Effizienter Netzzugang mit Multipath TCP

Jannik Seemann

6. September 2020

1 Einleitung

Moderne Endgeräte verfügen oft über mehrere Netzwerkschnittstellen. Zum Beispiel unterstützen Smartphones und manche Notebooks sowohl WLAN als auch Mobilfunk. Das weit verbreitete Transportprotokoll Transmission Control Protocol (TCP) profitiert von dieser Eigenschaft nicht [1]. Eine TCP Verbindung kann auf Client und Serverseite nur eine Netzwerkschnittstelle verwenden. Multipath TCP (MPTCP) ist ein vielversprechender Ansatz, wie bei existierenden Netzwerken der Durchsatz gesteigert und Redundanz geschaffen werden kann [2]. Bei mobilen Endgeräten wie Smartphones kann beispielsweise der Durchsatz von der WLAN und Mobilfunkanbindung kombiniert werden. Sollte dabei zum Beispiel die WLAN Verbindung wegfallen, kann die Mobilfunkverbindung weiterhin genutzt werden, ohne dass eine neue Verbindung aufgebaut werden muss. Um Multipath TCP verwenden zu können, muss dies von beiden Kommunikationspartnern unterstützt werden. Nicht immer sind beide Akteure modifizierbar. Im ersten Teil der Arbeit werden geeignete Bereitstellungswege für Multipath TCP in vorhandene Netzwerke vorgestellt. Ein Kriterium ist, dass so wenig Modifikation an vorhandenen Komponenten wie möglich notwendig sein soll. Daher werden auf Proxyserver basierte Systeme betrachtet. Im Fokus steht in dieser Arbeit, dass mobile Endgeräte davon profitieren.
Im zweiten Teil der Arbeit werden virtuelle Testbeds entworfen, mit welchen Multipath TCP unterstützende Systeme gut evaluiert werden können. Mithilfe der Testbeds sollen die Bereitstellungsmöglichkeiten für Multipath TCP evaluiert werden.

2 Verwandte Arbeiten

Als Referenz zu Multipath TCP dient das RFC 6824 [2], welche alle grundlegende Bestandteile der Technologie beschreibt. Es gibt Arbeiten welche sich mit praxisorientierten Problemen beschäftigen, wie das Verhalten bei mobilen Netzwerken mit Fokus auf dem Verhalten bei Handover: *Exploring Mobile/WiFi Handover with Multipath TCP* [3] oder der Einsatz in einem Rechenzentrum: *Improving Datacenter Performance and Robustness with Multipath TCP* [4]. In *Multipath TCP Deployments* [5] werden Optionen zur Bereitstellung von MPTCP vorgestellt. Dabei werden Möglichkeiten eines socks Proxies aufgezeigt.
Besonders interessant sind Arbeiten, welche sich mit Testbeds befassen. In *Mininet/Netem Emulation Pitfalls A Multipath TCP Scheduling Experience* [6] wird vor möglichen Abweichungen zwischen virtuellen Testbeds und realen Netzwerken gewarnt. In *Multi-Path TCP in Real-World Setups – An Evaluation in the NORNET CORE Testbed* [7] wird der Einsatz von Multipath TCP in einem großen physischem Testbed vorgestellt.
Neben den eher praktischen Arbeiten wird sich auch mit algorithmischen Fragestellungen wie Scheduling oder Congestion Control beschäftigt. Dazu gehören die Arbeiten *Experimental Evaluation of Multipath TCP Schedulers* [8], welches die verschiedenen Scheduling Algorithmen miteinander experimentell vergleicht oder *Performance Comparison of Congestion Control Strategies for Multi-Path TCP in the NORNET Testbed* [9] was das Verhalten der verschiedenen Congestion Control Algorithmen gegenüber stellt.

3 Verwandte Kommerzielle Lösungen

Neben den eher akademischen Arbeiten wird Multipath TCP auch in kommerziellen Produkten eingesetzt. Für Endkunden gibt es von manchen Internet Service Provider die Option über Multipath TCP die Bandbreite zu erhöhen.

3.1 Korea Telecom - GIGA Path

Die *Korea Telecom* stellte 2014 einen Service mit dem Name *GIGA Path* vor, mit welchem der Durchsatz durch Kombination von Mobilfunk und WLAN verbessert werden soll. Verfügbar ist der Service für ausgewählte Smartphones, für welche MPTCP Patches bereitgestellt werden. Die Endgeräte bauen zu einem socks Proxy eine MPTCP Verbindung auf. Der Proxy kommuniziert über eine reguläre TCP Verbindung mit dem Zielserver. Die Benutzer

müssen um das Angebot zu nutzen eine Applikation installieren. Dabei legen die Benutzer fest zu welchen Domains über den MPTCP Proxy kommuniziert werden soll. Ausschließlich der Verkehr zu den eingetragenen Domains werden über den Proxy geleitet. Dem Benutzer stehen verschiedene Optionen zu Verfügung, auf was optimiert werden soll: maximaler Durchsatz, Verkehr über Mobilfunk minimieren und WLAN bevorzugen. Bei der letzten Variante ist Multipath TCP von Vorteil, da beim Wechsel von Netzwerken die bestehenden Verbindungen nicht neu aufgebaut werden müssen [10].

3.2 Tessares Hybrid Access Networks, Swisscom Internet Booster

Im Gegensatz zu dem Deployment der Korea Telecom kommen bei diesen Lösungen eine Multipath TCP Verbindung nicht zwischen Endgerät und Router, sondern zwischen dem Router und einem Proxy Server zum Einsatz. Dieser Lösungsansatz ist im Vergleich zu der Lösung von Korea Telecom darauf ausgelegt, die Netzwerkanbindung stationär zu verbessern. Es wird DSL mit Mobilfunk kombiniert. Die Endgeräte des Benutzers müssen das Multipath TCP Protokoll nicht unterstützen. Der Schweizer ISP *Swisscom* bietet so eine Lösung mit der Bezeichnung *Internet Booster* für Endkunden an [11]. Die Firma *Tessares* verkauft an Telekommunikationsabieter unter der Bezeichnung *Hybrid Access Networks* MPTCP Proxy Lösungen nach diesem Prinzip[12].

4 Das vorhandene Netzwerk

In dieser Arbeit wird ein Client-Server Netzwerk betrachtet. Ein mobiles Endgerät verfügt über zwei Netzwerkschnittstellen. Das Gerät kann sich entweder über WLAN oder über Mobilfunk mit Netzwerken verbinden. Ein einfaches, privates Netzwerk besteht aus Servern, welche über ein Switch mit einem Router verbunden sind. Durch einen Internet Service Provider ist der Router mit dem Internet verbunden. Das Endgerät will mit einem Webserver im privaten Netzwerk kommunizieren.

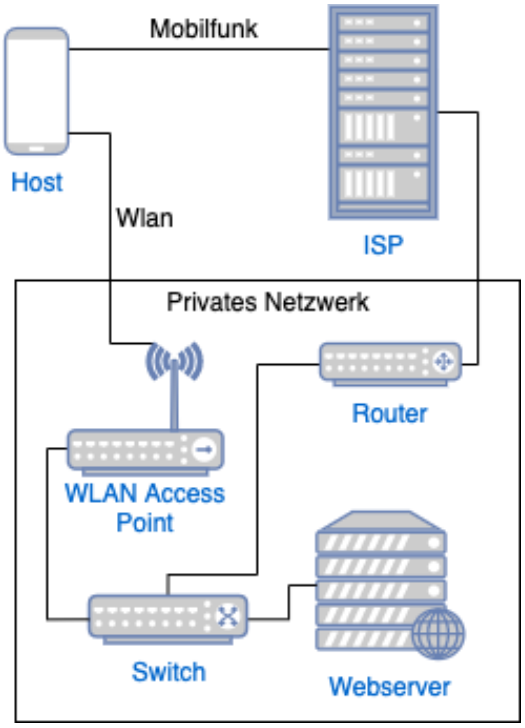


Abbildung 1: Die Netzwerktopologie

Als Kommunikationsprotokoll kommt überwiegend TCP zum Einsatz. TCP alleine kann die Eigenschaft, dass das Endgerät über mehrere Netzwerkschnittstellen verfügt nicht ausnutzen. Eine TCP Verbindung kann nicht über mehrere Netzwerkschnittstellen gleichzeitig kommunizieren.
Aufbauend auf dieser Situation soll eine Lösung gefunden werden, wie durch Multipath TCP die Dienstgüte für das Endgerät verbessert werden kann. Der Lösungsweg soll möglichst wenig Änderungen an den vorhandenen Systemen erforderlich machen.

