alten Designs war die bereits erwähnte Zwiebel-Struktur. Ein einzelner, kompromittierter Onion-Router hätte den gesamten Datenverkehr aufzeichnen und anhand dieser Daten-Struktur die nachfolgenden Knoten zum Entschlüsseln zwingen können. *Tor* verwendet einen „teleskopartigen“ Ansatz zum Pfad-Aufbau. Der Schlüsselaustausch erfolgt mit jedem Knoten einzeln. Nachdem die Schlüssel gelöscht wurden, kann kein Datenverkehr mehr nachträglich entschlüsselt werden. [8]

**TRENNUNG VON ANONYMITÄT UND PROTOKOLLBEREINIGUNG** Im Vergleich zu früheren Ansätzen benötigt *Tor* nicht für jedes Protokoll bzw. jede Anwendung einen eigenen „Anwendungs-Proxy“, damit das jeweilige Protokoll unterstützt wird. Stattdessen verwendet *Tor* das SOCKS-Interface [18], was eine Unterstützung der meisten TCP-Anwendungen ohne weitere Modifikation ermöglicht. Jedoch unterstützt *Tor* durch diesen Ansatz auch leider nur TCP-Anwendungen, und kein UDP. [8]

**CONGESTION CONTROL** *Tor* bietet durch seinen großteils dezentralen Aufbau effizientere Mechanismen an, um den Datenverkehr besser auf das Netzwerk zu verteilen. Dadurch beugt es so genannten „Bottle-Necks“, also Engstellen, besser vor, als frühere Implementierungen. [8]

*Tor ist nicht komplett dezentral, da es Directory Server verwendet.*

**DIRECTORY SERVER** In früheren Designs wurden die Zustands-Informationen der Nodes über das gesamte Netzwerk geflutet. *Tor* hingegen



bietet einzelne, vertrauenswürdige Nodes, so genannte „Directory Server“ oder auch „Directory Authorities“ an, von denen die Onion-Router im Netzwerk diese Informationen in bestimmten Intervallen abrufen. [8]

**INTEGRITÄTSÜBERPRÜFUNG** Tor überprüft im Gegensatz zu vorherigen Implementierungen die Integrität der Daten, bevor sie das Netzwerk verlassen. Dies macht eine Veränderung der Daten durch einzelne, kompromittierte Nodes nahezu unmöglich. [8]

**RENDEZVOUSPUNKTE UND HIDDEN SERVICES** In einem klassischen Onion-Routing-Netzwerk kennt jeder Teilnehmer nur seinen direkten Vorgänger und Nachfolger. Eine Ausnahme bildet der Client, da dieser auch das endgültige Ziel kennt. Dies kann jedoch auch durch so genannte „Rendezvouspunkte“ verhindert werden. Das sind spezielle Server, welche die Identität des Empfängers ebenfalls schützen. Die Empfänger werden somit zu *Hidden Services*. [8]

*Die Summe all dieser Hidden Services ist das, was wir heute gemeinhin als „Darknet“ bezeichnen [21].*

Jeder Onion-Router in einem Tor-Netzwerk hat eine mit TLS gesicherte Verbindung zu jedem anderen Router im Netzwerk. Die Tor-Software läuft dabei vollständig im User-Space. Auf den Systemen der Nutzer muss mindestens ein Tor-Proxy installiert sein, um sich mit dem Tor-Netzwerk verbinden zu können. Dieser Proxy ruft die Informationen zu verfügbaren Onion- Routern ab und baut die Pfade durch das Netzwerk auf. Er ist außerdem für die schichtweise Verschlüsselung mit den Schlüsseln der für die Kommunikation verwendeten Onion-Router verantwortlich. [8] Um den Zugang zum Tor-Netzwerk auch weniger technisch versierten Nutzern zu erleichtern, wurde außerdem ein Tor-Browser entwickelt, der wie ein normaler Browser zum Surfen im Internet genutzt werden kann.

Die Daten werden in 512 Byte großen Paketen durch das Netzwerk geschickt, so genannte „Cells“. Eine Cell, oder Zelle, besteht dabei aus einem Header und dem Payload. Ersterer enthält eine Circuit-ID zum Identifizieren des Pfades, über welchen die Zelle geroutet wurde, und ein Befehl, wie der nächste Onion-Router die Zelle interpretieren soll. [8]

Tor ist ein verteiltes System, das bedeutet, dass die Onion-Router nicht im Besitz einer einzelnen Person oder Organisation sind. Stattdessen sind die unter Tor auch „Relays“ genannten Onion-Router über viele verschiedene Nutzer weltweit verteilt. Im Prinzip kann jeder einen eigenen Relay betreiben und ihn mit dem Tor-Netzwerk verbinden. [16] Eine so genannte „Exit-Policy“ erlaubt es jedem Betreiber eines Exit-Nodes ein Stück weit zu beeinflussen, welche Art von Datenverkehr über diesen laufen soll, indem man Ports oder Hosts blockieren kann. Das ist insofern wichtig, da es juristische Konsequenzen verhindern kann, falls illegale Inhalte über diesen Exit-Node abgefragt werden. [8]

*Exit-Nodes sind die letzten Relays im Netzwerk, welche letztendlich mit dem eigentlichen Ziel kommunizieren.*



Standardmäßig besteht ein Tor-Circuit aus drei Onion-Routern: *Entry-Guard*, *Middle-Node* und *Exit-Node*. Der *Entry-Guard* kommuniziert direkt mit dem Client und der *Exit-Node* mit dem Internet. Der *Middle-Node* liegt dazwischen. Einer der Haupt-Angriffsvektoren, um die Anonymität in Tor zu gefährden, ist durch die Analyse des Datenverkehrs. Das Teleskop-Prinzip von Tor verhindert zwar eine Assoziation zwischen Sender und Empfänger. Jedoch, wenn *Entry-Guard* und *Exit-Node* zusammenarbeiten, könnten diese durch eine Analyse des Datenverkehrs die Kommunikation einem bestimmten Sender und Empfänger zuordnen. Hierzu müssten beide Onion-Router unter Kontrolle von ein und derselben Person bzw. Organisation sein. In solch einem Szenario spielt es auch keine Rolle mehr, wie viele Onion-Router zwischen *Entry-* und *Exit-Node* liegen. Aus genau diesem Grund ist es auch nicht sinnvoll, dass ein Circuit aus mehr als drei Routern besteht. Es würde keine zusätzliche Anonymität bieten, dafür aber weitere Performance-Einbußen. Zwar könnte man daraus schlussfolgern, dass dann zwei Onion-Router schon ausreichen würden. Trotzdem sind drei Knoten besser als zwei, da man bei zwei Knoten schon direkt sehen würde, welcher andere Onion-Router verwendet wurde. Da die Circuits aber anhand verschiedener Faktoren wie z. B. der Auslastung einzelner Router ermittelt werden und in regelmäßigen Abständen wechseln, wird die Gefahr, dass *Entry-Guard* und *Exit-Node* derselben Person gehören, relativ klein gehalten. [8, 16] Es wird jedoch deutlich, dass das Maß an Anonymität mit der Anzahl an aktiven Nutzern und Betreibern von Onion-Routern in Verbindung steht. Systeme wie Tor leben also davon, dass sie von möglichst vielen Menschen genutzt werden bzw. möglichst viele Nutzer auch einen eigenen Onion-Router betreiben. [15]

Dieses Maß an Anonymität hat aber leider seinen Preis. Ver- und Entschlüsselung kann zeitaufwendig sein. Und da jedes einzelne Paket schichtweise ver- und entschlüsselt wird, geht dies zu Lasten der Übertragungsrate. Außerdem arbeitet Tor komplett im User-Space, was weitere Performance-Einbußen mit sich bringt, weil Daten zwischen User-Space und Kernel hin und her geschickt werden müssen. Und auch Circuit-Aufbau und -wechsel sind aus zeitlicher Sicht nicht kostenlos. [8, 25] Es stellt sich also die Frage nach einer Lösung, die performanter als Tor ist bei gleicher oder zumindest ähnlicher Anonymität.

## 2.3 VPN

Einen weiteren und etwas effizienteren Ansatz zur Anonymisierung im Netz stellt ein *Virtual Private Network* (VPN) dar. Bei einem VPN wird ein verschlüsselter Netzwerktunnel zum Ziel-Server aufgebaut [20]. Dabei wird ein VPN-Server als Proxy verwendet, wodurch die eigene IP-Adresse maskiert wird, da man von nun an die Adresse

des Proxys besitzt, solange man über das VPN verbunden ist. Der Ziel-Server sieht also nur die IP-Adresse des VPN-Proxys. Das Netzwerk ist virtuell, da es den VPN-Tunnel auf bestehende Netzwerk-Infrastruktur aufsetzt. Als „Tunneln“ bezeichnet man das Einpacken von Paketen eines Netzwerkprotokolls in Pakete eines anderen Netzwerkprotokolls [20]. Im Fall von VPN werden IP-Pakete in Pakete eines spezifischen VPN-Protokolls gekapselt. Das Netzwerk ist privat, weil der VPN-Server die Verbindung Ende-zu-Ende verschlüsselt. Dies unterscheidet ein VPN auch von einem normalen Proxy-Server. [10, 20] Häufige Anwendungsbeispiele sind:

- Verbindet man sich mit einem VPN-Dienst und bekommt von ihm eine entsprechende IP-Adresse, kann man so agieren, als wäre man im selben Netzwerk wie der VPN-Dienst angemeldet, auch wenn man es physisch nicht ist. Auf diese Art und Weise kann man z. B. auch von Fern auf Dienste und Ressourcen im Firmennetzwerk zugreifen, deren Zugriffsrechte normalerweise auf das Firmennetzwerk beschränkt sind.
- VPN kann auch zum Umgehen von Geoblocking genutzt werden. Manche Inhalte sind nur in bestimmten Ländern verfügbar. Besitzt man keine IP-Adresse die zum Adressraum dieses Landes gehört, werden diese Inhalte blockiert. Verbindet man sich jedoch mit einem VPN-Server, der in diesem Land steht, kann man die Blockierung umgehen, da man über diesen Dienst eine passende IP-Adresse bekommt.
- Sich mit einem ungeschützten WLAN-Hotspot oder generell mit einem Fremdnetzwerk zu verbinden, kann gefährlich sein, da Dritte hier leichter an sensible Daten gelangen können. Tunnelt man seinen Netzwerkverkehr jedoch mit einem VPN, sind die gesendeten Daten auf jeden Fall verschlüsselt und vor Dritten besser geschützt.

Auch wenn VPNs relativ effizient sind und auch ein gewissen Maß an Anonymität liefern, so ist letzterer Punkt jedoch nicht mit Tor oder Onion-Routing im Allgemeinen vergleichbar. Eine große Schwäche von VPNs ist nämlich, dass der Anbieter des Dienstes sowohl die originale IP-Adresse kennt, als auch das Ziel. Will man dies verhindern, ist die Verwendung einer einfachen VPN-Verbindung nicht ausreichend.

### 2.3.1 Kaskadierende VPNs

Eine Möglichkeit, der mangelnden Anonymität von VPNs entgegenzuwirken, wäre, mehrere VPN Server in einer Kette nacheinander zu schalten. Dies bezeichnet man auch als „kaskadierende VPNs“ [22]. Bei diesem Ansatz verbindet sich der Client mit einem VPN-Server,

welcher sich anschließend wieder mit einem weiteren VPN-Server verbindet usw. Der letzte Server in der Kette verbindet sich anschließend mit dem eigentlichen Ziel-Server. Das Prinzip wird nochmal in [Abbildung 2](#) dargestellt.

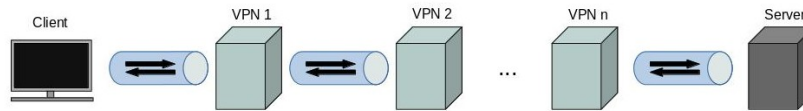


Abbildung 2: Bei einem kaskadierenden VPN werden mehrere VPN-Server hintereinander geschaltet.

Auf diese Art und Weise gibt es keinen Knotenpunkt mehr in der Kette, der sowohl den Sender als auch den Empfänger kennt. Doch auch wenn dieser Aufbau bereits einem Onion-Routing-Netzwerk sehr nahe kommt, bietet es nach wie vor nicht das selbe Maß an Anonymität. Zum einen müssten die VPN-Server im Besitz von unterschiedlichen Anbietern sein, um zu verschleiern, wer mit wem kommuniziert. Zum anderen muss man den Anbietern vertrauen, dass diese keine Nutzer-Aktivität aufzeichnen. Auch ist die Spur durch so ein Netzwerk leichter zu verfolgen, da die Server bzw. Anbieter nicht regelmäßig automatisch gewechselt werden.

### 2.3.2 VPN-*Onion-Routing*

Bisher wurde aufgezeigt, dass Tor zwar ein sehr gutes Maß an Anonymität bietet, dies jedoch leider zu Lasten der Übertragungsrate geht. Auf der anderen Seite gibt es VPN, was zwar sehr effizient ist, aber bei weitem nicht die gleiche Anonymität bietet wie Tor. Kaskadierende VPNs sind zwar ein guter Mittelweg, besitzen aber leider dennoch ein paar Schwächen. Führt man diese Idee aber noch einen Schritt weiter Richtung Tor, drängt sich einem gleich eine weitere Lösung auf: *VPN-*Onion-Routing**. Hier wird das Prinzip kaskadierender VPNs mit dem des Onion-Routings verschmolzen. Der Client kapselt die Daten in mehrere VPN-Pakete, wodurch eine ähnliche Zwiebel-Struktur wie beim klassischen Onion-Routing entsteht. Die Verschlüsselung erfolgt ebenfalls mit den Schlüsseln der einzelnen VPN-Server, welche die Pakete anschließend in umgekehrter Reihenfolge entschlüsseln und an den nächsten Server weiterleiten. Im Ergebnis erhält man ein Netzwerk, das vergleichbare Anonymitätsmerkmale aufweist wie Tor, dabei jedoch wesentlich effizienter ist. So zumindest lautet die Hypothese, welche in den nachfolgenden Kapiteln überprüft werden soll. Um dabei die best möglichen Ergebnisse zu erzielen, wird ein VPN-Protokoll benötigt, das sowohl effizient, als auch sicher und flexibel ist.



