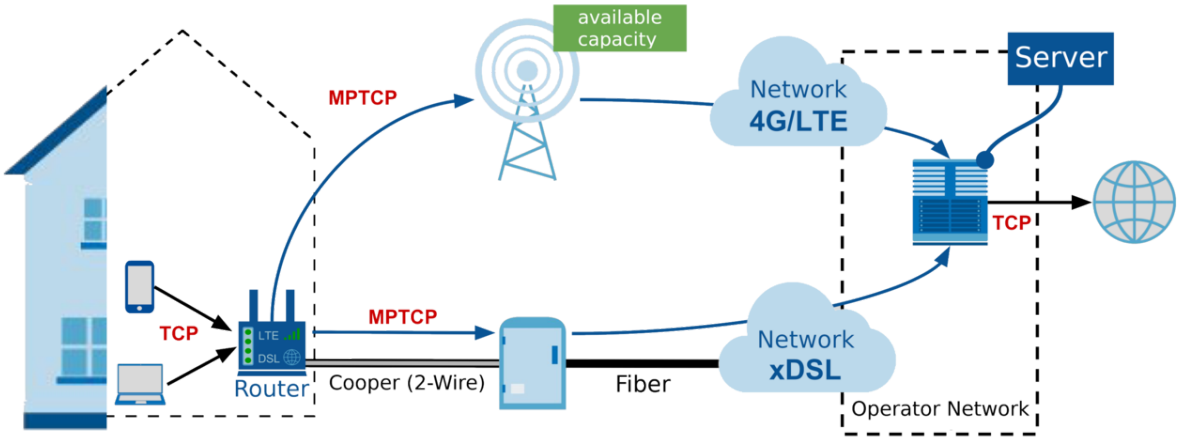
1. Einleitung

1.1. Problematik

Dieses Projekt soll dazu dienen, dass Nutzer mit einem beliebigen Endgerät die Möglichkeit haben das Multipath-TCP (MPTCP) Protokoll zu nutzen. Um dieses Protokoll nutzen zu können sind Einstellungen am Kernel notwendig, die bei den meisten Betriebssystemen nicht trivial sind. In Rahmen dieses Projekts wird es sich zu nutzen gemacht, dass es eine Implementierung eines *MPTCP Linux-Kernels* gibt, der einfach installiert werden kann

1.2. MPTCP

Multipath-TCP ist ein Standart der Transportschicht beschrieben in RFC 6824. Es beschreibt die logische Zusammenfassungen mehrerer TCP Verbindungen zu einer, um die Bandbreite bestenfalls zu erhöhen. Dabei kann ausgenutzt werden, dass ein Endgerät mehrere verschiedene Netzwerkanbindungen besitzt um somit die Bandbreite der gesamten Verbindung auf annähernd die Summe der Bandbreite der einzelnen Anbindungen zu erhöhen.



Quelle

Die Grafik zeigt einen beispielhaften Aufbau einer *MPTCP-Verbindung* über DSL und LTE zum selben Server. Hier könnte also zum Beispiel eine Datei über beide Interfaces übertragen werden.

1.3. Idee

Um die Problematik mit den unterschiedlichen Betriebssystemen zu umgehen, soll hier ein Socks-Proxy so angepasst werden, dass er anhand einer Konfigurationsdatei die Socket Einstellungen anpasst. Es müssen also alle IP-Adressen der Webseiten in die Datei geschrieben werden, zu denen man eine *MPTCP-Verbindung* aufbauen möchte. Zusätzlich können einige Parameter dabei eingestellt werden. Wird dann vom Nutzer auf dem Endgerät der Proxy verwendet kann dieser stellvertretend eine *MPTCP-Verbindung* aufbauen. Es wird also eine reguläre Verbindung zwischen Proxy und Nutzer aufgebaut und nur zwischen Proxy und Server wird das MPTCP-Protokoll verwendet. Damit dieser Aufbau sinnvoll angewendet werden

kann, muss also das Bottleneck zwischen Proxy und Server liegen, da sonst keine Bandbreite gewonnen werden kann.

Um die Einstellungen am Server nicht nur statisch zu halten, kann die konifgurationsdatei mithilfe einer REST-API manipuliert werden. Es ist also zur Laufzeit möglich Einstellungen von Verbindungen zu verändern. Eine solche Änderung wird allerdings erst ab der nächsten Verbindung zum angegebenen Ziel wirksam, da ein aktiver Socket nicht mehr verändert werden kann.

2. Inbetriebnahme

2.1. Einrichten des Kernels

Um den bereitgestellten Proxy mit MPTCP im vollen Umfang nutzen zu können muss MPTCP installiert und eingerichtet werden. Dafür müssen folgende Schritte ausgeführt werden:

- Installation des Kernels
- Konfiguration der Routing Tabellen
- Konfiguration der MPTCP Settings **(MPTCP_ENABLE Feld muss auf 2 gesetzt werden)**

Eine detaillierte Anleitung dazu findet man hier.

Nachdem die Installation abgeschlossen ist, muss der Kernel im Normalfall jedes Mal manuell gestartet werden. Das kann man machen, indem man während dem Neustart des Betriebssystems im richtigen Moment die Taste **Shift** drückt, was den Grub Bootmanager öffnet. Wählt man dann den Punkt *Advanced Options for Ubuntu* aus, kann man den MPTCP-Kernel auswählen und starten. Wird das in einer virtuellen Maschine (VirtualBox) gemacht, merkt man einen Leistungseinbruch und eine verkleinerte Anzeige, das ändert allergings nichts an der Funktionalität.