5 Multipath TCP

Multipath TCP erweitert TCP um die Fähigkeit eine logische Verbindung über mehrere Netzwerkschnittstellen des Senderes und des Empfängers zu betreiben. Eine Multipath TCP Verbindung besteht aus mehreren TCP Verbindungen, welche Subflows genannt werden. Ein Subflow ist somit eine TCP Verbindung zwischen je einer festen Netzwerkschnittstellen des Senders und des Empfängers. Um Multipath TCP zu nutzen, müssen sowohl Sender und Empfänger die MPTCP Erweiterung unterstützen. Ein Ziel von MPTCP ist möglichst kompatibel mit Middleboxes, also anderen Geräten zwischen Sen-

der und Empfänger zu sein. Multipath TCP ist vollständig mit TCP kompatibel. Sollte eine Partei die Protokollerweiterung nicht unterstützen findet die Kommunikation über TCP statt [13].
Eine Multipath TCP Verbindung wird durch einen 3-Wege Handshake aufgebaut. Vereinfacht findet zwischen MPTCP fähigen Geräten folgender Handshake statt: Beim TCP Verbindungsaufbau wird *MP_CAPABLE* als Option mitgeschickt. Der Empfänger sendet darauf ebenfalls die *MP_CAPABLE* Option zurück. Der Initiator des Verbindungsaufbau bestätigt die Antwort. Während des Handshakes findet auch ein Schlüsselaustausch statt. Diese Schlüssel dienen dazu, um beim Aufbau von weiteren Subflows diese zu authentifizieren [2].
Mit der Option *ADD_ADDR* werden IP Adressen der verfügbaren Netzwerkschnittstellen ausgetauscht. Dadurch können beide Kommunikationspartner weitere Subflows initieren. Dabei werden TCP Verbindungen aufgebaut, welche als Option *MP_JOIN* mitsenden. Durch die zuvor ausgetauschten Schlüssel, können Subflows zu einer MPTCP Verbindung zugeordnet werden. Subflows können jederzeit in einer MPTCP Verbindung auf und abgebaut werden.
Daten werden von Sender zu Empfänger über die Subflows gesendet. Wenn die Kommunikation über ein Subflow nicht funktioniert, können die Daten über einen anderen Subflow versendet werden. Da ein Subflow eine normale TCP Verbindung ist, kommt innerhalb eines Subflows das reguläre TCP Acknowledge System zu Einsatz. Um zu garantieren, dass alle über die Subflows verteilte Daten sequentiell beim Empfänger ankommen, kommt ein Subflowübergreifendes Acknowledge System zum Einsatz [2].
Es existieren Mechanismen um Prioritäten über die Subflows festzulegen [2]. Dadurch können Subflows als reine Backuplösung konfiguriert werden. Über diese werden nur Daten gesendet, wenn keine anderen Subflows verfügbar sind.
Um eine MPTCP Verbindung zu schließen, kommen auf Subflow Ebene neben regulären TCP *FIN* Flags auch ein MPTCP verbindungsübergreifendes *FIN* Flag zum Einsatz [2]. Daher werden in MPTCP werden zwischen zwei Arten von *FIN* Flags unterschieden. Da Subflows TCP Verbindungen sind, bewirken herkömmliche *FIN* Flags nur das Beenden eines Subflows. Eine komplette MPTCP Verbindung kann über den *DATA_FIN* Mechanismus geschlossen werden [2].
Auf `https://multipath-tcp.org/` [14] ist nachlesbar, wie ein mit Multipath TCP gepachter Linux Kernel installiert werden kann. Es existieren für viele gängige Linux Distributionen fertige Pakete, welche über Paketmanager installiert werden können. Wenn mehre Netzwerkinterfaces vorhanden sind, muss noch das Routing konfiguriert werden. Auf `https://multipath-tcp.`

`org/` sind Pythonskripte bereitgestellt, welche die Routingkonfiguration automatisieren [14].

5.1 Konfiguration unter Linux

Die Linux Kernel Implementierung von Multipath TCP kann auf verschiedene Weise konfiguriert werden. Einen großen Einfluss auf das Verhalten von Multipath TCP macht die Wahl des Algorithmus für das Path Managment, Scheduling und für die Congestion Control. Die Linux Kernel Implementierung wird mit jeweils verschiedenen Algorithmen ausgeliefert. Konfigurationen werden über das Verändern von Kernel Parameter über *sysctl* vorgenommen. Alternativ kann Multipath TCP über Socket Options konfiguriert werden [14].

Path Manager: Der Path Manager ist für das Erzeugen neuer Subflows zuständig. In der aktuellen Linux Kernel Implementierung (Version 0.92) stehen vier verschiedene Implementierungen bereit [14].

- fullmesh: Dies ist der standartmäßig verwendete Path Manager. Es werden alle mögliche Subflows über alle Netzwerkinterfaces aufgebaut [14].
- default: Der Host initialisiert kein Verbindungsaufbau von neuen Subflows und tauscht keine IP Adressen aus [14].
- ndiffports: Dieser Path Manager baut verschiedene Pfade über verschiedene Ports anstelle über verschiedene Netzwerkschnittstellen mit unterschiedlichen IP Adressen auf. Dadurch sollen von Middleboxen verursachte Limitierungen des Durchsatzes umgangen werden [15].
- binder: Erweitert MPTCP mit *Loose Source and Record Routing (LSRR)*. Dieser Path Manager wird für ein auf MPTCP und einem Proxy basierendem System benötigt, welches zur Optimierung von Privaten Netzwerken mit mehreren Schnittstellen zum Internet gedacht ist. Das Ziel ist, dass möglichst alle Schnittstellen zum Internet gleichmäßig ausgenutzt werden [16].

Über den sysctl Befehl können Path Manager ausgewählt werden:

```
$ sysctl -w net.mptcp.mptcp_path_manager=default
```

Scheduler: Der Scheduler entscheidet über welchen verfügbaren Subflow Datenpakete gesendet werden [17]. Der Standardscheduler der Linux Kernel Implementierung wählt den Subflow mit der kleinsten Round Trip Time aus. Pakete werden so lange über den ausgewählten Subflow gesendet, bis die durch das Congestion Window bestimmte maximale Datenmenge erreicht ist [14]. Es steht auch ein Round Robin Scheduler zur Verfügung. Allerdings hat sich gezeigt, dass der Standardscheduler im allgemeinen leistungsfähiger ist [8]. Der dritte verfügbare Scheduler sendet Datenpakete über alle verfügbare Subflows. Diese Art des Schedulings optimiert auf eine möglichst geringe Latenz [14].
Wie beim Path Manager lässt sich über den sysctl Befehl ein Scheduler auswählen:

```
$ sysctl -w net.mptcp.mptcp_scheduler=default
```

Congestion Control: Multipath TCP stellt besondere Anforderungen an den Congestion Control Mechanismus. Die Linux Implementierung von Multipath TCP wird mit Congestion Control Algorithmen mit verschiedenen Stärken ausgeliefert [18]. Generell müssen Congestion Control Algorithmen für Multipath TCP vier Ziele erreichen: Es soll ein möglichst hoher Durchsatz erreicht werden. Der Algorithmus muss den Ausfall eines Subflows bewältigen können. Normale TCP Ströme dürfen nicht beeinträchtigt und unfair behandelt werden. Auch soll der Congestion Control Algorithmus eine Verbindung mit einer geringen Latenz ermöglichen. Jeder Congestion Control Algorithmus stellt einen Kompromiss dar.

- Linked Increase Algorithm (lia): Die Größe des Congestion Window soll möglichst fair auf alle Subflows aufgeteilt werden. Der Durchsatz ist mindestens so gut wie bei regulärem TCP. Allerdings können normale TCP Verbindungen benachteiligt werden.

- Opportunistic Linked Increase Algorithm (olia): Der Nachteil von olia ist, dass dieser Algorithmus nicht gut darin ist Änderungen des Netzwerkes, wie den Wegfall eines Subflows, zu verarbeiten.

- Balanced Linked Adaptation Congestion Control Algorithm (balia): Kompromiss zwischen schneller Anpassungsfähigkeit an Netzwerkänderungen und dass normale TCP Verbindungen nicht benachteiligt werden.

- Delay-based Congestion Control for MPTCP (wVegas): Die Algorithmen lia, olia und balia betrachten den Verlust von Paketen, um das

Congestion Window zu bestimmen. In wVegas wird das Congestion Window über die Round Trip Time berechnet. Dies führt zu einem geringeren Verlust an Paketen.

Unter Linux wird standardmäßig unter Multipath TCP auch TCP Cubic verwendet. Die speziellen Congestion Control Algorithmen werden als Kernelmodule ausgeliefert. Um einen Congestion Control Algorithmus einzusetzen, muss als erstes das entsprechende Kernelmodul geladen werden:

```
$ modprobe mptcp_balia
```

Danach kann der entsprechende Algorithmus verwendet werden:

```
$ sysctl -w net.ipv4.tcp_congestion_control=balia
```

Es hat sich gezeigt, dass in der verwendeten Version der Multipath TCP Implementierung für den Linux Kernel der Linked Increase Algorithm (lia) entgegen der Dokumentation nicht mit ausgeliefert wird. Somit wird dieser Algorithmus im weiteren Verlauf der Arbeit nicht weiter betrachtet.