Es gibt diverse VPN-Protokolle, die zwar im Kern dasselbe tun, nämlich sichere VPN-Tunnel erstellen, sich aber in Sicherheit und Konfigurierbarkeit stark unterscheiden. So gelten Protokolle wie IPsec oder OpenVPN <sup>1</sup> zwar als relativ sicher, sind aber für technisch nicht versierte schwer zu konfigurieren. Andere wie z. B. PPTP sind zwar leichter zu bedienen, gelten aber mittlerweile eher als unsicher. Zum Glück kann man bei den meisten VPN-Anbietern jedoch auswählen, welches Protokoll verwendet werden soll. Dabei taucht ein spezielles Protokoll seit einiger Zeit immer häufiger auf: Das *WireGuard-Protokoll* [9]. WireGuard ist ein recht neues VPN-Protokoll, das von Jason A. Donenfeld als virtuelles Interface für den Linux-Kernel entwickelt wurde. Es soll dabei viel effizienter, sicherer und vor allem leichter zu bedienen sein, als andere Konkurrenzprodukte. Effizienter, weil WireGuard auf der dritten Schicht des OSI-Modells und komplett im Kernel-Space arbeitet, wodurch teure Übertragungen zwischen Kernel-Space und User-Space vermieden werden. Sicherer, weil WireGuard aus weniger als 4000 Zeilen Code besteht und modernste Kryptografieverfahren verwendet, was die Wahrscheinlichkeit potenzieller Sicherheitslücken stark minimiert. Leichter zu bedienen, weil WireGuard nur ein virtuelles Interface anbietet, das anschließend mit klassischen Linux-Netzwerk-Tools wie `ip(8)` und `ipconfig(8)` bearbeitet werden kann. Das Interface selbst muss nur mit einem Private-Key und den Public-Keys der Peers konfiguriert werden, mit denen es später kommunizieren will. Anschließend kann das Interface direkt verwendet werden. [9] Aktuell unterstützt WireGuard für die Übertragung nur UDP. Die Entwickler haben sich auch bewusst gegen eine Unterstützung von TCP entschieden, da das Tunneln von TCP über TCP zu enormen Performance-Einbußen führt.<sup>2</sup>

*Zum Vergleich,  
IPsec und  
OpenVPN bestehen  
jeweils aus mehreren  
hunderttausend  
Zeilen Code.*

### 3.1 KOMMUNIKATION

Für die Kommunikation mit anderen Geräten wird das so genannte „Cryptokey-Routing“ verwendet. Endpunkte, mit denen ein WireGuard-Client kommunizieren kann, nennt man *Peers* [9]. Damit der Client weiß, mit welchen Peers er kommunizieren darf, hält er in seinem Speicher eine *Cryptokey-Routing-Tabelle*. [Tabelle 1](#) zeigt, wie eine solche Tabelle aussehen kann. In WireGuard erhält jeder Peer – sowie auch der Client – einen privaten und einen öffentlichen Schlüssel

---

<sup>1</sup> Siehe <https://openvpn.net>

<sup>2</sup> Siehe <https://www.wireguard.com/known-limitations>

**WireGuard Interface**

Öffentlicher Schlüssel	Privater Schlüssel	UDP Port
/S9fT4...404=	qPx+Rd...7UM=	49415

**Peers**

Öffentlicher Schlüssel	Erlaubte Adressen
lZ6r4g...8Us=	192.168.0.4\32, 192.168.0.5\32
xT7l5t...J90=	192.168.2.0\24
oTx6Gr...Kqu=	0.0.0.0\0

Tabelle 1: Beispiel einer Cryptokey-Routing-Tabelle.

(Private Key und Public Key). Ein jeder Peer wird anhand seines öffentlichen Schlüssels identifiziert. Diese Schlüssel werden, wie in [Tabelle 1](#) dargestellt, mit einer Liste an zulässigen IP-Adressen verknüpft. Wird z. B. über das WireGuard-Interface aus [Tabelle 1](#) ein Paket versendet, wird in dieser Tabelle nachgesehen, mit welchem Schlüssel das Paket verschlüsselt werden muss. Soll z. B. ein Paket an 192.168.0.5 gesendet werden, wird die Ziel-IP-Adresse des Paketes mit den erlaubten IP-Adressen aus der Routing-Tabelle verglichen. Stimmt diese mit einem Eintrag überein, wird der damit assoziierte öffentliche Schlüssel, in diesem Beispiel *lZ6r4g...8Us=*, zur Verschlüsselung verwendet und das Paket an diesen Peer gesendet. Erhält das Interface ein Paket z. B. von *xT7l5t...J90=*, entschlüsselt und authentifiziert es dieses. Anschließend wird wieder in der Routing-Tabelle nachgesehen, ob die Quell-IP-Adresse des entschlüsselten Paketes aus dem Subnetz 192.168.2.0 stammt. Falls nicht, wird das Paket fallengelassen. [9]

Natürlich können mit WireGuard auch Pakete an Remote-Endpunkte verschickt werden, die sich in anderen Netzwerken befinden. Hierzu kann optional jedem Peer eine öffentliche IP-Adresse zusammen mit entsprechendem UDP-Port zugeordnet werden. Es ist optional, da ein WireGuard-Interface, wenn es ein Paket aus einem Remote-Netzwerk erhält und dieses erfolgreich entschlüsselt und authentifiziert hat, die äußere externe IP-Adresse als Remote-Endpunkte in die Tabelle einträgt. [9]

**3.2 KRYPTOGRAPHIE UND SICHERHEIT**

An dieser Stelle soll noch ein kurzer Einblick in die von WireGuard verwendeten kryptografischen Verfahren gegeben werden. Wie bereits erwähnt ist ein wesentlicher Sicherheitsaspekt von WireGuard die Verwendung von modernen und als sicher geltenden kryptografischen Verfahren. Am Beginn jeder Kommunikation steht ein Zwei-

Wege-Handshake, welcher zugleich dem Schlüsselaustausch dient. Der Handshake basiert dabei auf dem „IK-Muster“ von Noise [23]. Dabei handelt es sich um ein Protokoll-Framework, dem auch die übrigen verwendeten Kryptographieverfahren angehören: [9]

- Für die Verschlüsselung der Pakete wird *ChaCha20* [5], ein symmetrisches Verschlüsselungsverfahren aus der Familie der Stream-Cipher, verwendet.
- Zusammen mit der Verschlüsselung wird *Poly1305-AES* [3] eingesetzt, welches für die sichere Authentifizierung der Nachrichten zuständig ist.
- Der Schlüsselaustausch erfolgt mit *Curve25519* [4], einer auf elliptischen Kurven basierenden Diffie-Hellman-Funktion.
- Beim Hashing kommt *BLAKE2s* [2] zum Einsatz, ein schneller und sicherer Hash-Algorithmus, welcher auf SHA-3 basiert.
- Für das Generieren der Hash-Table-Schlüssel wird die Pseudo-Random-Funktion *SipHash24* [1] verwendet.
- Die Schlüsselableitung der öffentlichen Schlüssel von den privaten basiert auf der *HMAC-based Extract-and-Expand Key Derivation Function* (HKDF) [19].

Zusätzlich zu den oben aufgelisteten Verfahren setzt WireGuard noch weitere Praktiken ein, um die Sicherheit zu erhöhen. So antwortet ein Interface nicht auf Nachrichten, die nicht authentifiziert wurden. Auf diese Weise sind WireGuard-Interfaces gegenüber illegitimen Peers oder anderen Netzwerk-Scanning-Tools unsichtbar. Auch besitzt WireGuard einen eingebauten Mechanismus, um Post-Quantum-Entschlüsselung [14] vorzubeugen. Die Sicherheit der meisten gängigen Verschlüsselungsverfahren basiert auf der Tatsache, dass es zur Zeit noch kein Verfahren oder keine Computer gibt, mit dem die Verschlüsselung in annehmbarer Zeit geknackt werden kann. Daher können Quanten-Computer eine Bedrohung für viele moderne Kryptographieverfahren wie beispielsweise Curve25519 sein. Um dem entgegenzuwirken, bietet WireGuard optionale „Pre-Shared-Keys“ an, also symmetrische Schlüssel, welche die Peers zusätzlich austauschen. Diese 256-bit-Schlüssel sollen dafür sorgen, dass aufgezeichneter Datenverkehr, der in Zukunft nachträglich mit einem Quanten-Computer entschlüsselt werden soll, zusätzlich gesichert ist. Dies basiert auf der Annahme, dass die Schlüssel bis dahin schon lange vergessen sind. Zudem wird auch Denial-of-Service-Angriffen entgegengewirkt. Den Angriffsvektor stellen in diesem Fall die CPU-intensiven Curve25519-Multiplikationen dar, welche bei jedem Handshake durchgeführt werden. Ein Angreifer könnte hier, ähnliche wie bei einer SYN-Flood-Attacke, immer wieder einen Handshake initiieren, um so die CPU

des Peers auszulasten. Spezielle Cookies, die bei einer Auslastung anstelle der Handshake-Nachricht gesendet werden, sollen dem entgegenwirken. [9]



Nachdem bisher ausführlich auf die Theorie eingegangen wurde, sollen nun in diesem Kapitel die Netzwerkumgebungen sowie deren Konfiguration erläutert werden. Als erstes werden Methoden und Verfahren der Versuchsaufbauten sowie der Tests vorgestellt und erörtert. Anschließend werden die eingesetzten Tools und Technologien durchleuchtet, die beim Aufbau der Netzwerke und Tests zum Einsatz kamen. Zum Abschluss werden Topologie und die Parameter des Tor-Netzwerkes sowie der verschiedenen WireGuard-Konfigurationen vorgestellt. Die Ergebnisse der Messungen werden in [Kapitel 5](#) evaluiert.

#### 4.1 METHODIK UND VERFAHREN

Um eine homogene, vergleichbare und leicht reproduzierbare Testumgebung zu schaffen, wurden sämtliche Konfigurationen und Tests lokal durchgeführt. Es wurde also keine Netzwerk-Infrastruktur mithilfe von Hardware und Kabeln aufgebaut, sondern ausschließlich auf softwareseitige Simulationen gesetzt. Diese Simulationen können bei Installation der entsprechenden Abhängigkeiten leicht reproduziert und durchgeführt werden.

##### 4.1.1 Technische Daten

Bei dem System, das für die Messungen eingesetzt wurde, handelt es sich um ein *Manjaro-Linux*<sup>1</sup>, welches zu den Arch-Linux-Distributionen gehört. Speziell wurde die 64-Bit-Version mit einer KDE-Plasma Desktopumgebung und der Kernel-Version 5.10.59-1 verwendet. Das Gerät, auf dem die Simulationen durchgeführt wurden, ist ein HP 250 G7 Notebook mit 16 GB RAM, einem Intel Core i5-1035G1 Prozessor und einer 512 GB SSD-Festplatte.

##### 4.1.2 Allgemeines Test-Setup

Für die Simulation der Netzwerk-Geräte wurden Linux-Network-Namespaces verwendet. Damit lassen sich auf einem Linux-System virtuelle Netzwerkgeräte anlegen, von denen jedes seinen eigenen TCP/IP-Stack, eigene Firewall-Regeln und eigene, konfigurierbare Netzwerk-Interfaces besitzt. Die Pfade, Interfaces und Routing-Regeln wurden mithilfe

---

<sup>1</sup> Siehe <https://manjaro.org/>

von IP-Table-Rules definiert. Für den Aufbau der WireGuard-Netzwerke wurde eigens ein Kommandozeilentool entwickelt, das die Konfiguration des Netzwerks und der Interfaces erleichtern soll. Dieses Tool wird in einem späteren Abschnitt vorgestellt. Die Tests selbst wurden in Form von Netzwerkmessungen durchgeführt. Zu diesem Zweck wurden zufällig generierte Daten von einem Client über das jeweilige Netzwerk an einen Server gesendet. Dabei wurde der Netzwerkverkehr sowohl am Client als auch am Server mitgeschnitten. Die Standard-Dauer eines Tests beträgt 60 Sekunden, das heißt, dass jeweils eine Minute lang Daten von dem Client an den Server geschickt werden. Um jedoch auch die Effizienz der Netzwerke unter einer etwas länger andauernden Last zu überprüfen, wurden als Referenz ebenfalls 5-minütige Messungen durchgeführt. Damit die Messergebnisse trotz der Zeitunterschiede vergleichbar bleiben, wurde darauf geachtet, dass jeder Test immer genau 100 Messergebnisse liefert. Das bedeutet, dass z. B. bei den 1-minütigen Messungen alle 0.6 Sekunden und bei den 5-minütigen Messungen alle 3 Sekunden ein neuer Messbericht erzeugt und abgespeichert wird. Um die Aussagekraft mancher Messergebnisse validieren zu können, wurden auf Basis von Reihenmessungen Konfidenzintervalle berechnet. Die maximale Bandbreite wurde in jedem Netzwerk auf 1 GBit begrenzt. Um die Gefahr einer Erhöhung der Paketverlustquote durch das Überlaufen von TCP-Puffern zu minimieren, wurden die maximalen Puffergrößen auf die in [Tabelle 2](#) angegebenen Werte gesetzt. Bei allen Tests wurde eine Socket-Puffer-Größe von 512 KByte verwendet. In iperf3 hat das Setzen von hohen TCP-Puffergrößen auch einen Einfluss auf UDP-Traffic <sup>2</sup>.

Puffer	Wert
net.core.wmem_max	25165824
net.core.rmem_max	25165824

Tabelle 2: Die gewählten TCP-Puffergrößen für das Senden und Empfangen von Daten über Netzwerk.

#### 4.1.3 WireGuard vs. Tor vs. Multihop-Netzwerk

Es wurden insgesamt sieben Netzwerk-Setups getestet. Je ein WireGuard-Multihop-Netzwerk, bestehend aus zwei, drei und vier Peer-Nodes, ein Tor-Netzwerk, welches aus drei Onion-Routern besteht, sowie je ein Multihop-Netzwerk ohne WireGuard oder Tor mit zwei, drei und vier Nodes. Die unterschiedliche Anzahl der Knotenpunkte in den Netzwerken wurde gewählt, um zu beobachten, wie sich die Performance bei Hinzufügen von weiteren Knoten verändert. Das Multihop-

<sup>2</sup> Siehe <https://github.com/esnet/iperf/issues/261>

Netzwerk ohne Onion-Routing wurde ebenfalls als Referenz hergenommen, um die Performance-Einbußen von WireGuard-basiertem Onion-Routing zu evaluieren. Lediglich das Tor-Netzwerk besteht nur aus drei Onion-Routern, da dies die standardmäßige Konfiguration darstellt, die Tor auch in der Praxis verwendet.

#### 4.1.4 UDP vs. TCP

Um die Effizienz der Konfiguration unter Verwendung der beiden Übertragungsprotokolle UDP und TCP miteinander zu vergleichen, wurden bei sämtlichen Tests auch jeweils beide Protokolle für die Übertragung verwendet. Einzige Ausnahme bilden die Messungen des Tor-Netzwerkes. Hier konnte aus technischen Gründen nur TCP getestet werden, da Tor kein UDP unterstützt.

#### 4.1.5 Gemessene Kennzahlen

Untersucht wurden bei den Messungen verschiedene Kennzahlen, die zur Beurteilung der Geschwindigkeit eines Netzwerkes zur Hilfe genommen werden. Der wohl wichtigste Messwert hierbei ist die *Übertragungsrate*, welche angibt, wie viele MBit pro Sekunde durch das Netzwerk übertragen wurden. Auch der *Goodput*, also die Anzahl der tatsächlichen Nutzdaten in MByte, welche sich nach Abzug von Meta- und Paketinformationen ergibt, ist eine wichtige Kennzahl, die bei den Tests ermittelt wurde. Hierzu wurden die serverseitig gemessenen, sprich die tatsächlich angekommenen Daten, zur Hand genommen. Bei den Tests mit TCP wurden zusätzlich die *Round-Trip-Time* sowie die *Retransmits* gemessen. Bei ersterem handelt es sich um die Zeit, die ein Paket benötigt, um von der Quelle zum Ziel und wieder zurück gesendet zu werden. Unter *Retransmits* fasst man alle Pakete zusammen, die während der Übertragung verloren gegangen sind und deshalb erneut gesendet werden mussten. Diese beiden Werte können nur unter Verwendung von TCP gemessen werden, da für deren Messung eine feste Verbindung zwischen Quelle und Ziel bestehen muss, was unter UDP nicht der Fall ist. Dafür werden bei den Tests mit UDP hingegen noch zusätzlich der *Jitter* sowie der *Paketverlust* gemessen. Unter *Jitter* versteht man das Taktzittern bei der Übertragung, also eine Schwankung bei der Genauigkeit des Übertragungstaktes. Dieser wird in Millisekunden angegeben. Der *Paketverlust* ist das Pendant zu den *Retransmits* unter TCP. Da UDP verbindungslos ist, werden verlorene Datenpakete nicht erneut gesendet, womit diese vollständig verloren bleiben. Der *Paketverlust* kann sowohl in verlorenen Paketen, als auch in Prozent angegeben werden. In diesem Fall wurde der prozentuale Anteil verwendet.

#### 4.1.6 Lokale Bedingungen vs. Internet-Bedingungen

Um bei den Messungen möglichst realitätsnahe Testbedingungen zu erreichen, wie sie auch im Internet zu finden sind, wurden klassische Netzwerk-Ereignisse wie Paketverlust und Paket-Drop emuliert. Für die Emulation der Paketverlustquote wurde dabei das Gilbert-Elliot-Modell [13] verwendet. Dabei handelt es sich um ein Markovsches Modell mit zwei Zuständen, einem guten und einem schlechten Zustand, sowie den Wahrscheinlichkeiten eines Zustandswechsels. Im guten Zustand ist die Verlustwahrscheinlichkeit sehr gering, was optimalen Netzwerkbedingungen gleichkommt. Im schlechten Zustand hingegen ist die Verlustquote relativ hoch, was ein Netzwerk mit starken Interferenzen simuliert. Das Modell benötigt dabei vier Parameter:  $p$ ,  $r$ ,  $1-h$  und  $1-k$ . Die Werte der Parameter wurden folgendermaßen gewählt:

$p$	$r$	$1-h$	$1-k$
0.000132253	0.00811837	0.440309	0.000628

Tabelle 3: Die Parameter für das Gilbert-Elliot-Modell zur Emulation von Paketverlusten in einem simulierten Netzwerk.

$p$  ist dabei die Wahrscheinlichkeit einer Zustandsänderung von gut zu schlecht,  $r$  die Wahrscheinlichkeit einer Zustandsänderung von schlecht zu gut,  $1-h$  die Verlustquote im schlechten und  $1-k$  die Verlustquote im guten Zustand. Die Werte wurden entsprechend des in [13] vorgestellten Modells gewählt, da mit diesen Werten ein sehr realitätsnahes Ergebnis erreicht werden kann. Die Evaluierung für dieses Modell wurden mithilfe von Netzwerkinfrastruktur der Deutschen Telekom durchgeführt, weshalb die Netzwerk-Emulation die Bedingungen in einem Netz der Deutschen Telekom simuliert.

Als Referenz zu den Messungen bei emulierten Netzwerkbedingungen wurden sämtliche Tests auch ohne eine Emulation durchgeführt. Damit sollte überprüft werden, wie sich das WireGuard- und das Tor-Netzwerk unter optimalen Bedingungen verhalten und damit auszuschließen, dass die Ergebnisse nur den zufälligen Ereignissen des Gilbert-Elliot-Modells zuzuschreiben sind.

#### 4.1.7 Logging

Zur weiteren Verarbeitung der Testergebnisse wurden die Messwerte mitgeschnitten und in Form von Json-Dateien abgespeichert. Die Logs bestehen dabei sowohl aus dem clientseitigen, als auch aus dem serverseitigen Mitschnitten. Anschließend wurden sie mithilfe von Python-Skripten als Komma-Separierte Werte in CSV-Dateien umgewandelt, da diese leichter durch Statistik- und Datenanalysetools ver-

arbeitet werden können. Mithilfe eines weiteren Python-Skriptes wurden aus diesen CSV-Dateien anschließend die Plots und Statistiken für diese Arbeit erstellt.

## 4.2 VERWENDETE TECHNOLOGIEN

In diesem Abschnitt werden die Tools und Technologien vorgestellt, die bei der Inbetriebnahme der Netzwerke und Test-Umgebungen zum Einsatz kamen.

Wie bereit erwähnt werden für die Simulation der Netzwerkgeräte Linux-Network-Namespaces eingesetzt. Diese werden mit dem Linux-Tool `netns(8)` aufgebaut und konfiguriert. Mit dem Befehl `ip netns add <name>` lässt sich beispielsweise ein neuer Network-Namespace anlegen, welches anschließend mit `ip netns exec <name>` konfiguriert werden kann.

Die IP-Table-Rules und Routen wurden mit den klassischen Linux-Kommandozeilentools `ip route`, `ip addr` und `ip link` angelegt.

Für die Konfiguration der WireGuard-Interfaces wurde WireGuards Kommandozeilentool `wg(8)` verwendet. Damit lassen sich sowohl die Interfaces anlegen und Schlüssel generieren, als auch die Cryptokey-Routing-Tabellen konfigurieren.

Für die Emulation von Paketverlustquote und Paket-Korruption wurde `tc-netem(8)` verwendet. Damit lassen sich neben Paketverlust und Paket-Korruption auch weitere Artefakte von Netzwerkübertragungen wie z. B. Limitierung, Duplizierung oder Verzögerung von Paketen in ein Netzwerk-Interface einbauen, welches diese Artefakte anschließend mit einer angegebenen Wahrscheinlichkeit oder einem spezifizierten Rechen-Modell in den ausgehenden Netzwerkverkehr einbaut.

Die Messungen wurden mit `iperf3` durchgeführt. Um eine Messung durchzuführen, muss eine Instanz auf dem Client und eine auf dem Server laufen, wobei mit Parametern angegeben wird, ob eine Instanz im Client-Modus (`-c`) oder im Server-Modus (`-s`) laufen soll. Im Client-Modus versucht `iperf3` eine Verbindung zum Server aufzubauen, um anschließend die Daten in Intervallen an den Server zu schicken. Standardmäßig werden die Messergebnisse in der Konsole ausgegeben. Es kann jedoch optional angegeben werden, ob die gemessenen Werte in eine Log-Datei geschrieben werden sollen.

Der Client zur Konfiguration des WireGuard-Netzwerkes wurde mit der Programmiersprache Java entwickelt. Die Konfiguration des Clients, der Routen und der Peers mitsamt aller Interfaces erfolgt dabei über eine Json-Datei, die dem Programm als Kommandozeilenargument mit dem Befehl `add <Json File>` übergeben werden kann. Mit `remove <Interface Name>` lässt sich das WireGuard-Interface mit dem übergebenen Namen löschen, sofern es existiert. Der Befehl `show <Interface Name>` gibt die Konfiguration des übergebenen WireGuard-

Interfaces auf der Konsole aus. Die Aktivierung eines WireGuard-Interfaces erfolgt mit dem Befehl `up <Interface Name>`. All diese Informationen lassen sich auch mit `help` auf der Kommandozeile ausgeben. Dabei verwendet der Client die bereits vorgestellten Tools zur Konfiguration von Network-Namespaces, Interfaces und Routen.

#### 4.3 WIREGUARD-NETZWERK

In diesem Abschnitt sollen nun die Konfigurationen des WireGuard-Multihop-Netzwerkes beschrieben werden. Wie bereits zuvor erwähnt, wurden insgesamt drei verschiedene Konfigurationen getestet. Diese unterscheiden sich dabei lediglich in der Anzahl der verwendeten Peers. Diese Entscheidung wurde getroffen, um die Performance-Unterschiede beim Hinzufügen bzw. Entfernen von Peers, also Zwischenschritten bei der Übertragung, sowie den Verschlüsselungsschichten zu evaluieren. Auch der Einfluss auf die Anonymität des WireGuard-Multihop-Netzwerkes durch die Anzahl der Peers soll dadurch untersucht werden. Es gibt je ein Netzwerk mit zwei, drei und vier Peers. Zunächst soll das Netzwerk mit zwei Peers untersucht werden. Dessen Aufbau wird in [Abbildung 3](#) dargestellt.

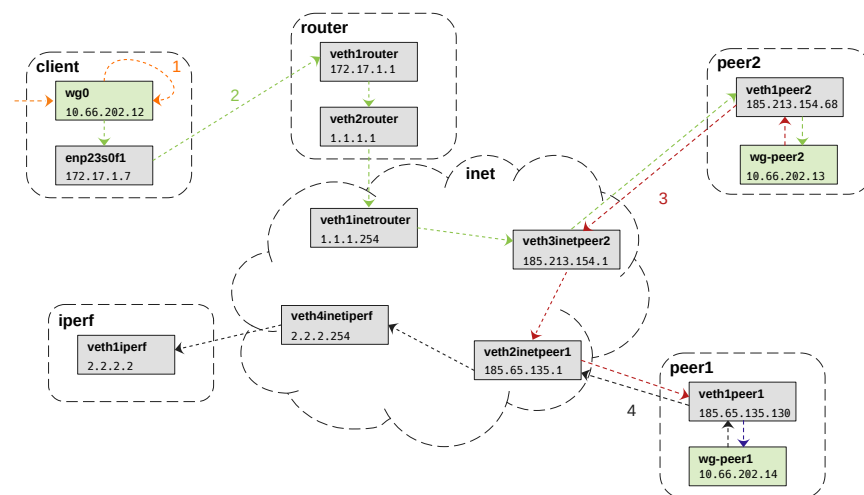


Abbildung 3: Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 2 Peers.

Das Netzwerk besteht aus einem Client, (hier `client`) und einem Server (hier `iperf`), welche miteinander kommunizieren. Der Client ist über einen Router (hier `router`) mit dem „Internet“ (hier `inet`) verbunden. Die Rolle des Entry-Nodes, also der Einstiegspunkt in ein Onion-Routing-Netzwerk, wird von Peer 2 (hier `peer2`) übernommen. Über Peer 1 (hier `peer1`) als Exit-Node wird das Onion-Routing-Netzwerk wiederum verlassen. Einen Middle-Node gibt es in dieser Konfiguration nicht. Das simulierte Internet besteht lediglich aus ein-

zernen Interfaces, wobei jedes davon einem der direkt verbundenen Geräte zugeordnet ist.

Überträgt der Client nun Daten an den Server mit der Ziel-IP-Adresse 2.2.2.2, werden die gesendeten Pakete zunächst in das clientseitige WireGuard-Interface `wg0` umgeleitet. Hier erfolgt anhand der Crypto-Key-Routing-Tabelle die Verschlüsselung der ersten Schicht mithilfe des öffentlichen Schlüssels von Peer 1 und der Ziel-IP-Adresse 185.65.135.130. Anschließend werden die Pakete erneut in `wg0` umgeleitet, welches nun eine weitere Verschlüsselungsschicht mit dem Schlüssel von Peer 2 und der Ziel-IP-Adresse 185.213.154.68 hinzufügt. Die nun mehrfach verschlüsselten Daten werden über den Router und das simulierte Internet an den Entry-Node Peer 2 weitergeleitet. Dieser entfernt in seinem WireGuard-Interface `wg-peer2` die erste Verschlüsselungsschicht. Anhand seiner eigenen Crypto-Key-Routing-Tabelle und der neuen, äußeren Ziel-IP-Adresse 185.65.135.130 kann dieser die Pakete an Peer 1 weiterleiten. In diesem wird nun im WireGuard-Interface `wg-peer1` die zweite und auch letzte Verschlüsselungsschicht entfernt. Durch einen entsprechenden Eintrag in seiner Crypto-Key-Routing-Tabelle weiß Peer 1, dass er die Pakete mit der Ziel-Adresse 2.2.2.2 an das endgültige Ziel, also den Server, weiterleiten kann.