5.2 Konfiguration über Socket Options

Alternativ zu Kernel Parameter kann MPTCP unter Linux über Socket Options konfiguriert werden. Im Gegensatz zum systemweitem konfigurieren über Kernel Parameter kann hier MPTCP pro Socket konfiguriert werden. Es ist dadurch möglich, dass verschiedene Sockets unterschiedliche Path Manager, Scheduler oder Congestion Control Algorithmen nutzen. Um Socket Options zu setzen können, steht die Funktion *setsockopt*, welche im Header *sys/socket.h* zu finden ist, bereit. Im folgenden Beispiel wird Multipath TCP für einen Socket deaktiviert.

```
int enabled = 0;
int error = setsockopt(socket, SOL_TCP, MPTCP_ENABLED, &
    mptcp_enabled, sizeof(mptcp_enabled));
```

Als dritter Parameter wird eine Konstante erwartet, welche die zu konfigurierende Option identifiziert. Die verfügbaren Optionen sind unter *https://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP* zu finden:

```
#define MPTCP_ENABLED 42
#define MPTCP_SCHEDULER 43
#define MPTCP_PATH_MANAGER 44
```

6 Integration von Multipath TCP

Es werden drei verschiedene Integrationsmöglichkeiten vorgestellt. Jede dieser Möglichkeit ist für ein anderes Szenario geeignet.

6.1 MPTCP zwischen Endgerät und einem Reverse Proxy

In diesem Aufbau können gezielt einzelne Server von MPTCP profitieren. Die Server müssen unter Kontrolle des Proxy Betreibers stehen. Somit eignet sich die Lösung um Webserver im eigenen privatem Netzwerk mit Multipath TCP nachzurüsten, ohne Modifikationen an den Servern vorzunehmen. Ideal ist, wenn sich der Proxy und der Webserver im gleichen Rechenzentrum befinden. Die Netzwerkanbindung innerhalb eines Rechnenzentrums stellt in der Regel nicht das Bottleneck dar.
Es wird vor dem Webserver ein Reverse Proxy geschaltet. Dieser Reverse Proxy unterstützt das MPTCP Protokoll. Ein MPTCP fähiges Endgerät kommuniziert über MPTCP mit dem Reverse Proxy, welcher daraufhin eine Verbindung mit dem eigentlichen Zielserver aufbaut. Es muss der zum Webserver dazugehörige DNS Eintrag so angepasst werden, dass eine Namensauflösung die IP Adresse des Reverse Proxies anstelle des Webserver liefert. Durch IP-based oder Name-based virtual hosting kann ein Reverse Proxy für mehrere Server Multipath TCP ermöglichen. Vorteilhaft ist, dass dieses System transparent gegenüber dem Endgerät agiert. Neben der Voraussetzung, dass das Endgerät das MPTCP Protokoll unterstützt, muss nichts weiteres am Endgerät konfiguriert werden. Darüber hinaus lässt sich bei diesem Deployment einfach Load Balancing zwischen dem Proxy und mehreren Servern, welche den gleichen Dienst anbieten, realisieren.

6.2 MPTCP zwischen Endgerät und einem socks Proxy Server

Diese Option orientiert sich an dem Giga Path Service der Korea Telecom. Hier kommt ein socks5 Proxy zum Einsatz.
Im Gegensatz zu der Reverse Proxy Lösung muss der Client das socks5 Protokoll unterstützen und die Adresse des Proxy Servers bekannt sein. Die Lösung ist also nicht transparent gegenüber dem Client. Dafür funktioniert diese Methode mit jedem beliebigen Webserver. Während der Reverse Proxy die unterstützten Server kennen musste, sendet hier der Client die Zieladresse und Port oder eine URL an den Proxy, welcher dann stellvertretend eine Verbindung zum Ziel aufbaut und die Anfrage ausführt. Zuvor muss der Client jedoch eine Verbindung zum Proxy aufbauen und sich authentifizieren. Bei socks5 kann die Kommunikation zwischen Client und Proxy über TCP oder UDP ablaufen [19].
Wenn sowohl der socks5 Proxy Server als auch der Client Multipath TCP fähig ist, kann die Kommunikation zwischen beiden über mehrere MPTCP Subflows stattfinden. Dies erlaubt zum Beispiel, dass Endgeräte über alle ver-

fügbaren Netzwerkschnittstellen kommunizieren können. Die Verbindungen zwischen dem Proxy und dem Zielserver findet über reguläres TCP statt.

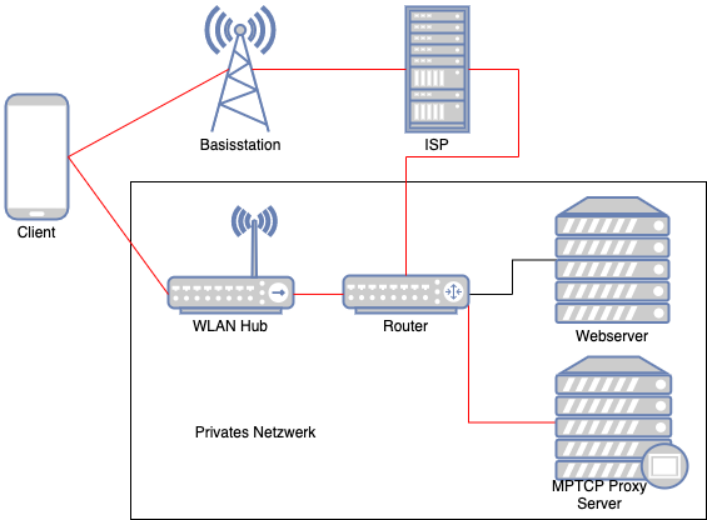


Abbildung 2: MPTCP zwischen Endgerät und Proxy. Der Proxy kann als Reverse Proxy oder als socks5 Proxy realisiert werden. Rot markiert sind die Links, über welche mit MPTCP kommuniziert wird. In diesem Beispiel ist das WLAN unter Kontrolle des Betreibers des privaten Netzwerks. Es ist auch denkbar, dass anstelle des WLAN ein zweites Mobilfunknetzwerk oder ein öffentliches WLAN eingesetzt wird.

6.3 MPTCP zwischen einem Router und einem socks Proxy Server

Diese Variante ist vergleichbar mit der Lösung der Swisscom oder Tessares. Multipath TCP kommt nicht zwischen dem Client und einem Proxy sondern zwischen einem speziellen Router und Proxy zum Einsatz. Die Clients und der Router kommunizieren über reguläres TCP. Durch diese Lösung können alle Geräte im Netzwerk von mehreren Anbindungen an mehrere Internet Service Provider profitieren. Da MPTCP nur zwischen Router und Proxy Server genutzt wird, profitieren auch Endgeräte ohne MPTCP Support von diesem Integrationsmodel. Diese Lösung ist deswegen auch transparent gegenüber den Endgeräten.
Einfach umsetzbar ist dieses Konzept mit dem Open Source Projekt *OpenMPTCPRouter*. Das Projekt stellt sowohl ein auf OpenWRT basiertes Softwarepaket für den Router als auch die Software für den Proxy Server bereit. Der gesamte Internetverkehr wird vom OpenMPTCPRouter über einen socks5

Proxy getunnelt. Der Proxy Server baut eine reguläre TCP Verbindung zu der Zieladdresse auf. Neben Multipath TCP wird auch eine Multipath Lösung für UDP mit ausgeliefert. OpenMPTCPRouter steht unter der GPL-3.0 Lizenz [20].

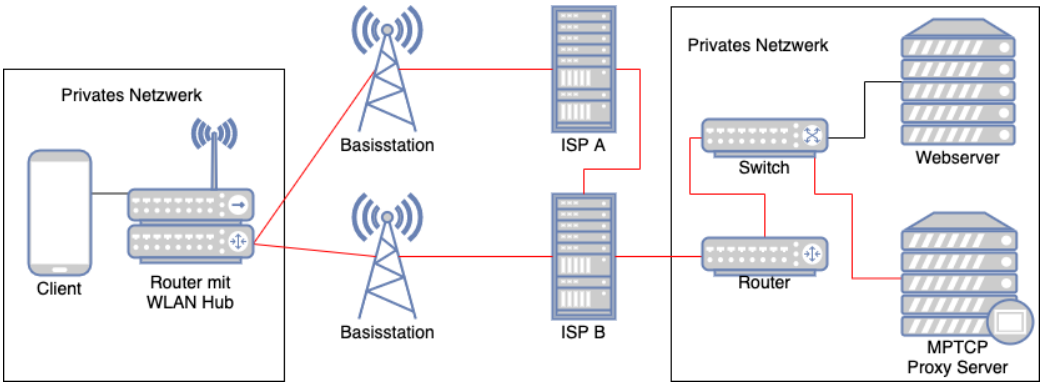


Abbildung 3: Veranschaulichung der Architektur mit MPTCP zwischen einem Router und socks5 Proxy Server. In diesem Szenario wird der Durchsatz von zwei mobilen Netzwerken kombiniert. Der Client muss MPTCP nicht unterstützen.

7 Mininet

Um Netzwerke zu testen kann man diese in kleinem Umfang, den Testbeds, nachbilden. Alternativ zu den aus Hardware bestehenden Testbeds, kann man auch auf virtuelle Lösungen zurückgreifen. Virtuelle Testbeds haben den Vorteil, dass die Beschaffung von Hardware entfällt und Netzwerke flexibel und schnell aufgebaut werden können. Mininet ist eine bewährte Lösung, welche das Simulieren eines Netzwerks auf einem lokalen Rechner ermöglicht. Mithilfe von Python Skripte können verschiedene Netzwerktopologien abgebildet werden. Hauptsächlich arbeitet Mininet mit dem Linux Kernel Feature, Network Namespaces. Network Namespaces ermöglichen, dass für Prozesse unabhängig voneinander eigene Netzwerkkonfigurationen erstellt werden können. Somit können für verschiedene Prozesse unter anderem unterschiedliche Netzwerkinterfaces, Routingtabellen, ARP-Tabellen oder Firewalls konfiguriert werden. Im Gegensatz zu Containern findet keine Isolation des Dateisystems statt. Alle Mininet Prozesse können also auf die gleichen Dateien im Dateisystem zugreifen [21].
Um Komponenten im Netzwerk zu konfigurieren, kommt das OpenFlow Protokoll zum Einsatz. OpenFlow ermöglicht Software-Defined Networking. Die einzelnen Netzwerkkomponenten werden von einem Controller gesteuert. Der

Controller kommuniziert mit den Komponenten und gibt das Verhalten vor. Zum Beispiel teilt der Controller einem Router mit, nach welchen Regeln Pakete weitergeleitet werden sollen.
Mithilfe von Wireshark können die Pakete an jeder Netzwerkkomponente analysiert werden. Durch XTerm kann auf jeder Netzwerkkomponente eine Shell geöffnet werden [21].

8 Konstruktion des Testbed

Durch beide Möglichkeiten Multipath TCP zu konfigurieren (Kernel Parameter und Socket Options) ergeben sich auch zwei Möglichkeiten, um ein Testbed aufzubauen.

8.1 Verschieden konfigurierte Sockets

Über Socket Options werden Sockets unterschiedlich konfiguriert. So lassen sich für mache Sockets Multipath TCP deaktivieren während andere Sockets Multipath TCP unterstützen. Das Problem an dieser Vorgehensweise ist, dass direkt mit Sockets interagiert werden muss. Bei Drittsoftware erfordert das die Modifizierung des Quellcodes. Dies ist mit hohem Aufwand verbunden oder bei proprietärer Software unmöglich.
Für die Arbeit wurde iperf3 [22] modifiziert. Die Modifikation fügt neue Optionen hinzu:

```
—mptcp_enabled 1
—mptcp_cc olia
—mptcp_pm fullmesh
—mptcp_scheduler default
```

Es lässt sich bei dem von iperf3 genutzten Socket einstellen, ob MPTCP verwendet wird, welcher Congestion Control Algorithmus, Path Manager und Scheduler Algorithmus verwendet werden. Der Quellcode und eine Kurzbeschreibung des modifizierten iperf3 Fork ist in einem Repository in Github [1] zu finden.
Der Testaufbau besteht aus einer virtuellen Maschine in welcher Ubuntu Server 18.04.4 LTS installiert ist. Der Multipath TCP Kernel Patch ist darauf installiert. Standardmäßig ist MPTCP aktiviert. Mit Mininet werden drei Hosts, ein Switch und ein Router simuliert. Die Hosts werden als Client, Proxyserver und Webserver verwendet. Der Webserver soll Multipath TCP, im Gegensatz zu dem Client und Proxy, nicht unterstützen. Der Server und der Client werden über iperf3 simuliert. Beim Server wird MPTCP über die

[1] https://github.com/SubmergedTree/MPTCP-iperf3

neu hinzugekommene Option *mptcp_ enabled* deaktiviert. Am Client können verschiedene MPTCP Konfigurationen vorgenommen werden. MPTCP Verkehr findet zwischen dem Client und dem Proxy statt. Da kein Zugriff auf den Socket des Proxies besteht, können nur Konfigurationen über Kernel Parameter vorgenommen werden, welche am Client und Server durch die neuen iperf3 Optionen auf Socket Ebene überschrieben werden.

Das simulierte Netzwerk wird in 3 Subnetze unterteilt:

- Mobilfunk: Dieses Netzwerk besteht aus einer Verbindung zwischen dem Client und dem Router. Darüber wird möglichst realistisch eine Verbindung über Mobilfunk simuliert. Mininet ermöglicht das simulieren von Eigenschaften der Verbindungen zwischen den Komponenten. Für den Mobilfunk wird der Durchsatz auf 10 Mbit/s beschränkt und eine Latenz von 100 Millisekunden simuliert. Außerdem wird ein Paketverlust von 1% eingestellt.

- WLAN: Wie beim Mobilfunknetzwerk werden hier die Eigenschaften einer WLAN Verbindung simuliert. Der Durchsatz wird auf 20 Mbit/s beschränkt und eine Latenz von 20 Millisekunden wird konfiguriert.

- Das dritte Netzwerk ist das Netzwerk des Server/Infrastrukturbetreibers und enthält den Proxy und den Webserver. Hier erlauben alle Verbindungen einen maximalen Durchsatz von 50 Mbit/s.

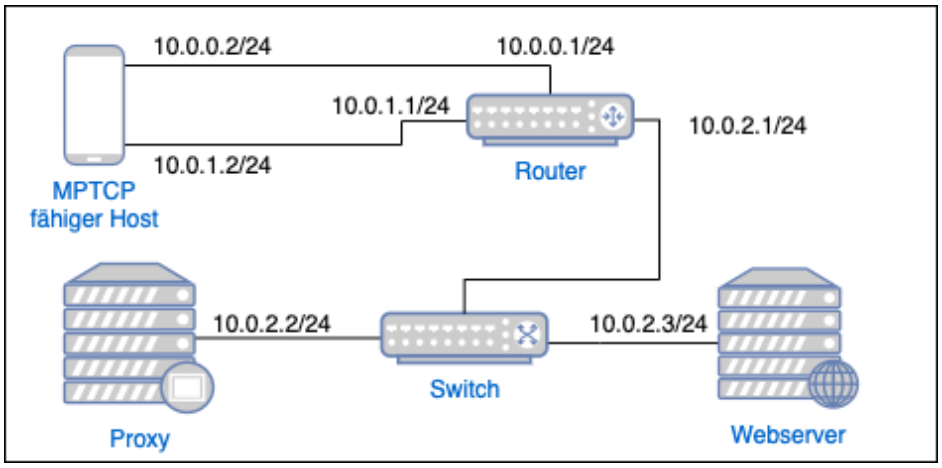


Abbildung 4: Testbed: Konfiguration von MPTCP über Socket Options.

8.2 Mehrere Virtuelle Maschinen - Betriebssystemweite Konfiguration von Multipath TCP

Nicht immer kann direkt mit Sockets gearbeitet werden. Bei Drittsoftware besteht oftmals kein Zugriff auf den Quellcode oder eine Modifizierung wäre zu aufwändig. Trotzdem sollen Komponenten mit MPTCP sowie ohne MPTCP Unterstützung simuliert werden können. Dies ist in Mininet ohne weiteres nicht möglich. Das liegt daran, dass sich alle von Mininet simulierten Geräten den Linux Kernel mit dem Gastsystem teilen. Multipath MPTCP ist im Linux Kernel implementiert. Es ist folglich nicht möglich, das MPTCP Feature exklusiv für bestimmte Mininet Hosts zu deaktivieren.
Um diese Limitierung zu überwinden, müssen zwei Mininet Instanzen auf zwei verschiedene Gastsystemen eingesetzt werden. Ein Gastsystem benutzt ein in der aktuellen MPTCP Version gepachteten Linux Kernel das Andere nicht. Beide Instanzen müssen transparent miteinander verbunden werden. Das heißt, für die simulierten Geräte darf es bis auf die Unterstützung von MPTCP kein Unterschied machen, in welcher Mininet Instanz der Kommunikationspartner bereitgestellt ist. Als Gastsystem kommt jeweils Ubuntu Server 18.04.4 LTS zum Einsatz. Beide werden in Virtual Box als virtuelle Maschine betrieben. Auf beiden Gastsystemen ist Mininet installiert.
Der für die korrekte Funktion von SDNs benötigte Controller kann von beiden Mininet Netzwerken geteilt werden. Der Controller wird auf einem Gastsystem gestartet. Durch das Remote Controller Feature kann die Mininet Instanz in dem anderen Gastsystem auf den Controller zugreifen.
Jetzt müssen noch beide Mininet Netzwerke transparent miteinander verbunden werden. Dafür eignen sich das Generic Route Encapsulation (GRE) Protokoll [23]. Mit dem GRE Protokoll lassen sich Point-to-Point Tunnel zwischen Switches umsetzen. Dabei wird das zu versendende Paket in ein GRE Packet gekapselt und über IPv4 versendet. Ein zu beachtender Fallstrick ist, dass die Maximum Transmission Unit (MTU) bei den Netzwerkschnittstellen beider virtuellen Maschinen erhöht werden muss. Wird dies nicht beachtet kann Paketverlust auftreten. In diesem Experiment wurde die MTU jeweils auf 5000 gesetzt:

```
$ sudo ifconfig <network interface> mtu 5000
```

Wie bei der Simulation mit Socket Options bestehen drei Subnetze je für Mobilfunk, WLAN und dem Netzwerk indem sich der Webserver und Proxy befindet. Auch Durchsatz, Latenz und Paketverlust werden identisch gewählt.