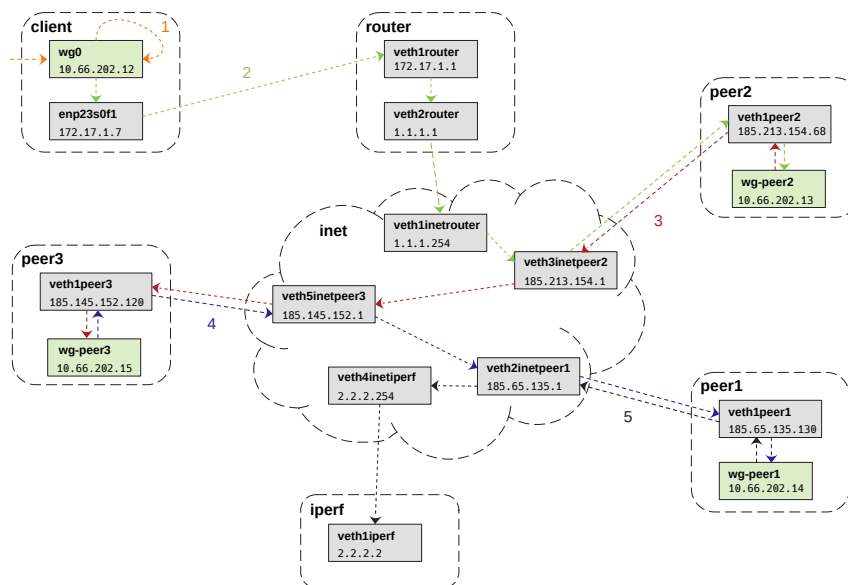


Abbildung 4: Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 3 Peers.

Dieser Ablauf findet im wesentlichen in allen drei Konfigurationen gleichermaßen statt. Im Netzwerk mit 3 Peers, wie in [Abbildung 4](#) dargestellt, existiert noch ein Middle-Node. Dessen Rolle wird von Peer 3 (hier `peer3`) übernommen. Die gesendeten Daten werden, anstatt nur zweimal, auch ein drittes Mal in das WireGuard-Interface

wg0 des Clients umgeleitet. Nach der Verschlüsselungsschicht des Exit-Nodes und vor der Schicht des Entry-Nodes wird eine weitere Schicht eingefügt, welche mit dem öffentlichen Schlüssel von Peer 3 verschlüsselt wird und die Ziel-Adresse 185.145.152.120 besitzt. Nach dem selben Prinzip wie zuvor leitet Peer 2 die Daten nun nicht direkt an Peer 1, sondern an Peer 3 weiter, welcher wiederum seine Verschlüsselungsschicht entfernt und die Pakete an Peer 1 weiterreicht. Auf die selbe Art und Weise wurde auch das Netzwerk mit 4 Peers erweitert. Peer 4 (hier peer4) wird als zweiter Middle-Node zwischen Peer 3 und Peer 1 in das Circuit eingefügt. Auch wie zuvor werden die Daten im Client ein zusätzliches Mal in wg0 umgeleitet, mit dem Schlüssel von Peer 4 verschlüsselt und der IP-Adresse desselben als Ziel versehen, wie in [Abbildung 5](#) gezeigt.

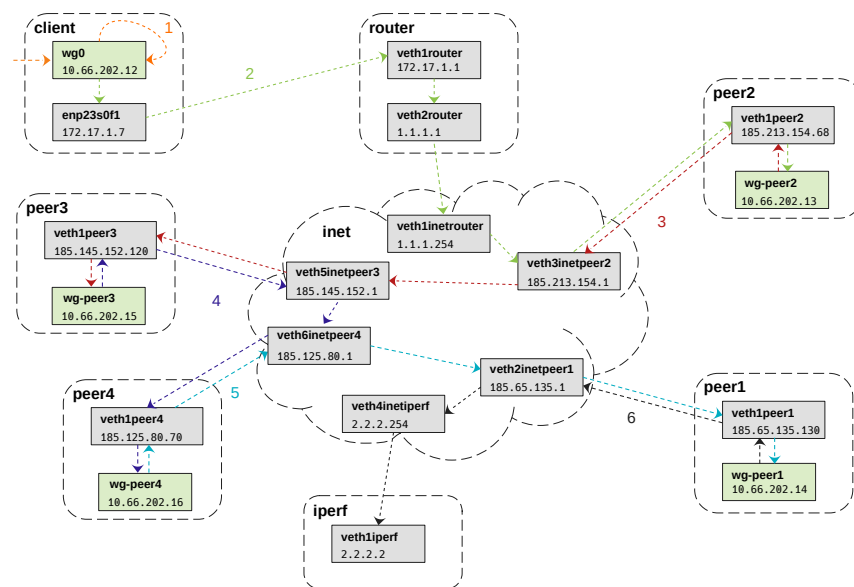


Abbildung 5: Topologie und Kommunikation im simulierten WireGuard-Multihop-Netzwerk mit 4 Peers.

#### 4.4 TOR-NETZWERK

Nachdem nun detailliert der Aufbau des WireGuard-Multihop-Netzwerkes beschrieben wurde, sollen in diesem Abschnitt Topologie und Kommunikationsablauf im lokal getesteten Tor-Netzwerk erläutert werden. Für das Tor-Netzwerk existiert, im Gegensatz zu den anderen beiden Netzwerk-Setups, nur eine Konfiguration. Das hat den Hintergrund, da versucht wurde, einen möglichst praxisnahen Aufbau von Tor zu rekonstruieren. Daher werden, wie in der Praxis auch, nur drei Onion-Router verwendet. Wie aus [Abbildung 6](#) zu entnehmen ist, besteht das Tor-Netzwerk ebenfalls im wesentlichen aus einem Client (hier CLIENT) und einem Server (hier HTTP).



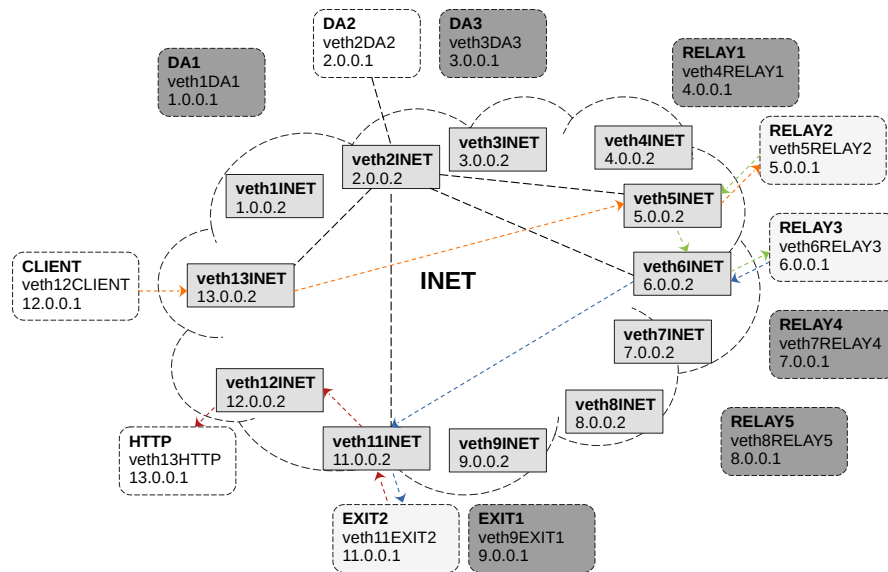


Abbildung 6: Topologie des Tor Netzwerkes mit Beispiel eines möglichen Circuits.

Bei der Generierung des Netzwerkes wurden zudem insgesamt drei Directory-Authorities (hier DA1, DA2 und DA3), fünf Relay-Knoten (hier RELAY1 bis RELAY5) und zwei Exit-Knoten (hier EXIT1 und EXIT2) angelegt. Auch existiert wieder ein „Internet“, über das der Datenverkehr geroutet wird. Wie in der Praxis wird per Zufall eine Directory-Authority gewählt, über welche die Route durch das Netzwerk ausgewählt wird. Im Beispiel aus [Abbildung 6](#) wurde DA2 als Directory-Authority herangezogen. In der Praxis würde dieser nun anhand verschiedener Kriterien wie beispielsweise der Auslastung der einzelnen Relays entscheiden, welche davon als Onion-Router für das Netzwerk genommen werden sollen. In diesem Beispiel wurden RELAY1 und RELAY2 ausgewählt. Die Rolle des Exit-Nodes wird von EXIT2 übernommen. Je nach Konsens des Netzwerkes kann die Wahl der einzelnen Knoten in der Praxis variieren.

Am Beginn einer Kommunikation steht der Client. Dieser besitzt einen so genannten *Onion-Proxy* [8]. Dieser ruft von DA2 die Liste aller aktuell nutzbaren Onion-Router ab. Anhand dieser Liste erzeugt der Client eine zufällige Route durch das Tor-Netzwerk, in diesem Fall RELAY2, RELAY3 und EXIT2, und vollzieht nach und nach mit den einzelnen Routern einen Schlüsselaustausch. Dieser Schritt geschieht erst, wenn Daten über das Netzwerk gesendet werden sollen und nicht schon vorher. Anschließend verschlüsselt der Client die Pakete zuerst mit dem Schlüssel von EXIT2, dann von RELAY3 und zum Schluss von RELAY2. Anschließend werden die Pakete an RELAY2 gesendet. Dieser entschlüsselt die erste Verschlüsselungsschicht und kann anhand der zweiten Schicht feststellen, dass er die Pakete an RELAY3 weiterleiten soll. Dieser entfernt die zweite Verschlüsselungs-

*Hinweis: Das im Beispiel gezeigte Circuit dient nur der Veranschaulichung und stellt nicht zwangsläufig einen real getesteten Circuit dar.*

schicht und entnimmt der letzten Schicht das nächste Ziel der Pakete, EXIT2. Der Exit-Node entfernt nun die letzte Verschlüsselungsschicht, wodurch er das endgültige Ziel der Kommunikation erfährt und die Daten entsprechend an HTTP weiterleitet.

#### 4.5 MULTI HOP-NETZWERK

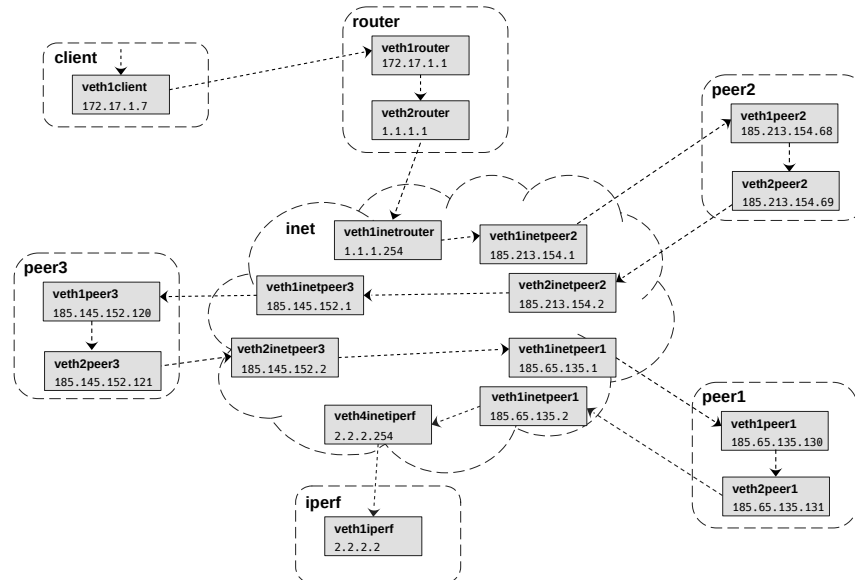


Abbildung 7: Topologie und Kommunikationsablauf des Multihop-Netzwerks ohne WireGuard mit drei Knoten.

Wie zuvor erwähnt, wurde als Referenz für die Performance des WireGuard-Multihop-Netzwerk ein weiteres Multihop-Setup ohne Verwendung von WireGuard oder einem anderen VPN-Protokoll eingerichtet. Die Intention dahinter ist, herauszufinden, wie stark die Verwendung von Onion-Routing mit WireGuard die Performance und auch alle anderen gemessenen Kennzahlen beeinflusst. Aus diesem Grund bestehen die Netzwerke ohne VPN-Protokoll und Onion-Routing aus den selben Komponenten, wie ihre WireGuard-basierten Pendanten. Dadurch entfallen natürlich auch die Anonymisierungs- und Verschlüsselungs-Features der WireGuard-Konfiguration. [Abbildung 7](#) zeigt den Aufbau eines Multihop-Netzwerkes mit drei Knoten. Zur besseren Vergleichbarkeit und der Einfachheit wegen wurde die Namenskonvention des WireGuard-Setups beibehalten. So werden die Knoten weiterhin als Peers bezeichnet, auch wenn es sich dabei nicht mehr WireGuard-Peers handelt. In diesem Setup entfallen die WireGuard-Interfaces und somit die schichtweise Ver- und Entschlüsselung. Stattdessen leiten die Peers die Daten direkt an den nächsten Knoten in der Route weiter. Wie im WireGuard-Netzwerk sendet der Client die Daten über

den Router an den ersten Knoten, hier Peer2. Diesmal jedoch, ohne sie vorher zu verschlüsseln. Peer2 leitet die Daten direkt an Peer3 weiter, da er sie nicht vorher entschlüsseln muss. Anschließend werden die Pakete an Peer1 gesendet, welcher sie schließlich an das endgültige Ziel weiterleitet. Dabei wird die IP-Adresse des Clients nicht maskiert, was bedeutet, dass jedes Gerät im Netzwerk den Absender, sowie den Empfänger kennt.

Bei dem Aufbau mit zwei Knoten entfällt Peer3 aus der Kette und Peer2 sendet die Daten direkt an Peer1. Das Setup mit den vier Knoten fügt, analog zum WireGuard-Netzwerk mit vier Peers, einen weiteren „Middle-Node“ Peer4 zwischen Peer3 und Peer1 ein. Dieser besitzt die Interfaces `veth1peer4` mit der IP-Adresse `185.125.80.70` und `veth2peer4` mit `185.125.80.71`. Auch werden wie bei den anderen Peers zwei weitere Interfaces im „Internet“ eingefügt, `veth1inetpeer4` und `veth2inetpeer4` mit den IP-Adressen `185.125.80.1` für ersteres und `185.125.80.2` für letzteres. An dieser Stelle sollte bereits angemerkt werden, dass es beim Vergleich des Multihop-Netzwerkes keine nennenswerte Unterschiede in den Ergebnissen bezüglich der Anzahl der Peers gab. Deshalb wird im weiteren Verlauf ausschließlich auf die Konfiguration mit drei Knoten Bezug genommen.