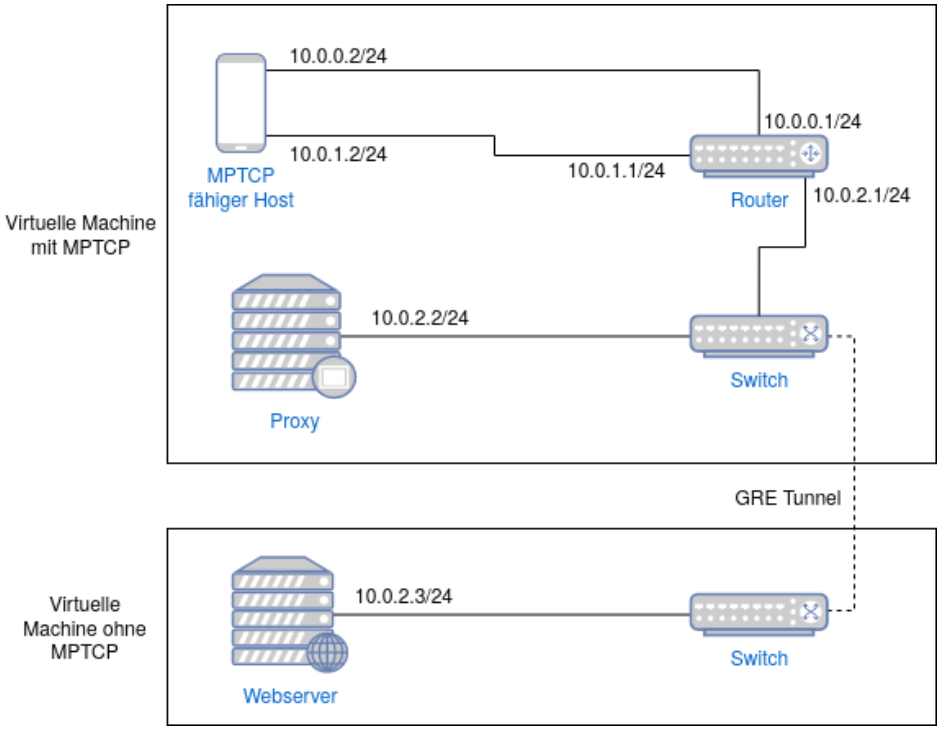


Abbildung 5: Testbed mit mehreren Virtuellen Maschinen

Mit Wireshark lässt sich der Netzwerkverkehr beobachten:

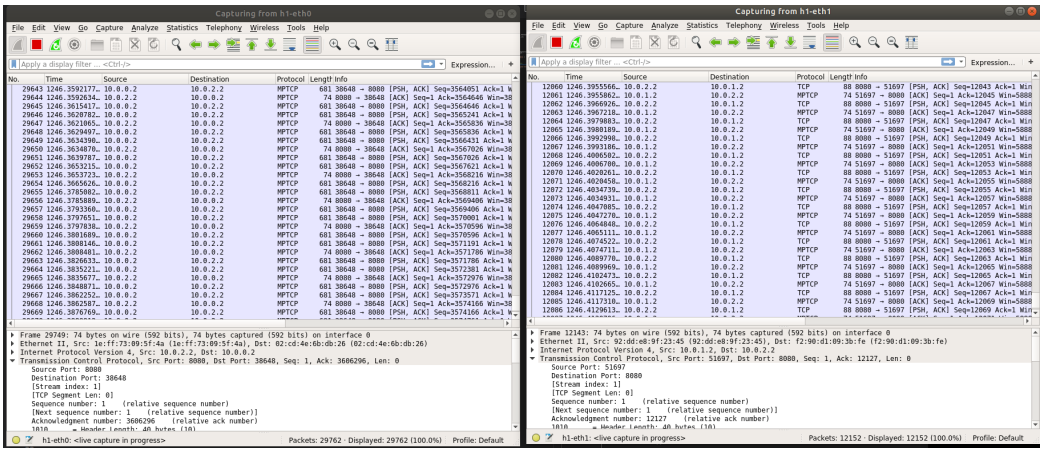


Abbildung 6: Der Netzwerkverkehr auf beiden Netzwerkschnittstellen des MPTCP fähigen Clients.

9 Evaluation

Ziel ist es beide Testbeds zu evaluieren. Außerdem soll betrachtet werden, wie stark sich Multipath TCP auf die Performance auswirkt.
In der Evaluation wird als Proxylösung einen über NGINX realisierten Reverse Proxy eingesetzt. Der Overhead, welcher ein Socks Proxy mit sich bringt, hat kein Einfluss auf das Ergebnis. Das Testbed ist allerdings flexibel genug, dass der Reverse Proxy auch gegen ein Socks Proxy ausgetauscht werden kann.
Zur Simulation von Last wird die vorgestellte modifizierte Version von *IPerf3* verwendet. Dadurch kann am Client direkt die verwendeten Congestion Control und Scheduling Algorithmen gewählt und MPTCP aktiviert und deaktiviert werden. Der Reverse Proxy agiert als direkter MPTCP Kommunikationspartner. Es besteht beim Proxy kein direkter Zugriff auf den Socket. Deshalb können Socket Options nicht gesetzt werden. Hier muss deshalb mit Kernel Parametern gearbeitet werden. Serverseitig kann entweder Multipath TCP über Socket Options deaktiviert werden oder der Server in einer über Tunnel verbundenen zweiten virtuellen Maschine ohne MPTCP Unterstützung bereitgestellt werden.
Um aussagekräftige Ergebnisse zu erlangen, müssen die Eigenschaften der Links zwischen den einzelnen Komponenten realistisch gewählt werden. Mininet erlaubt das Setzen der Bandbreite, Latenz und Packetverlust.
Die Komponente, welche den Client simuliert, ist mit zwei Links mit einem Router verbunden. Jeweils ein Link simuliert eine Verbindung über WLAN und eine Verbindung über Mobilfunk. Alle anderen Links sind einheitlich konfiguriert:

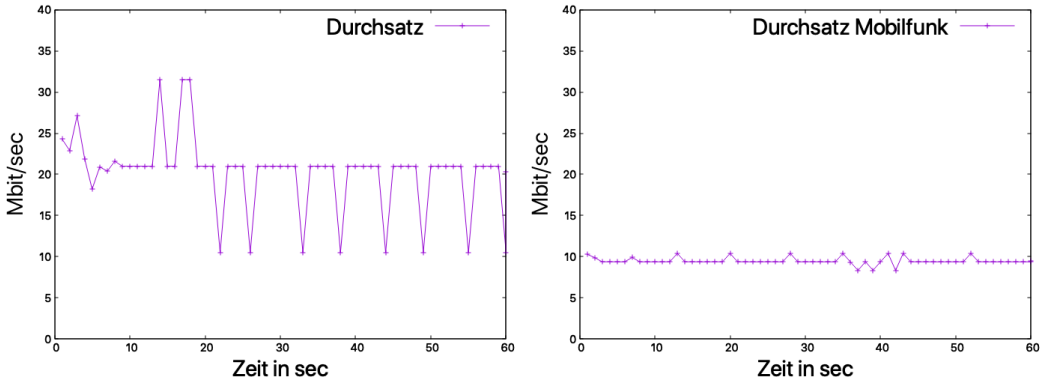
Link	Bandbreite	Latenz	Packetverlust
Client zu Router: WLAN	20 Mbit/s	20 ms	0 %
Client zu Router: Mobilfunk	10 Mbit/s	100 ms	1 %
Sonstige Links	50 Mbit/s	0 ms	0 %

Abbildung 7: Konfigurationen der Verbindungen zwischen den Komponenten.

Alle Tests werden auf identischer Infrastruktur ausgeführt. Zur Virtualisierung wird VirtualBox verwendet. Als Gastsystem kommt MacOS zum Einsatz. Der Prozessor ist ein Intel i5 - ein Dual-Core Prozessor aus dem Jahr 2015.

9.1 Ohne MPTCP

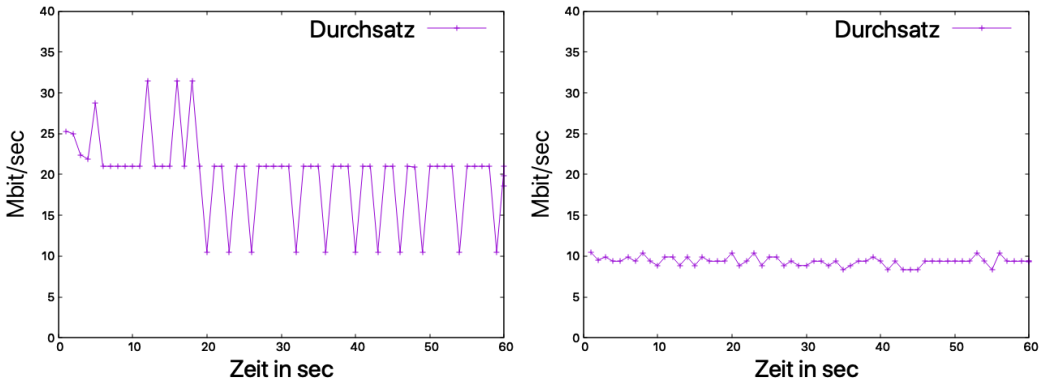
Zu Begin soll zur Referenz die Leistung beider Netzwerkschnittstellen unabhängig voneinander gemessen werden. Dafür wird am Client Multipath TCP deaktiviert und iperf3 so konfiguriert, dass in den Testdurchgängen die unterschiedlichen Netzwerkschnittstellen verwendet werden. Die Werte werden am Client gemessen. Getestet wird zuerst mit dem Socket Options Testbed:



(a) Nur WLAN (b) Nur Mobilfunk

Abbildung 8: Einzelne Betrachtung der Netzwerkschnittstellen unter (normalem) TCP.

Bei WLAN wird durchschnittlich ein Durchsatz von 20.3 Mbit/s Sekunde erzielt und beim Mobilfunk durchschnittlich 9.49 Mbit/s. Die gemessene Round Trip Time (RTT) beträgt im Durchschnitt bei WLAN 21.4 ms und beim Mobilfunk 63 ms.
Beim Testbed mit den 2 virtuellen Maschinen ergibt sich ein sehr ähnliches Bild:



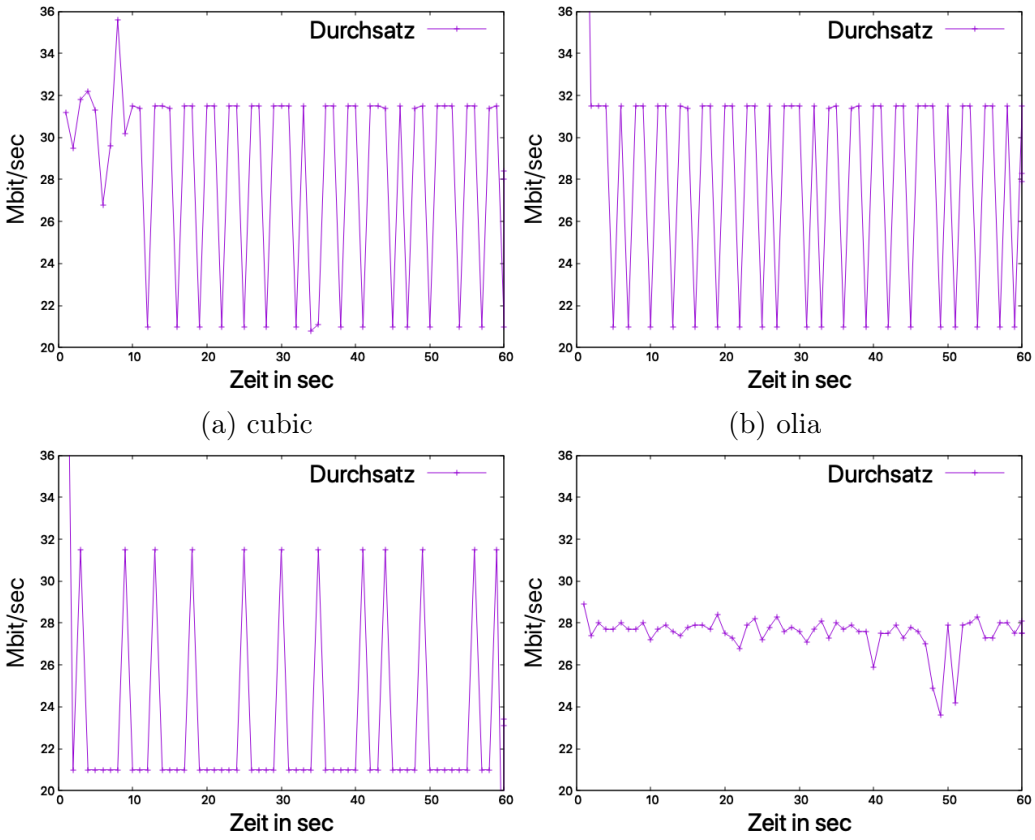
(a) Nur WLAN (b) Nur Mobilfunk

Abbildung 9: Einzelne Betrachtung der Netzwerkschnittstellen unter (normalem) TCP.

Bei WLAN wird ein Durchsatz von 19.9 Mbit/s und eine durchschnittliche Latenz von 20.8 ms erreicht. Beim Mobilfunk sind es 9.38 Mbit/s Durchsatz und 96.6 ms Latenz.

9.2 Vergleich der Testbeds

Es werden das Socket Options basierte und das Testbed mit zwei virtuellen Maschinen welche über einen GRE Tunnel verbunden sind, miteinander verglichen. Standardmäßig ist MPTCP beim iperf3 Client aktiviert. Serverseitig ist MPTCP deaktiviert. Die Performance beider Testbeds werden veglichen. Dabei kommen verschiedene Congestion Control Algorithmen zum Einsatz. Hier wird der Durchsatz in Megabits pro Sekunde und die Latenz in Millisekunden gemessen. Neben dem über Iperf3 generiertem Verkehr liegt keine Last an dem System an.
Als erstes wird das Testbed mit mehreren virtuellen Maschinen betrachtet. Der Verkehr verläuft durch einen GRE Tunnel.



(a) cubic (b) olia

(c) balia (d) wVegas

Abbildung 10: Einfluss verschiedener Congestion Control Algorithmen auf den Durchsatz

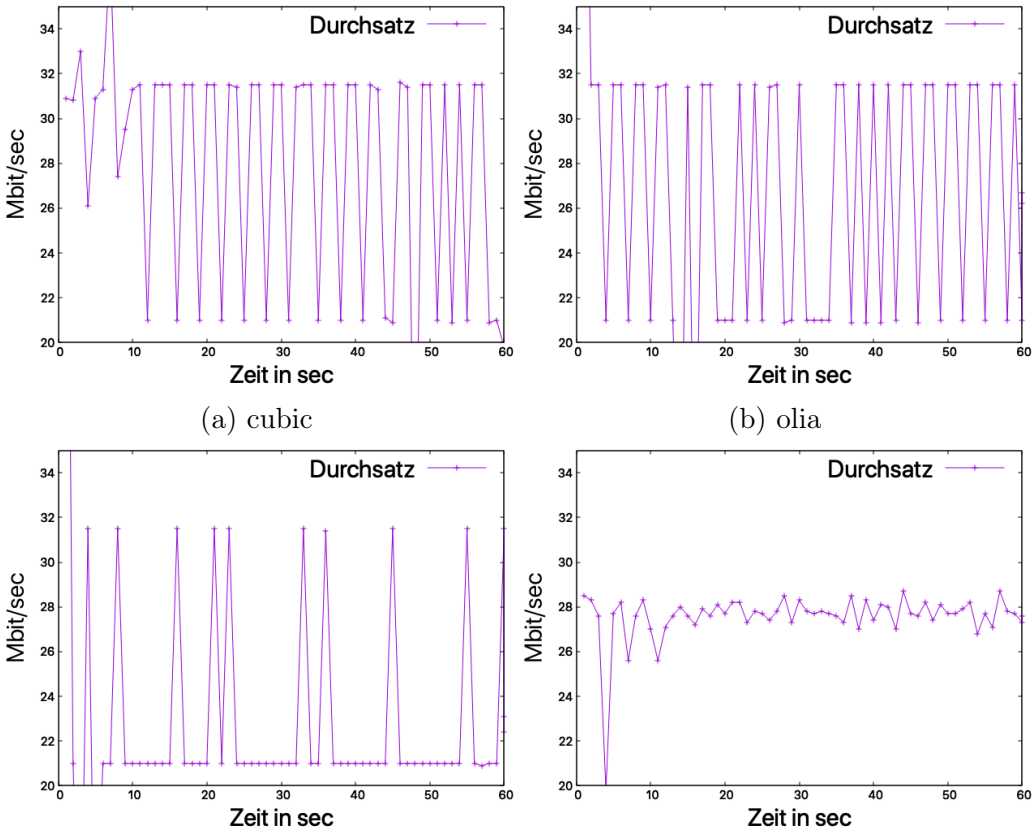
Congestion Control Algorithmus	Durchschnittlicher Durchsatz
cubic	28.4 Mbit/s
olia	28.3 Mbit/s
balia	23.4 Mbit/s
wvegas	27.5 Mbit/s