## AUSWERTUNG UND DISKUSSION DER ERGEBNISSE

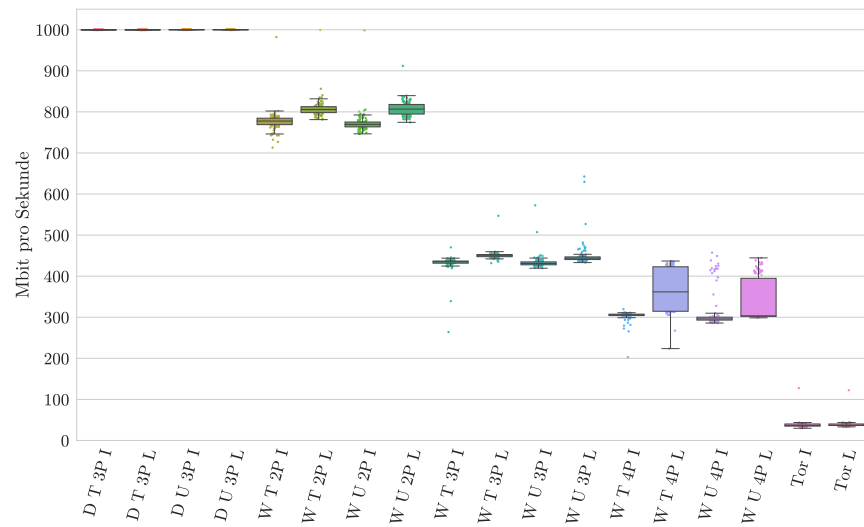
---

Nach ausführlicher Erläuterung von Aufbau, Ziel und Durchführung der Tests, sowie den Werkzeugen und Methoden, die dafür zum Einsatz kamen, sollen in diesem Kapitel nun die Ergebnisse der Tests vorgestellt, ausgewertet und evaluiert werden. Dafür wird zunächst die gemessene Übertragungsrate sowie der Durchsatz der einzelnen Konfigurationen sowohl unter Verwendung von TCP, als auch UDP miteinander verglichen. Anschließend werden die nur bei der Übertragung mit TCP gemessenen Kennzahlen *Retransmits* und *Round-Trip-Time* evaluiert. Den Abschluss bilden die bei den UDP-Tests messbaren Werte *Jitter* und *Paketverlust-Quote*. Zum Schluss werden die vorgestellten Ergebnisse bewertet und diskutiert.

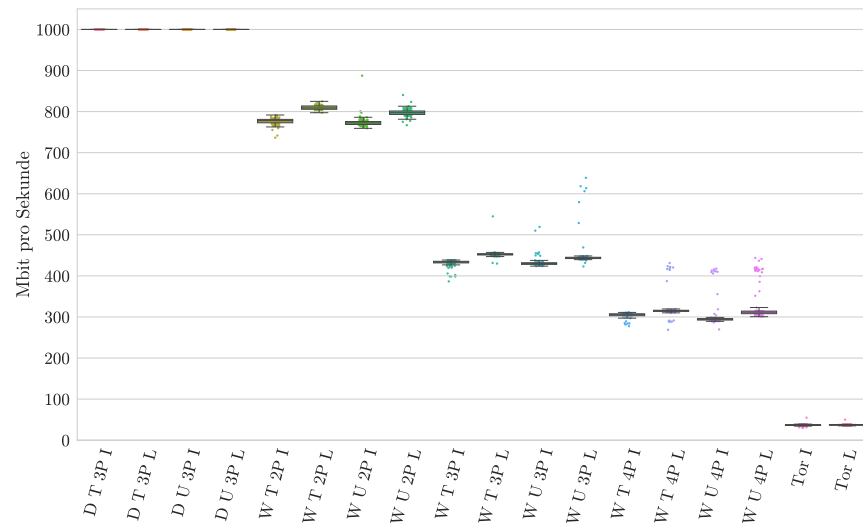
### 5.1 AUSWERTUNG

#### 5.1.1 Übertragungsrate

Im folgenden werden die Übertragungsraten der Messungen, getrennt nach Testdauer, gegenübergestellt. Die Ergebnisse sind dabei als MBit pro Sekunde zu verstehen. Als Maß wurden die Quantile der Messwerte über den jeweiligen Zeitraum herangezogen und als Box-Diagramm dargestellt. Die im folgenden gezeigten Ergebnisse werden dabei nach Netzwerk, Transportprotokoll, Anzahl der Peers im Netzwerk und den Testbedingungen, also mit oder ohne Verwendung des in [Unterabschnitt 4.1.6](#) vorgestellten Modells zur Emulation von Netzwerkbedingungen, unterteilt. Die optimalen Bedingungen ohne Modell werden im Weiteren als „lokale Bedingungen“, die Bedingungen mit Modell als „Internet-Bedingungen“ bezeichnet. [Abbildung 8a](#) zeigt die Übertragungsraten bei einer Messzeit von 60 Sekunden. Wie der Abbildung zu entnehmen ist, weisen alle WireGuard-Konfigurationen eine höhere Übertragungsrate als Tor auf. Auch ist ein eindeutiger Abfall der Performance bei Hinzufügen von weiteren Peers festzustellen, wobei sich dieser beim Schritt von zwei auf drei Peers deutlicher bemerkbar macht, als von drei auf vier. Doch auch letztere Konfiguration ist immer noch um ein vielfaches schneller, als Tor. So beträgt die Übertragungsrate des Setups WireGuard, TCP, 4 Peers, lokale Bedingungen im Median gerundet 303,495 MBit pro Sekunde, während die von Tor unter lokalen Bedingungen nur bei rund 36,7 MBit pro Sekunde liegt.



(a) 60 Sekunden



(b) 300 Sekunden

Abbildung 8: Vergleich der Übertragungsraten in MBit/Sekunde bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), WireGuard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))

Am schnellsten ist das Setup WireGuard, UDP, 2 Peers, lokale Bedingungen mit einer Übertragungsrate von 806,752 MBit pro Sekunde im Median. Schneller sind nur die Multihop-Messungen, hier als „direkte“ Messungen bezeichnet, bei denen weder Tor, noch WireGuard zum Einsatz kamen. Beide Messergebnisse liegen bei 1 GBit pro Sekunde, was in der Einschränkung der Bandbreite auf genau diesen Wert begründet liegt. Daraus resultiert, dass dieses Setup die komplette, zur Verfügung stehende Bandbreite ausnutzt. Dieses hohe Messergebnis lässt sich vor allem dadurch erklären, dass weder Verschlüsselung, noch Onion-Routing verwendet werden, sowie durch den daraus resultierenden, geringeren Protokoll-Overhead. Da es hier bei der Verwendung von zwei, drei oder vier Peers keinen nennenswerten Unterschied bezüglich der Geschwindigkeit gab, werden der Übersichtlichkeit wegen nur die Messergebnisse bei drei Peers dargestellt. Auch lässt sich erkennen, dass die Netzwerke ohne Netzwerk-Emulation minimal schneller sind, als ihre Pendants mit emuliertem Netzwerk. Ähnlich verhält es sich mit den 300-Sekunden Messungen, wie in [Abbildung 8b](#) zu sehen ist. Auch hier sind die WireGuard-Setups mit 2 Peers am schnellsten, während Tor die geringste Übertragungsrate besitzt. Es ist auch zu erkennen, dass die Streuung der Messwerte bei allen Konfigurationen gering ist. Einzige Ausnahme bilden die beiden WireGuard-Tests mit 4 Peers unter lokalen Bedingungen und einer Testdauer von 60 Sekunden. Hier ist die Streuung der Messwerte sowohl mit UDP als auch TCP relativ hoch. Da sich dieses Verhalten jedoch nicht bei der 300-Sekunden langen Messung finden lässt, kann es sich hierbei auch nur um ein zufälliges Artefakt handeln.

### 5.1.2 Goodput

Dieser Abschnitt befasst sich mit der Evaluierung der reinen, transferierten Nutzdaten, dem so genannten „Goodput“. Verglichen werden hier die absoluten Werte, also die nach dem jeweiligen Zeitraum insgesamt übermittelten Daten, sowohl bei Verwendung von UDP, als auch TCP. Zu diesem Zweck wurden die serverseitig gemessenen Werte genommen, da hier die bei TCP anfallenden Retransmits nicht mitgeschnitten werden. Pakete, die bei UDP verloren gehen, werden hier ebenfalls nicht mit erfasst, womit nur die Daten übrig bleiben, die tatsächlich übertragen wurden. Die Ergebnisse sind dabei in Megabyte angegeben.

[Abbildung 9a](#) zeigt den Vergleich des Goodputs bei 60, [Abbildung 9b](#) bei 300 Sekunden. Wie den Grafiken zu entnehmen ist, konnten mit den WireGuard-Konfigurationen einiges mehr an Daten übertragen werden, als mit Tor. Auch ist eine deutliche Abstufung bei unterschiedlicher Anzahl an Peers zu erkennen. Die Stufe zwischen zwei Peers und drei Peers ist dabei größer, als zwischen drei Peers und

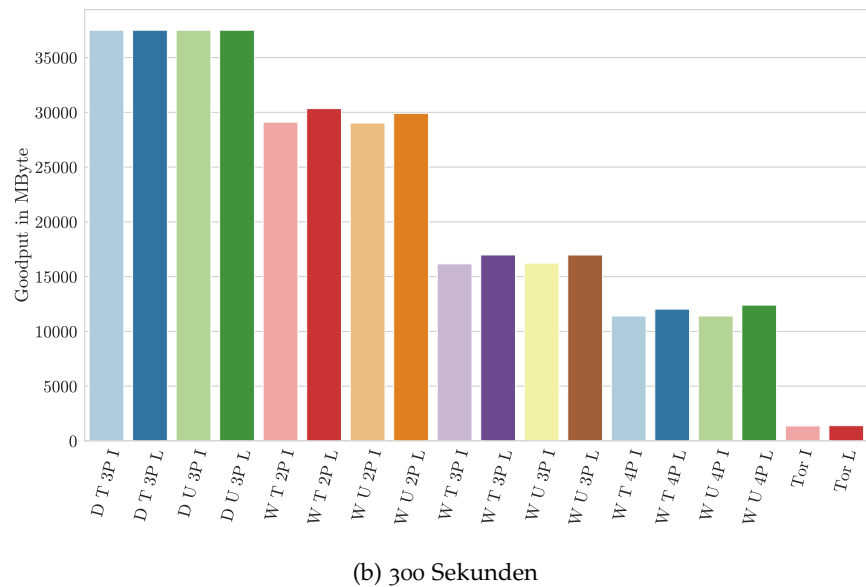
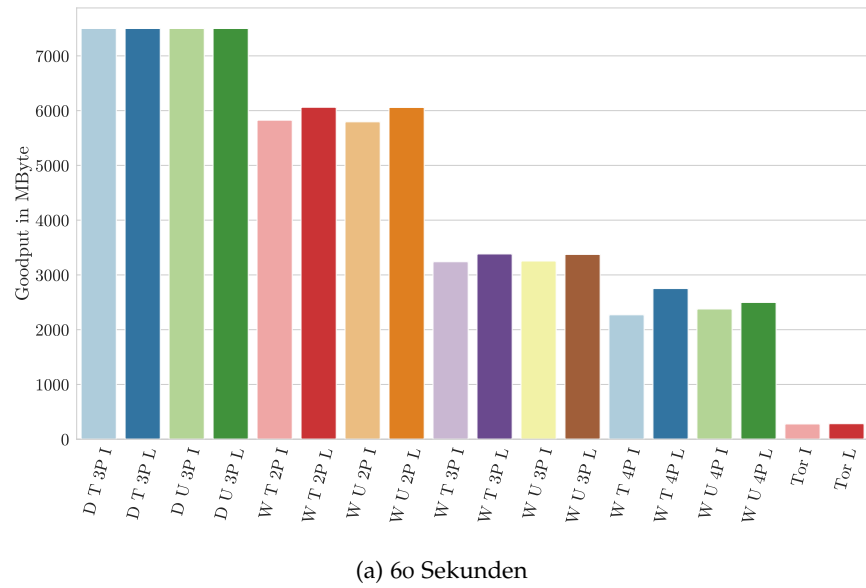


Abbildung 9: Vergleich des Goodputs bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), WireGuard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))



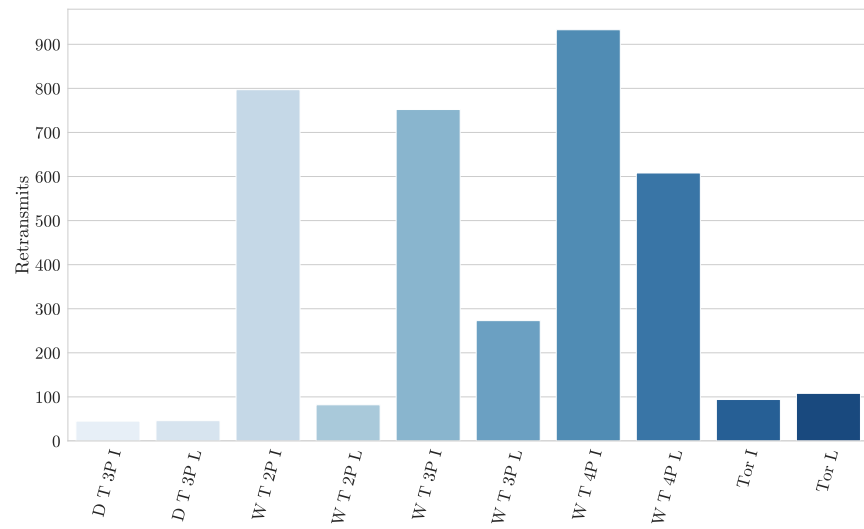
vier Peers. Dies stimmt mit den Messergebnissen der Übertragungsraten überein. Auch lässt sich ein Unterschied bei der Messung unter lokalen und Internet-Bedingungen feststellen. So konnten bei den Setups ohne die Verwendung des Gilbert-Elliott-Modells mehr Daten übertragen werden, als mit. Dies lässt sich dadurch erklären, dass die unter Internet-Bedingungen auftretenden Paketverluste dafür sorgen, dass unter UDP weniger Pakete am Server ankommen und unter TCP manche Pakete erneut geschickt werden müssen, was den selben Effekt hat. Am meisten Daten wurden auch hier wieder mit dem Multihop-Netzwerk ohne WireGuard oder Tor übertragen. Hier ist die Anzahl an gesendeten Daten bei allen vier Setups – TCP lokal, UDP lokal, TCP Internet, UDP Internet – mit einem Wert von etwa 7499,94 MByte bei 60 Sekunden und etwa 37499,92 MByte bei 300 Sekunden, fast identisch. Wie bei der Übertragungsrate liegt hier Ursache wieder in der auf 1 GBit/Sekunde begrenzten Bandbreite, die von allen vier Netzwerk-Konfigurationen vollständig ausgenutzt wurden.