Abbildung 11: Durchschnittlicher Durchsatz bei dem Testbed mit zwei virtuellen Maschinen

Algorithmus	Durchschnitt	Median	Standardabweichung
cubic	24 ms	22 ms	7.8 ms
olia	20 ms	17 ms	7.8 ms
balia	21 ms	21 ms	9.1 ms
wvegas	21 ms	19 ms	7.7 ms

Abbildung 12: Durchschnittliche Latenz bei dem Testbed mit zwei virtuellen Maschinen

Jetzt wird das Testbed, welches mit Sockets Options arbeitet, betrachtet:



(a) cubic

(b) olia

(c) balia

(d) wVegas

Abbildung 13: Einfluss verschiedener Congestion Control Algorithmen auf den Durchsatz

Congestion Control Algorithmus	Durchschnittlicher Durchsatz
cubic	27.8 Mbit/s
olia	26.7 Mbit/s
balia	23.1 Mbit/s
wvegas	27.6 Mbit/s

Abbildung 14: Durchschnittlicher Durchsatz bei dem Socket Options Testbed

Algorithmus	Durchschnitt	Median	Standardabweichung
cubic	25 ms	27 ms	7.1 ms
olia	21 ms	17 ms	8.1 ms
balia	19 ms	16 ms	6.5 ms
wvegas	22 ms	22 ms	8.1 ms

Abbildung 15: Durchschnittliche Latenz bei dem Socket Options Testbed

Leistungsmäßig sind beide Testbeds vergleichbar. Bei der Visualisierung des Durchsatzes ergeben sich für jeden Congestion Control Algorithmus ein charakteristisches Bild. Der Durchsatz ist bei cubic, olia und wVegas auf vergleichbarem Niveau. balia ermöglicht eine gute Anpassungsfähigkeit an Netzwerkänderungen und benachteiligt normale TCP Verbindungen nicht. Dieser Kompromiss kann aber die Ursache für den geringeren Durchsatz sein. Bei wVegas ist zu erkennen, dass der Durchsatz nie so stark abfällt, wie es bei den anderen Congestion Control Algorithmen der Fall ist. Das liegt daran, dass die Sendeleistung verringert wird, wenn die Round Trip Time steigt und nicht erst, wenn schon ein Paket verloren ist.
In den Messreihen sind bei der Latenz nur Abweichung von wenigen Millisekunden aufgetreten. Überraschend ist, dass teilweise eine bessere Latenz als 20 Millisekunden aufgetreten ist. Erwartet wurde, dass die Latenz größer als die Latenz beim alleinigen Einsatz von WLAN ist. Da der default Scheduler verwendet wurde, sollten die ersten Pakete über den Subflow mit der geringsten RTT gesendet werden, bis dass Congestion Windows voll ist. In dem Testaufbau entspricht das der WLAN Schnittstelle. In dieser Phase sollte die Latenz vergleichbar wie bei alleinigem Einsatz von WLAN unter normalem TCP sein. In der zweiten Phase, wenn das Congestion Window voll ist, sollte die Übertragung über den Subflow, welcher über den Mobilfunk verbunden ist, laufen. Dabei sollte die Latenz deutlich höher sein. Daher sollte die Latenz im Mittel deutlich höher sein, als bei dem alleinigen verwenden von WLAN unter normalem TCP. Dies ist aber nicht der Fall.

9.3 Der Redundant Scheduler

Neben dem default Scheduler wird mit der Linux Implementierung auch der Redundant Scheduler ausgeliefert. Dieser überträgt die Pakete über alle verfügbaren Subflows. Dadurch werden automatisch die Pakete über den Subflow mit der geringsten Latenz übertragen. Dabei kann es sein, dass der Durchsatz nicht verbessert wird oder sogar schlechter ist als wenn nur normales TCP zum Einsatz kommt. In diesem Versuch kommt als Congestion Control Algorithmus olia zum Einsatz. Es wird nur das Socket Options Testbed verwendet, da die vorangegangenen Test gezeigt haben, dass die Ergebnisse beider Testbeds vergleichbar sind [14].

Protokoll	Durchschnitt	Median	Std. Abw.
TCP (WLAN)	17 ms	16 ms	3.2 ms
MPTCP; default Scheduler	19 ms	16 ms	6.5 ms
MPTCP; redundant Scheduler	20 ms	17 ms	8.1 ms

Abbildung 16: Latenz bei verschiedenen Schedulern. Latenz bei TCP als Vergleichswert.

Zwischen dem redundant und default Scheduler ist in der vorliegende Konfiguration kein Unterschied Messbar. Auch bei stark erhöhter Latenz ergibt sich kein messbarer Vorteil vom redundant gegenüber dem default Scheduler. Es wird beim Mobilfunk mit einer Latenz von 800 ms anstelle von 100 ms getestet:

Protokoll	Durchschnitt	Median	Std. Abw.
default Scheduler	21 ms	20 ms	7.5 ms
redundant Scheduler	22 ms	21 ms	8.3 ms

Abbildung 17: Latenz bei verschiedenen Schedulern. Erhöhte Latenz von 800 ms bei der Mobilfunk Verbindung.

Zusammenfassend lässt sich zum redundant Scheduler sagen, dass sich auch bei sehr hoher Latenzdifferenz kein Vorteil gegenüber dem default Scheduler ergibt.

9.4 Verhalten bei Ausfall eines Netzwerk(-schnittstelle)

Durch Multipath TCP besteht eine logische TCP Verbindung aus mehreren physischen Subflows. In den Testbeds bestehen Verbindungen vom Client zum Proxyserver aus zwei Subflows. In einem realem Szenario kann es

vorkommen, dass eine Verbindung ausfällt. Ausgelöst werden kann dies beispielsweise durch Mobilität. Durch das Verlassen einer Mobilfunkzelle oder der Reichweite eines WLAN Access Points bricht die jeweilige Verbindung ab.
In diesem Versuch soll die Auswirkung eines Abbruchs einer Verbindung näher betrachtet werden. Dafür wird während einer laufenden Sitzung eine Netzwerkschnittstelle deaktiviert. Eine Netzwerkstelle kann über *ifconfig* deaktiviert und wieder aktiviert werden:

```
$ sudo ifconfig <network interface> down
$ sudo ifconfig <network interface> up
```

Eine Limitierung von Mininet ist, dass wenn mehrere Links zwei Geräte verbinden, zur Laufzeit nur alle Links aber nicht Einzelne entfernt werden können. Daher muss dies über das An- und Ausschalten von Netzwerkschnittstellen simuliert werden.
Es wird wieder das Socket Options Testbed verwendet. Beobachtet wird die Regenerationszeit. Dies ist die Zeit die benötigt wird, um den ursprünglichen Durchsatz nach erneutem Aktivieren einer Verbindung zu erreichen. Pro Congestion Control Algorithmus wird das Experiment 15 mal wiederholt.

Congestion Control Algorithmus	Durchschnitt	Median	Standardabweichung
cubic	16.2s	17s	8,29s
olia	15s	13s	7,16s
balia	15.4s	14s	6,58s
wVegas	13.93s	16s	7.53s

Abbildung 18: Kennzahlen Regenerationszeit

Es lassen sich nur geringe Unterschiede zwischen den einzelnen Congestion Control Algorithmen erkennen. Am Besten schneidet olia und balia ab. Am schlechtesten cubic. Dass balia gut auf die Situation reagieren kann, ist zu erwarten. Das olia ähnlich gut wie balia abschneidet ist überraschend, da das Reagieren auf eine Netzwerkänderung eine Schwäche von olia darstellen sollte.