Auch bei diesem Test bildet Tor wieder das Schlusslicht mit der geringsten Anzahl an übertragenen Daten. Interessant ist dabei vor allem zu sehen, dass mit Tor unter lokalen Bedingungen nach 300 Sekunden sogar weniger Daten übertragen werden konnten (ca. 1394 MByte), als mit WireGuard unter Internet-Bedingungen und vier Peers (ca. 2271 MByte) nach 60 Sekunden. Mit anderen Worten: das langsamste WireGuard-Multihop-Netzwerk konnte in nur einem fünftel der Zeit immer noch mehr Daten übertragen, als das schnellste Tor-Netzwerk. Dies zeigt, dass das WireGuard-Multihop-Netzwerk gegenüber Tor einen enormen Effizienz-Vorteil besitzt.

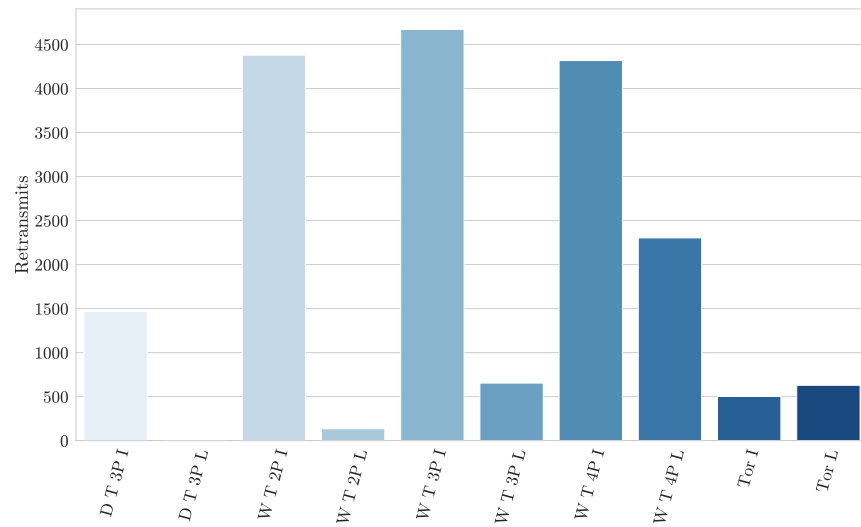
### 5.1.3 Retransmits

Im folgenden wird die Anzahl der Retransmits in den verschiedenen Netzwerk-Setups miteinander verglichen. Da das erneute Senden von verlorenen Paketen eine Funktion ist, die nur TCP unterstützt, werden hier auch nur die Ergebnisse der TCP-Messungen berücksichtigt. Das Hauptaugenmerk bei diesen Messungen liegt vor allem auf dem Unterschied zwischen lokalen und Internet-Bedingungen, da die Emulation mithilfe des Gilbert-Elliott-Modells einen direkten Einfluss auf die Paketverlustquote hat, was unter TCP zu einem erneuten Senden der Pakete führt. Schaut man sich die Daten in [Abbildung 10a](#) und [Abbildung 10b](#) an, wird der zuvor geschilderte Sachverhalt sofort ersichtlich. Bei durchweg allen Tests, mit Ausnahme von Tor, ist die Anzahl an Retransmits bei Übertragung unter Internet-Bedingungen deutlich höher, als bei lokalen Bedingungen.

Auch scheint die Anzahl an Knoten im WireGuard-Netzwerk einen direkten Einfluss auf den Paketverlust zu haben, zumindest unter lokalen Bedingungen, da die Anzahl an Retransmits mit der Zahl der



(a) 60 Sekunden



(b) 300 Sekunden

Abbildung 10: Vergleich der Anzahl an Retransmits bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), Wire-Guard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))

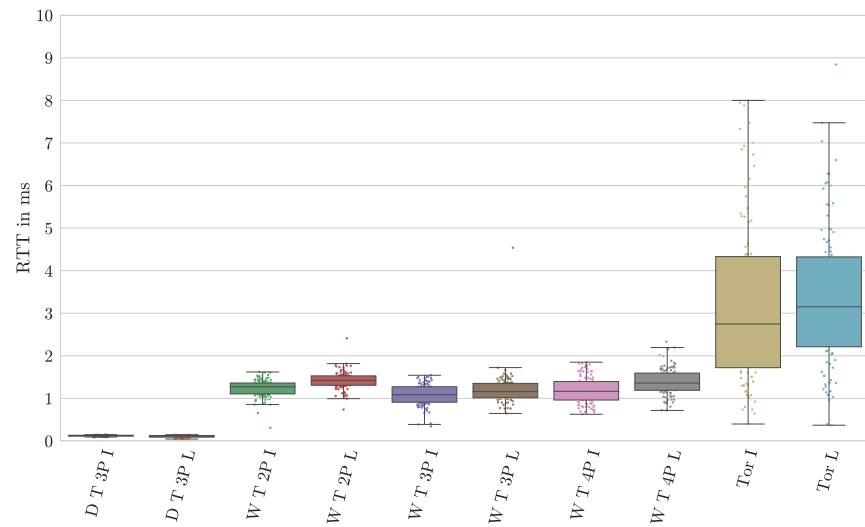
Peers im Netzwerk steigt. Dass die Verwendung von WireGuard oder Tor generell einen Einfluss auf den Paketverlust haben, wird dadurch deutlich, dass das Multihop-Netzwerk unter lokalen Bedingungen die wenigsten, bei dem 5-minütigen Test sogar gar keine Retransmits aufweist. Alles in allem scheint jedoch die Verlustquote bei Tor am stabilsten zu sein, was jedoch keinen nennenswerten Geschwindigkeitsvorteil gegenüber WireGuard einzubringen scheint.

#### 5.1.4 Round-Trip-Time

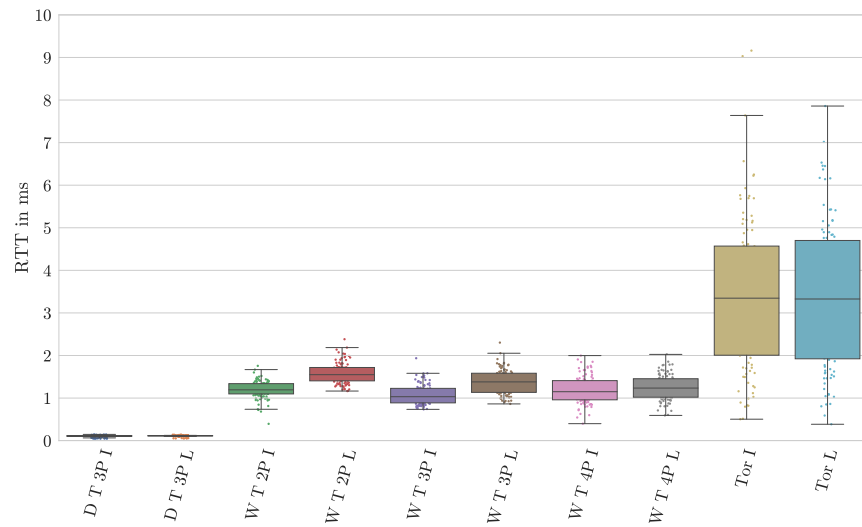
Genau wie bei den Retransmits ist die Round-Trip-Time, also die Zeit, die ein Paket von der Quelle zum Ziel und wieder zurück benötigt, eine rein von TCP unterstützte Kennzahl, weshalb auch hier nur die TCP-Messungen berücksichtigt werden. Die Messergebnisse sind dabei in Millisekunden angegeben. Als Maß wurde hier die durchschnittliche Round-Trip-Time pro Intervall-Schritt über den jeweiligen Zeitraum herangezogen und die Datenreihen in Quantile unterteilt als Box-Diagramm dargestellt. Betrachtet man die in [Abbildung 11a](#) und [Abbildung 11b](#) dargestellten Ergebnisse, wird schnell deutlich, dass das Multihop-Netzwerk mit WireGuard eine geringere Round-Trip-Time als Tor aufweist. Darüber hinaus weisen die Messergebnisse bei einer viel größeren Streuung auf, wobei sich die meisten Ergebnisse zwischen dem ersten Quantil und dem Median befinden. Gerade die Tor-Messung unter Internet-Bedingungen beinhaltet bei einer Testdauer von 60 Sekunden eine sehr große Streuung von etwa 400  $\mu$ s bis knapp 8 ms. Die niedrigste Round-Trip-Time tritt erwartungsgemäß bei dem Multihop-Netzwerk ohne WireGuard oder Tor auf, was zu einem großen Teil auf die fehlende Ver- und Entschlüsselung zurückzuführen ist. Alles in allem scheinen sowohl bei den Messungen über 60, als auch über 300 Sekunden die lokalen Bedingungen im Median eine minimal höhere Round-Trip-Time zu besitzen, als unter Internet-Bedingungen. Dies lässt sich nur dadurch erklären, dass bei den Internet-Bedingungen keine Verzögerungen eingebaut wurden und diese somit keinen direkten Einfluss auf die Round-Trip-Time haben.

Es wird auch schnell deutlich, dass das Hinzufügen weiterer Peers im WireGuard-Setup ebenfalls nur einen sehr geringen Einfluss auf die Ergebnisse hat. Vergleich man z. B. die WireGuard-Messungen bei 60 Sekunden, so ergibt sich unter lokalen Bedingungen ein Median von 1,42 ms bei zwei, 1,15 ms bei drei und 1,36 ms bei vier Peers. In diesem Setup ist also das Netzwerk mit zwei Peers sogar „langsamer“, als mit vier Peers. Das gleiche Verhalten findet sich ebenfalls bei den 300-Sekunden-Messungen, sowie unter Internet-Bedingungen wieder. Da die Differenzen zwischen den Messwerten hier jedoch sehr gering sind, kann es sich dabei auch nur um ein Artefakt der lokalen laufenden Simulation handeln.

*Intervall-Schritte:  
0,6 Sekunden bei  
den 60 Sekunden  
Messungen und 3  
Sekunden bei den  
300 Sekunden  
Messungen.*



(a) 60 Sekunden



(b) 300 Sekunden

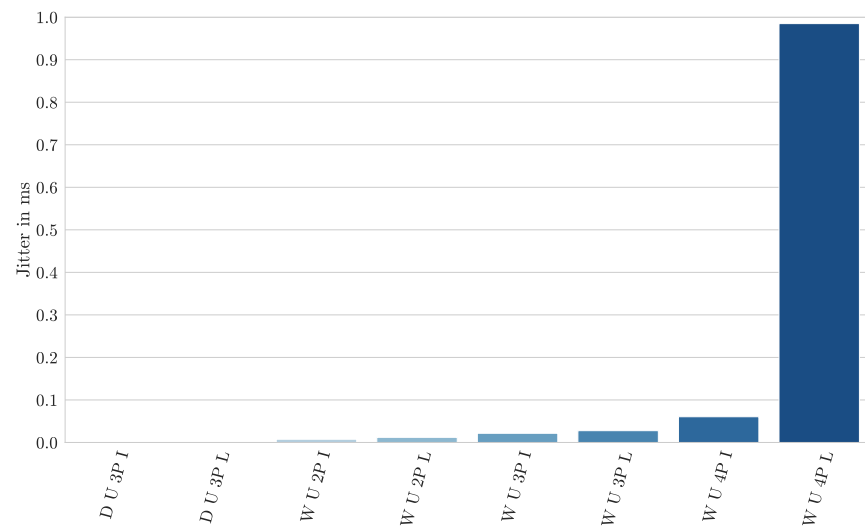
Abbildung 11: Vergleich der Round-Trip-Time in Millisekunden bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), WireGuard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))

### 5.1.5 Jitter

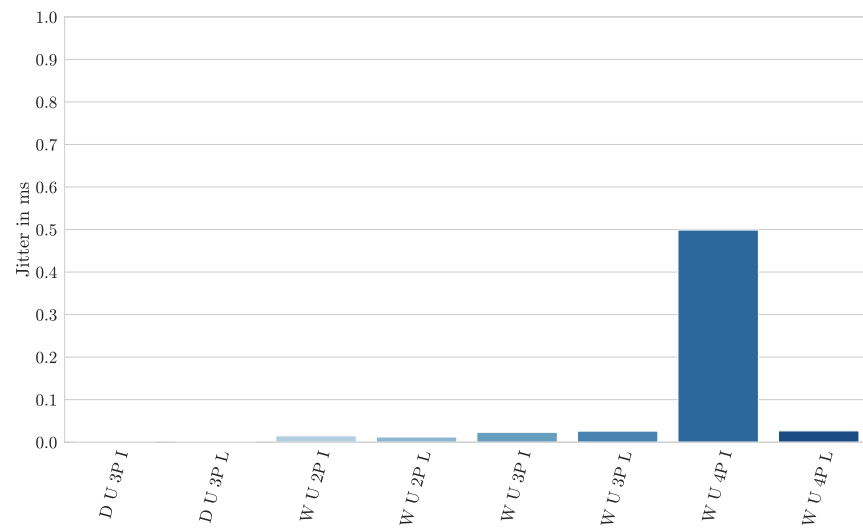
Nachdem zuvor hauptsächlich die Ergebnisse der TCP-Messungen vorgestellt wurden, wird nun auf die Messergebnisse der UDP-Tests eingegangen. Ähnlich wie bei Retransmits und Round-Trip-Time gibt es auch Kennzahlen, die nur bei Verwendung von UDP erhoben werden können. Eine dieser Kennzahlen ist der Jitter, also das zeitliche Taktzittern bei einer Datenübertragung. Da dieser Wert vor allem Echtzeit-Anwendungen wie z. B. VoIP beeinflusst, für welche vornehmlich UDP zur Übertragung genommen wird, wird dieser Wert auch nur bei den UDP-Messungen erfasst. Daher werden auch nur diese Messungen miteinander verglichen, was zur Folge hat, dass hier die Tor-Messungen nicht berücksichtigt werden können, da Tor nur TCP verwendet. Im folgenden werden die durchschnittlichen Jitter-Werte pro Übertragung miteinander verglichen. Die Werte selbst sind dabei in Millisekunden angegeben. Die Ergebnisse werden auch hier wieder anhand der Testdauer nach 60 Sekunden in [Abbildung 12a](#) und 300 Sekunden in [Abbildung 12b](#) getrennt dargestellt.