10 Ergebnis

Zu Beginn der Arbeit wurden verschiedene Deploymentmöglichkeiten für Multipath TCP in ein vorhandenes Netzwerk erörtert. Danach wurden zwei virtuelle Testumgebungen vorgestellt um Multipath TCP evaluieren zu können. Die erste Testumgebung verwendet zwei verschiedenen virtuelle Ma-

schinen, welche durch einen Tunnel verbunden sind. Eine virtuelle Maschine unterstützt MPTCP, die Andere nicht. In der zweiten Testumgebung wird nur eine virtuelle Maschine benötigt. Diese unterstützt Multipath TCP. Die Konfiguration von MPTCP wird durch Socket Options direkt am Socket vorgenommen. Es hat sich ergeben, dass beide Testbeds geeignet sind um ein Multipath TCP Deployment zu evaluieren.
In dieser Arbeit wurde die Linux Implementierung von Multipath TCP betrachtet. Allgemein kommt es beim Einsatz von Multipath TCP zu einer steigerung des Durchsatzes. Man kann beinahe von einer Addition der Bandbreite der Verbindungen reden.
In den Versuchen haben alle Congestion Control Algorithmen gut abgeschnitten. Nur bei balia war die Bandbreite messbar vermindert. Allerdings wurden in dem Tests Faktoren, wie zusätzlicher Verkehr oder Einfluss auf und von anderen TCP Verbindungen vernachlässigt. Um eine genauere Aussage über die Congestion Control Algorithmen zu treffen, müssen in nachfolgenden Versuchen diese Faktoren miteinbezogen werden.
Der redundant Scheduler konnte in den Versuchen sich selbst unter extremen Bedingungen nicht von dem default Scheduler absetzten.
Das Wegfallen und Dazukommen von Verbindungen kann von Multipath TCP gut verarbeitet werden. Aber in den Versuchen hat sich auch hier kein Congestion Control Algorithmus von den anderen Absetzen können.
Insgesamt stellt sich Multipath TCP als interessante Technologie dar. Wenn mehrere Netzwerkschnittstellen verfügbar sind hat Multipath TCP einen deutlich positiven Einfluss auf die Bandbreite. Mit den Testbed sind stehen zwei Möglichkeiten bereit, um Multipath TCP weiter zu evaluieren.

11 Ausblick

11.1 MPTCP im Rechenzentrum/Cloud

In dieser Arbeit wurde ausschließlich auf die Vorteile für Endgeräte wie Smartphones eingegangen. MPTCP kann aber auch im Kontext von Rechenzentren Vorteile bieten [24]. Die vorgestellten Testbeds können auch beim Untersuchen von Multipath TCP im Rechenzentrum eingesetzt werden.

11.1.1 Proxy Erweiterung mit LSRR - Binder

Bei großen privaten Netzwerken mit mehreren Schnittstellen zum Internet wäre es wünschenswert, dass die Schnittstellen gleichmäßig ausgenutzt werden. Im Paper *Binder: A System to Aggregate Multiple Internet Gateways in*

Community Networks [16] wird eine Lösung dafür vorgestellt. Bei herkömmlichem Routing wählt jeder Router mithilfe von Longest Prefix Matching die Adresse des nächsten Hobs aus. Der Sender eines Pakets hat keinen Einfluss welchen Weg das Paket genau nimmt. Hier ist die Zielsetzung, dass die Subflows einer MPTCP Verbindung über alle Schnittstellen zum Internet verteilt werden. Um dies zu erreichen wird MPTCP mit Loose Source and Record Routing (LSRR) erweitert. Durch LSRR kann der Initiator der Verbindung den Pfad voraus festlegen. Der Initiator der MPTCP Verbindung legt dabei die Pfade der Subflows so fest, dass die Subflows über möglichst verschiedene Schnittstellen des privaten Netzwerk zum Internet verlaufen.
Die Linux Kernel Implementierung von Multipath TCP wird mit einer Implementierung dieses Verfahrens ausgeliefert. Um es zu benutzen, muss der Path Manager mit dem Namen *Binder* ausgewählt werden.
Problematisch an dieser Lösung ist, dass LSRR anfällig für Angriffe ist. Da die Routinginformation im IP Header stehen, kann ein Angreifer die Pfade modifizieren und die Pakete umleiten. Daher verwerfen manche Geräte Pakete, welche LSRR nutzen. Um dies zu verhindern, werden vor dem Übergang der Pakete in das Internet die LSRR Einträge im Header gelöscht [16] .

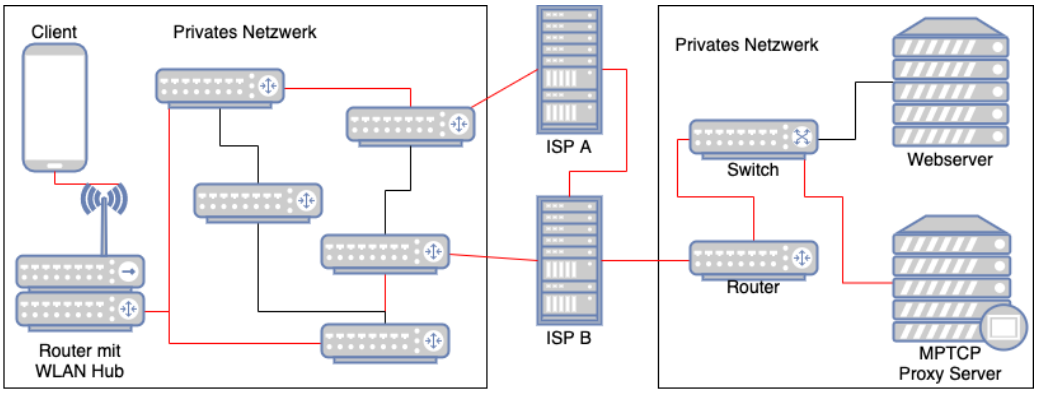


Abbildung 19: Durch die Erweiterung von MPTCP mit LSRR können die Subflows (rot) gezielt über bestimmte Pfade geroutet werden. In diesem Beispiel sorgt LSRR dafür, dass Subflows über beide ISPs geleitet werden.

11.2 Verhalten bei Mobilität und Handovers

Das Verhalten bei Handover wurde in dieser Arbeit nur rudimentär getestet, indem eine Schnittstelle deaktiviert worden ist. Um richtige Mobilität zu simulieren gibt es bessere Lösungen. Eine Lösung, speziell für WLAN ist Mininet-Wifi [25].

Literatur

[1] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: from theory to practice. In *International Conference on Research in Networking*, pages 444–457. Springer, 2011.

[2] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, et al. Tcp extensions for multipath operation with multiple addresses. Technical report, RFC 6824, 2013.

[3] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. Exploring mobile/wifi handover with multipath tcp. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 31–36, 2012.

[4] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):266–277, 2011.

[5] Olivier Bonaventure and S Seo. Multipath tcp deployments. *IETF Journal*, 12(2):24–27, 2016.

[6] Alexander Frömmgen. Mininet/netem emulation pitfalls: A multipath tcp scheduling experience. *Technical Report*, 2017.

[7] Thomas Dreibholz, Xing Zhou, and Fu Fa. Multi-path tcp in real-world setups–an evaluation in the nornet core testbed. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 617–622. IEEE, 2015.

[8] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pages 27–32, 2014.

[9] Fa Fu, Xing Zhou, Thomas Dreibholz, Keying Wang, Feng Zhou, and Quan Gan. Performance comparison of congestion control strategies for multi-path tcp in the nornet testbed. In *2015 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1–6. IEEE, 2015.

[10] S Seo. Kt's giga lte. *Presentation at IETF*, 93, 2015.

[11] From blockers for advertising calls to boosters for internet access: new services for your home. https://www.swisscom.ch/en/about/news/2016/11/20161115-mm-neue-service-fuer-daheim.html, November 2016.

[12] G Detal, S Barré, B Peirens, and O Bonaventure. Leveraging multipath tcp to create hybrid access networks.

[13] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, Janardhan Iyengar, et al. Architectural guidelines for multipath tcp development. *IETF, Informational RFC*, 6182:2070–1721, 2011.

[14] Multipath tcp - linux kernel implementation. https://multipath-tcp.org/pmwiki.php/Main/HomePage, 2020.

[15] Kun Wang, Thomas Dreibholz, Xing Zhou, Fu Fa, Yuyin Tan, Xi Cheng, and Qining Tan. On the path management of multi-path tcp in internet scenarios based on the nornet testbed. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 1–8. IEEE, 2017.

[16] Luca Boccassi, Marwan M Fayed, and Mahesh K Marina. Binder: a system to aggregate multiple internet gateways in community networks. In *Proceedings of the 2013 ACM MobiCom workshop on Lowest cost denominator networking for universal access*, pages 3–8, 2013.

[17] Fan Yang, Paul Amer, and Nasif Ekiz. A scheduler for multipath tcp. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7. IEEE, 2013.

[18] Bruno Yuji Lino Kimura and Antonio Alfredo Frederico Loureiro. Mptcp linux kernel congestion controls. *arXiv preprint arXiv:1812.03210*, 2018.

[19] Marcus Leech, Matt Ganis, Y Lee, Ron Kuris, David Koblas, and L Jones. Rfc1928: Socks protocol version 5, 1996.

[20] Openmptcprouter. https://www.openmptcprouter.com/, 2020.

[21] Mininet overview. http://mininet.org/overview/, 2020.

[22] Jon Dugan, Seth Elliott, Bruce A Mah, Jeff Poskanzer, and Kaustubh Prabhu. Iperf3: The tcp/udp bandwidth measurement tool. *URL https://iperf. fr*, 2005.

[23] Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. Generic routing encapsulation (gre). Technical report, RFC 2784, March, 2000.

[24] Maël Kimmerlin, Peer Hasselmeyer, and Andreas Ripke. Multipath cloud federation. In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pages 1–6. IEEE, 2017.

[25] Mininet-wifi. `https://mininet-wifi.github.io/get-started/`, 2020.