Was an [Abbildung 12a](#) sofort auffällt, ist der vergleichsweise sehr hohe Wert bei dem Setup WireGuard, UDP, 4 Peers lokale Bedingungen. Während die anderen Tests Werte unter 0,1 ms aufweisen, liegt hier der durchschnittliche Jitter bei fast einer Millisekunde. Dies lässt auf hohe Schwankungen im Übertragungstakt schließen, welche wohl vor allem bei diesem Setup auftreten. Vergleicht man dies jedoch mit den Werten aus [Abbildung 12b](#), ergibt sich ein vollkommen anderes Bild. Hier ist der Jitter bei WireGuard, 4 Peers und Internet-Bedingungen wesentlich höher als bei den anderen Konfigurationen, während der Wert bei gleichem Setup aber unter lokalen Bedingungen sehr gering ist. Dies legt die Vermutung nahe, dass es sich hierbei nur um ein zufälliges Artefakt der Übertragung handelt. Um diesen Verdacht zu untersuchen, wurde speziell für das Setup WireGuard, UDP, 4 Peers und lokale Bedingungen eine Messreihe mit 30 60-Sekunden-Messungen durchgeführt. Anhand dieser Stichprobe wurden anschließend für den Jitter Konfidenzintervalle mit einem Signifikanzniveau von 0,01 berechnet. Und tatsächlich ergab sich für das oben genannte Setup ein Konfidenzintervall mit einer Untergrenze von rund 0,011 ms und einer Obergrenze von rund 0,227 ms. Das bedeutet, dass 99 % der gemessenen Werte innerhalb dieser Intervallgrenzen liegen. Da der in [Abbildung 12a](#) dargestellte Wert von fast einer Millisekunde jedoch weit über der Obergrenze des Konfidenzintervalls liegt, kann hierbei von einem zufälligen Artefakt ausgegangen werden.

Alles in allem wird auch hier schnell ersichtlich, dass das Multihop-Netzwerk ohne WireGuard und Tor den geringsten Jitter aufweist, welcher sowohl bei lokalen und Internet-Bedingungen mit 60 Sekunden, als auch 300 Sekunden nur knapp über 0 liegt. Darüber hinaus



(a) 60 Sekunden



(b) 300 Sekunden

Abbildung 12: Vergleich des Jitters in Millisekunden bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), Wire-Guard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))

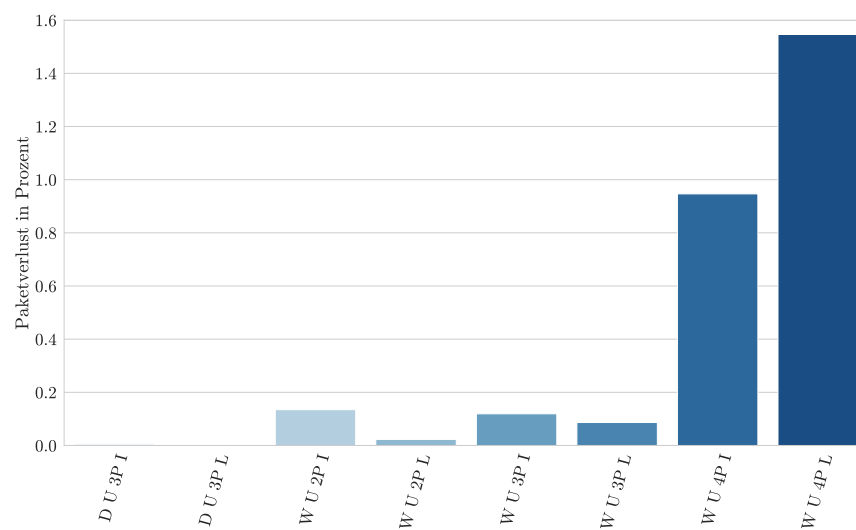
steigt er mit zunehmender Anzahl an Peers minimal an und hat bei vier Peers seinen höchsten Wert.

#### 5.1.6 Paketverlust

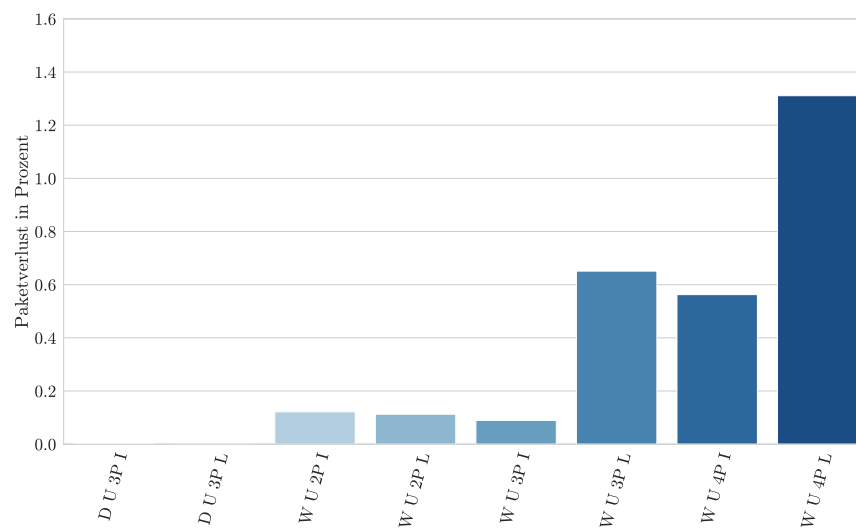
Abschließend werden die Ergebnisse des letzten gemessenen UDP-Features betrachtet. Da Paketverluste bei UDP nicht durch erneutes Senden ausgeglichen werden, dient hier die Paketverlust-Quote als geeignete Kennzahl, um die Qualität eines Netzwerkes zu bewerten. Die in diesem Abschnitt vorgestellten Ergebnisse sind in Prozent angegeben und beschreiben somit den Anteil der verlorenen Paketen an der gesamten Übertragung. Auch werden hier wieder die Ergebnisse nach Testdauer in 60 Sekunden und 300 Sekunden getrennt.

Was auch bei den in [Abbildung 13a](#) und [Abbildung 13b](#) dargestellten Ergebnissen direkt auffällt, ist eine relativ hohe Paketverlust-Quote bei den WireGuard-Tests mit 4 Peers, insbesondere im Setup mit lokalen Bedingungen. Vergleicht man dies mit der gleichen Konfiguration unter Internet-Bedingungen, ergibt sich, dass die Verwendung des Gilbert-Elliot-Modells, welches Paketverluste in die Datenübertragung einbaut, dennoch eine geringere Anzahl an verlorenen Paketen aufweist, als ohne Model. Wie auch zuvor kann es sich hierbei wieder um einen Ausreißer handeln. Um das zu überprüfen, wurde als Referenz erneut eine Messreihe sowohl für die lokalen, als auch die Internet-Bedingungen durchgeführt. Diese bestand wie im Abschnitt zuvor ebenfalls aus 30 Messungen mit je 60 Sekunden. Hierbei ergaben sich Konfidenzintervalle von 0,754 % bis 1,981 % für die lokalen Bedingungen und von 0,515 % bis 1,229 % für die Internet-Bedingungen. Somit scheinen die beiden Messwerte mit 1,546 % (lokal) und 0,946 % (Internet) tatsächlich noch innerhalb der jeweiligen Intervallgrenzen zu liegen, womit ein einmaliger Ausschlag der Werte auszuschließen ist. Da dieses Phänomen selbst bei mehreren Messungen immer wieder aufzutreten scheint, kann also davon ausgegangen werden, dass der Grund im Testaufbau selbst zu finden ist. Interessant ist hierbei auch zu berücksichtigen, dass die Anzahl an verlorenen Paketen bzw. Retransmits bei Verwendung von TCP nicht so hoch ist.

Bei den anderen Messergebnissen ist das Bild ein etwas anderes. In allen Szenarien weisen die Tests ohne WireGuard und Tor die geringste Anzahl an Paketverlusten auf. Hier liegt sogar der Prozentwert für die lokalen Bedingungen bei 60 und 300 Sekunden jeweils bei 0. Die anderen WireGuard-Tests zeigen ähnliche Ergebnisse. Das Internet-Modell erzeugte in der Regel mehr Paketverluste, als die lokalen Bedingungen. Eine weitere Ausnahme bildet das drei-Peer-Netzwerk bei einer Messdauer von 300 Sekunden. Auch hier ist der Anteil an verlorenen Paketen bei lokalen Bedingungen höher, als bei Internet. Die Auswertung einer weiteren Messreihe ergab in diesem Fall, dass



(a) 60 Sekunden



(b) 300 Sekunden

Abbildung 13: Vergleich der Paketverlustquote in Prozent bei einer Messzeit von 60 und 300 Sekunden. Die Beschriftung der X-Achse ist folgendermaßen zu interpretieren: D|W|Tor (Direkt (D), WireGuard (W) oder Tor), T|U (TCP (T) oder UDP (U)), 2|3|4P (Anzahl der Peers), I|L (Internet-Bedingungen (I) oder lokale Bedingungen (L))



das Konfidenzintervall für die lokale Messung zwischen 0,214 % und 0,573 % liegt, sowie zwischen 0,335 % und 0,852 % für die Messung unter Internet-Bedingungen. Daraus folgt, dass sich beide in [Abbildung 13b](#) dargestellten Werte außerhalb der Intervallgrenzen befinden. Somit scheint es sich auch hierbei wieder um einen zufälligen Ausreißer zu handeln.

## 5.2 DISKUSSION

Nachdem zuvor die Ergebnisse der Tests präsentiert wurden, soll in diesem Abschnitt darüber diskutiert werden. Alles in allem haben die Tests gezeigt, dass ein WireGuard-Multihop-Netzwerk, das Onion-Routing umsetzt, in der Tat um einiges effizienter als Tor ist. Vor allem beim Vergleich der Übertragungsraten und der tatsächlich gesendeten Datenmengen konnte aufgezeigt werden, dass selbst mit der „langsamsten“ WireGuard-Konfiguration dieser Testreihe in nur einer Minute mehr Daten übertragen werden konnten, als mit der schnellsten Tor-Konfiguration in fünf Minuten. Die Tests haben auch gezeigt, dass die Verwendung weiterer Peers einen recht starken Einfluss auf die Geschwindigkeit und auch die Stabilität des Netzwerkes hat. Vor allem beim WireGuard-Netzwerk mit vier Peers und bei der Verwendung von UDP zur Übertragung traten immer wieder Anomalien auf, die an dieser Stelle zumindest teilweise geklärt werden konnten. Nun steht jedoch immer noch die Frage im Raum, ob es sich beim vorgestellten WireGuard-Multihop-Netzwerk um eine, zumindest in der Zukunft, mögliche Alternative zu Tor handeln kann. Hierzu muss neben der Effizienz noch ein weiterer wichtiger Faktor in Betracht gezogen werden: die Anonymität.

Wie genau die Anonymität in Tor und in einem VPN gewährleistet wird, wurde bereits in einem vorherigen Kapitel erläutert. In beiden Fällen entsteht die Anonymität durch das Maskieren der eigenen IP-Adresse. Bei einem VPN hängt die Sicherheit und Anonymität zusätzlich noch vom verwendeten Protokoll ab. WireGuard ist zwar, wie gezeigt wurde, ein recht schnelles VPN-Protokoll, aber auch noch ein sehr junges. Auch wenn es durch seine Schlankheit und die Verwendung modernster Kryptographie-Verfahren relativ sicher erscheint, kann es etwaige Schwachstellen besitzen, die zum aktuellen Zeitpunkt noch nicht zu erahnen sind. Die Anonymität in Tor hängt vor allem von Aufbau, Größe und Verteilung des Netzwerkes ab [8, 17, 25]. Je weniger Tor-Knoten es gibt und je mehr davon unter Kontrolle der selben Personen oder Instanzen sind, desto leichter ist es, die Kommunikationsteilnehmer zu identifizieren. Der größte Angriffsvektor ist hier, dass sowohl der Entry-Guard, als auch der Exit-Node in der Hand von ein und der selben Person sind. In diesem Fall lässt sich mit Traffic-Analysen der Netzwerkverkehr analysieren und bestimmten Individuen zuordnen. Denn eines muss bei Tor klar sein:

die eigene IP-Adresse ist nicht komplett anonymisiert, da der Entry-Guard diese kennt. Es wurde bereits erläutert, dass der Zugriff auf Tor über einen zusätzlichen VPN-Dienst dieses Problem behebt, das aber wieder auf Kosten der ohnehin schon eher geringen Performance geht. Auch muss angemerkt werden, dass Tor die Übertragung nur bis zum Exit-Node verschlüsselt. Wird für die Datenübertragung selbst keine Verschlüsselung wie beispielsweise TLS verwendet, kann der Exit-Node den gesamten Datenverkehr auslesen, was natürlich großes Vertrauen gegenüber dem Exit-Node bedingt.

Der zuvor beschriebene Angriffsvektor bezüglich der zeitgleichen Kontrolle von Entry- und Exit-Node ist auch im WireGuard-Multihop-Netzwerk die größte Gefahr für die Anonymität. Hier lassen sich jedoch zusätzliche Maßnahmen besser umsetzen, da die Performance-Einbußen nicht so schwer ins Gewicht fallen. Weil die Verwendung von mehr als drei Knoten bzw. Peers im beschriebenen Szenario keine zusätzliche Sicherheit gewährt, kann also auch auf die ohnehin viel performantere Konfiguration mit drei Peers zurückgegriffen werden. Auch wenn für dieses Angriffs-Szenario zwei Knoten ausreichen, gewährleisten drei immer noch eine bessere Anonymität, da man bei zwei Knoten direkt sieht, welcher andere Knoten noch verwendet wird. Somit und mit den Ergebnissen der Tests ergibt sich, dass das Netzwerk mit drei Peers die beste Mischung aus Anonymität, Geschwindigkeit und Stabilität besitzt. Auch hat der Vergleich von UDP und TCP kaum nennenswerte Unterschiede in Bezug auf die Performance hervorgebracht. Nicht zuletzt liegt das vermutlich daran, dass WireGuard zur Übertragung UDP verwendet, was beim Tunneln TCP gegenüber einen Geschwindigkeitsvorteil bringt. Es können also ohne Probleme sowohl UDP-basierte als auch TCP-basierte Anwendungen ihre Daten durch das WireGuard-Multihop-Netzwerk tunneln. Bei Tor wird die Verwendung von UDP-basierten Anwendungen hingegen standardmäßig nicht unterstützt [8]. Dies muss bei Bedarf durch externe Services – z. B. durch Tunneln – umgesetzt werden. Und auch dann ist es aufgrund der ausschließlichen Verwendung von TCP in Tor nicht empfehlenswert.

## VERWANDTE ARBEITEN

---

Nach aktuellem Wissensstand gibt es keine Projekte, die sich ebenfalls mit der Implementierung oder der Evaluierung von Onion-Routing-Netzwerken mithilfe von VPNs beschäftigen. Es gibt jedoch eine ähnliche Arbeit des Instituts für Informationssysteme an der Hochschule Hof in Zusammenarbeit mit der Universität Luxemburg, die sich mit der Evaluierung der Performance von kaskadierenden VPN-Netzwerken befasst. [22] Bei diesem Versuch wurden die beiden VPN-Protokolle WireGuard und OpenVPN hinsichtlich ihrer Performanz in einer kaskadierenden VPN-Umgebung miteinander verglichen. Auch wurden Tests mit einer direkten Multi-Hop-Verbindung zwischen den Endgeräten ohne VPN-Protokolle als Referenz-Messung durchgeführt. Der Aufbau bestand aus zwei Endpunkten und drei Knoten mit gesicherten VPN-Tunneln zwischen den einzelnen Geräten bei den Messungen mit VPN-Protokoll. Dabei griff das Forschungsteam ebenfalls auf Linux-Network-Namespaces zurück, um die Netzwerkgeräte zu simulieren. Bei den Messungen kamen sowohl UDP als auch TCP als Übertragungsprotokoll zum Einsatz. Zudem wurden die Tests sowohl unter lokalen Bedingungen, also ohne emulierte Netzwerk-Ereignisse wie Paket-Verlust oder Delays, als auch unter Internet-Bedingungen, also mit emulierten Netzwerk-Ereignissen, durchgeführt. Dabei ergab sich, dass WireGuard unter TCP um 39,6 % schneller war, als OpenVPN. Lediglich unter lokalen Bedingungen war OpenVPN etwas effizienter als WireGuard, was als Nebeneffekt der optimalen Testumgebung angesehen wurde. Grundsätzlich waren die Messungen unter lokalen Bedingungen effizienter als die Messungen unter emulierten Bedingungen. Bei den anderen Messungen ergab sich kein signifikanter Unterschied. [22]

Das Konzept kaskadierender VPNs wird auch von manchen VPN-Anbietern angeboten. Oft werden dabei zwei VPN-Server hintereinander geschaltet, was als *Double VPN* <sup>1</sup> bezeichnet wird. Unabhängig von den oben vorgestellten Ergebnissen bieten einige VPN-Anbieter wie z. B. NordVPN <sup>2</sup> auch so genanntes *Onion over VPN* an, um die Sicherheit und Anonymität noch zusätzlich zu erhöhen. Bei diesem Ansatz wird auf das Tor-Netzwerk über einen zusätzlichen VPN-Dienst zugegriffen. Einer der Haupt-Nachteile von Tor ist, wie bereits erwähnt, dass es zwar die Identität der Anwenders verschleiert, jedoch nicht den Datenverkehr verschlüsselt, der das Netzwerk verlässt. Dem kann durch Onion over VPN entgegengewirkt werden, da

<sup>1</sup> Siehe <https://nordvpn.com/de/features/double-vpn/>

<sup>2</sup> Siehe <https://nordvpn.com/de/features/onion-over-vpn/>

die Daten durch den VPN-Dienst zusätzlich verschlüsselt durch Tor und das Internet übertragen werden. Nachteil hierbei sind jedoch die zusätzlichen Performance-Einbußen.

## FAZIT UND ZUKÜNFTIGE ARBEIT

---

Abschließend lässt sich sagen, dass die Ergebnisse dieser Arbeit die Hypothese bestätigt haben, dass ein WireGuard-Multihop-Netzwerk bei gleichen Bedingungen und gleicher Anzahl an Knoten in der Tat effizienter ist, als Tor. Ein Vergleich der verschiedenen Netzwerk-Konfigurationen untereinander zeigte auch, wie stark sich die Anzahl der Peers auf die Datenübertragung auswirkt. Um durch die Netzwerk-Emulation potentielle, zufällige Artefakte auszuschließen, wurden ebenfalls Messungen unter „perfekten“ Bedingungen durchgeführt. Diese bestätigten, bis auf ein paar Ausnahmen, die Gültigkeit der Messergebnisse. Die Ausnahmen konnten jedoch mithilfe statistischer Methoden als zufällige Ausfälle identifiziert werden. Grundsätzlich muss man jedoch beachten, dass es sich hierbei nur um eine erste Simulation handelt und dies bedeutet, dass unter Umständen andere Ergebnisse unter Realbedingungen in einem echten Netzwerk erzielt werden könnten. Dies wird vor allem durch die Ergebnisse der direkten Tests ohne WireGuard oder Tor bestätigt, welche in einem realen Netzwerk ebenfalls nicht so extreme Geschwindigkeiten erreicht hätten, wie in dieser Testreihe. Die Übertragung über ein tatsächliches Medium wie Kupfer oder Funk über evtl. weite Entfernungen ist ein Faktor, der bei der Datenübertragung nicht außer Acht gelassen werden darf. Weitere Experimente mit echter Hardware unter realen Bedingungen könnten in Zukunft durchgeführt werden, um die Ergebnisse dieser Arbeit zu überprüfen. Dennoch lassen die hier vorgestellten Daten darauf schließen, dass bei gleichen Bedingungen ein WireGuard-basiertes Onion-Routing-Netzwerk um einiges schneller ist, als Tor. Es wurde ebenfalls aufgezeigt, dass es im WireGuard-Netzwerk kaum Unterschiede zwischen dem Einsatz von UDP und TCP bei der Übertragung gibt. Das bedeutet, dass somit UDP- als auch TCP-basierte Protokolle zusammen mit diesem Setup ohne Probleme zum Einsatz kommen können.

Die Netzwerke in dieser Arbeit wurden mithilfe von IP-Table-Rules, Linux-Network-Namespace sowie dem Kommandozeilen-Tool von WireGuard aufgebaut. Für die Zukunft wäre es jedoch wünschenswert, die Möglichkeit des Onion-Routings direkt in WireGuard zu implementieren. Hierzu müsste es eine Funktion geben, mit der WireGuard-Pakete innerhalb eines Gerätes geroutet und somit mehrfach in das gleiche Interface geleitet werden können. Die Unterstützung von Onion-Routing ist auch ein Punkt auf der To-Do-Liste der WireGuard-Entwickler<sup>1</sup>. Auch der Aufbau des Netzwerkes bzw. der Routen durch das Netz-

---

<sup>1</sup> Siehe <https://www.wireguard.com/todo#onion-routing>

werk müsste, ähnlich wie bei Tor, dynamisch erfolgen. Sämtliche Routen und Netzwerke, die in dieser Arbeit getestet wurden, sind statisch aufgebaut worden.

## ANHANG

---

### A.1 WIREGUARD-INTERFACE-KONFIGURATION

Konfigurationen der WireGuard-Interfaces des Clients und der Peers für die WireGuard-Netzwerke. Da die Schlüssel bei jedem Aufbau neu generiert werden, werden hier Platzhalter verwendet.

#### A.1.1 Interfaces bei zwei Peers

##### **wgo:**

IP-Adresse	<i>10.66.202.12</i>
Öffentlicher Schlüssel	<i>client.public</i>
Privater Schlüssel	<i>client.private</i>
UDP Port	<i>49415</i>

##### **Peers**

Öffentlicher Schlüssel	Erlaubte Adressen	Endpunkt
<i>peer1.public</i>	<i>0.0.0.0\0</i>	<i>185.65.135.130:51820</i>
<i>peer2.public</i>	<i>10.64.0.1\32,</i> <i>185.145.152.120\32</i>	<i>185.213.154.68:51820</i>

Tabelle 4: Interface-Konfiguration von Interface wgo im Client bei zwei Peers.

##### **wg-peer1:**

IP-Adresse	<i>10.66.202.14</i>
Öffentlicher Schlüssel	<i>peer1.public</i>
Privater Schlüssel	<i>peer1.private</i>
UDP Port	<i>51820</i>

##### **Peers**

Öffentlicher Schlüssel	Erlaubte Adressen	Endpunkt
<i>client.public</i>	<i>10.66.202.12\32</i>	<i>185.213.154.68:51820</i>

Tabelle 5: Interface-Konfiguration von Interface wg-peer1 in Peer 1 bei zwei Peers.





## LITERATUR

---

- [1] Jean-Philippe Aumasson und Daniel J Bernstein. "SipHash: a fast short-input PRF". In: *International Conference on Cryptology in India*. Springer. 2012, S. 489–508.
- [2] Jean-Philippe Aumasson, Willi Meier, Raphael C-W Phan und Luca Henzen. "BLAKE2". In: *The Hash Function BLAKE*. Springer, 2014, S. 165–183.
- [3] Daniel J Bernstein. "The Poly1305-AES message-authentication code". In: *International workshop on fast software encryption*. Springer. 2005, S. 32–49.
- [4] Daniel J Bernstein. "Curve25519: new Diffie-Hellman speed records". In: *International Workshop on Public Key Cryptography*. Springer. 2006, S. 207–228.
- [5] Daniel J Bernstein u. a. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Bd. 8. 2008, S. 3–5.
- [6] Claude Castelluccia. "Behavioural Tracking on the Internet: A Technical Perspective". In: *European Data Protection: In Good Health?* Hrsg. von Serge Gutwirth, Ronald Leenes, Paul De Hert und Yves Pouillet. Dordrecht: Springer Netherlands, 2012, S. 21–33. ISBN: 978-94-007-2903-2. DOI: [10.1007/978-94-007-2903-2\\_2](https://doi.org/10.1007/978-94-007-2903-2_2). URL: [https://doi.org/10.1007/978-94-007-2903-2\\_2](https://doi.org/10.1007/978-94-007-2903-2_2).
- [7] Lance Cottrell, Peter Palfrader und Len Sassaman. "Mixmaster Protocol Version 2< draft-moeller-v2-01. txt". In: *Online specification* (2003).
- [8] Roger Dingledine, Nick Mathewson und Paul Syverson. *Tor: The second-generation onion router*. Techn. Ber. Naval Research Lab Washington DC, 2004.
- [9] Jason A Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel." In: *NDSS*. 2017, S. 1–12.
- [10] Paul Ferguson und Geoff Huston. *What is a VPN?* 1998.
- [11] David Goldschlag, Michael Reed und Paul Syverson. "Onion Routing". In: *Commun. ACM* 42.2 (Feb. 1999), 39–41. ISSN: 0001-0782. DOI: [10.1145/293411.293443](https://doi.org/10.1145/293411.293443). URL: <https://doi.org/10.1145/293411.293443>.
- [12] C. Gulcu und G. Tsudik. "Mixing E-mail with Babel". In: *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. 1996, S. 2–16. DOI: [10.1109/NDSS.1996.492350](https://doi.org/10.1109/NDSS.1996.492350).

- [13] Gerhard Hasslinger und Oliver Hohlfeld. "The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet". In: *14th GI/ITG Conference - Measurement, Modelling and Evaluation of Computer and Communication Systems*. 2008, S. 1–15.
- [14] Andreas Huelsing, Kai-Chun Ning, Peter Schwabe, Florian Weber und Philip R. Zimmermann. "Post-quantum WireGuard". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, S. 304–321. DOI: [10.1109/SP40001.2021.00030](https://doi.org/10.1109/SP40001.2021.00030).
- [15] Rolf Jagerman, Wendo Sabee, Laurens Versluis, Martijn de Vos und Johan Pouwelse. "The fifteen year struggle of decentralizing privacy-enhancing technology". In: *arXiv preprint arXiv:1404.4818* (2014).
- [16] Rob Jansen, Kevin S Bauer, Nicholas Hopper und Roger Dingledine. "Methodically Modeling the Tor Network." In: *CSET*. 2012.
- [17] Ishan Karunanayake, Nadeem Ahmed, Robert Malaney, Rafiqul Islam und Sanjay K. Jha. "De-anonymisation attacks on Tor: A Survey". In: *IEEE Communications Surveys & Tutorials* (2021), 1–1. ISSN: 2373-745X. DOI: [10.1109/comst.2021.3093615](https://doi.org/10.1109/comst.2021.3093615). URL: <http://dx.doi.org/10.1109/COMST.2021.3093615>.
- [18] David Koblas. "SOCKS". In: *UNIX Security III Symposium* (1992), S. 77–83.
- [19] Hugo Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *Advances in Cryptology – CRYPTO 2010*. Hrsg. von Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 631–648. ISBN: 978-3-642-14623-7.
- [20] Manfred Lipp. *VPN-virtuelle private Netzwerke: Aufbau und Sicherheit*. ISBN: 3827322529.
- [21] Gareth Owen und Nick Savage. "Empirical analysis of Tor hidden services". In: *IET Information Security* 10.3 (2016), S. 113–118.
- [22] Sebastian Pahl, Florian Adamsky, Daniel Kaiser und Thomas Engel. "Poster: Evaluating Cascading-VPN Performance". In: ().
- [23] Trevor Perrin. "The Noise protocol framework". In: *PowerPoint Presentation* (2018).
- [24] Michael G Reed, Paul F Syverson und David M Goldschlag. "Anonymous connections and onion routing". In: *IEEE Journal on Selected areas in Communications* 16.4 (1998), S. 482–494.
- [25] Robin Snader und Nikita Borisov. "A Tune-up for Tor: Improving Security and Performance in the Tor Network." In: *ndss*. Bd. 8. 2008, S. 127.
- [26] E. Snowden. *Permanent Record*. Henry Holt und Company, 2019. ISBN: 9781250237248. URL: <https://books.google.de/books?id=0XCcDwAAQBAJ>.

- [27] C. Wegener, T. Milde und W. Dolle. *Informationssicherheits-Management: Leitfaden für Praktiker und Begleitbuch zur CISM-Zertifizierung*. Xpert.press. Springer Berlin Heidelberg, 2016. ISBN: 9783662491676. URL: <https://books.google.de/books?id=rrSiDQAAQBAJ>.



## ERKLÄRUNG DER URHEBERSCHAFT

---

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

---

Ort, Datum

Unterschrift