2.2 Starten der Programme

Damit der Server voll Funktionsfähig ist, müssen zwei Programme gestartet werden.

Erstens wird der eigentliche Socks-Proxy gestartet: `./microsocks`. Sollte im Programm eine Änderung gemacht werden oder der Code noch nicht Kompiliert sein, kann das durch das Makefile einfach den Befehl `make microsocks` ausführen.

Zweitens muss man noch den Server mit der REST API starten. Im Code muss allerdings in der letzten Zeile erst die IP-Adresse der eigenen Maschine eingetragen werden. Danach kann in einem neuen Terminal die virtuelle Umgebung mit dem Befehl `source flask/bin/activate` gestartet werden. Danach kann der Server gestartet werden `python ./flask/REST.py` .

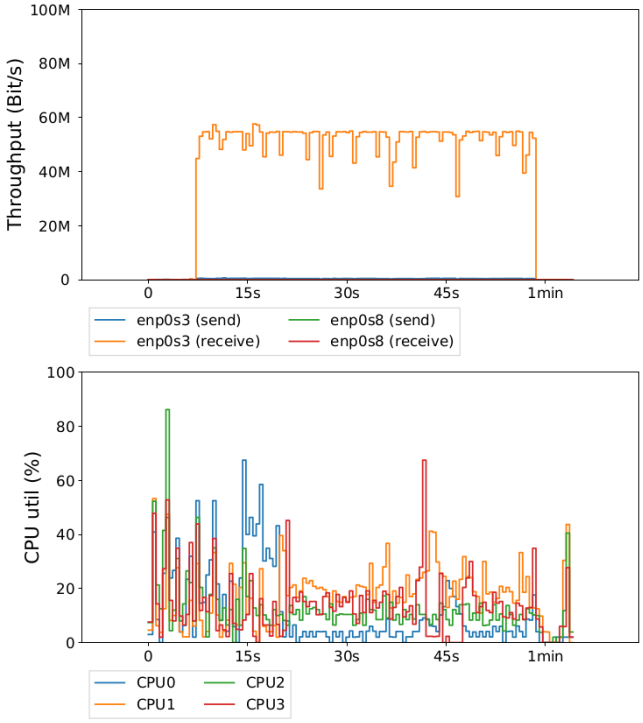
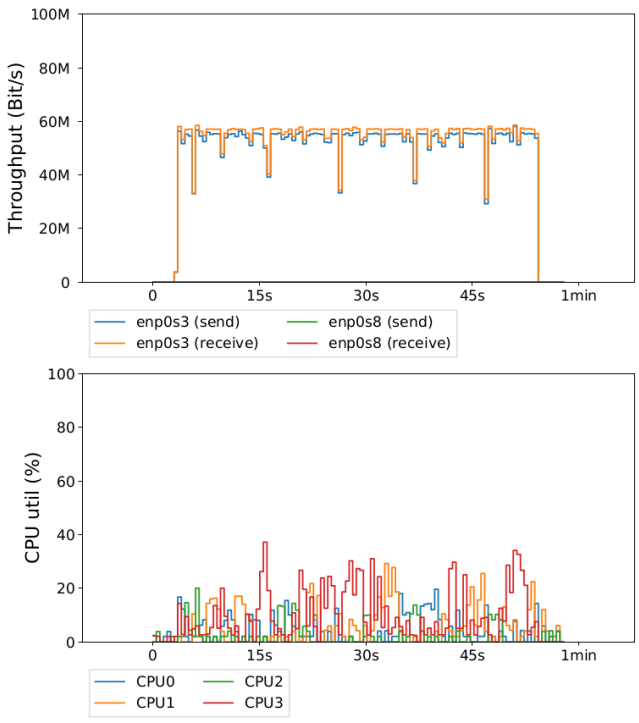
Sobald beide Programme laufen, ist der Proxy für eine beliebige Maschine einsatzbereit. Dafür muss in der entsprechenden Software (z.B. Firefox) einfach der Proxy in den Einstellungen aktiviert, die IP des Servers eingetragen und der Port 1080 angegeben werden. Die REST Befehle lassen sich mit einer beliebigen Software (z.B. Postman oder Visual Studio Code) absetzen.

3. Anpassungen an der Software

3.1 Auswahl des Proxys

Im Rahmen des Projektes war das Ziel nicht einen eigenen Proxy zu schreiben, sondern nur einen geegneten zu finden und diesen so weit zu verändern, dass die Beschriebene Funktionalität umgesetzt werden kann. Es wurde also eine Implementierung gesucht, die so wenig ovehead wie möglich produziert und somit - wenn möglich - keinen Einfluss auf die Übertragungsgeschwindigkeit nimmt. Außerdem war ein weiteres essentielles Kriterium, dass es in der Implementierung gut möglich ist die Socket Parameter einzustellen.

Die Wahl fiel schlussendlich auf den sogenannten microsocks, der durch einen relativ kurzen C-Code sehr übersichtlich ist. Um sicherzustellen, dass auch die Performance zufriedenstellend ist wurden noch einige Tests gemacht. Dafür wurde je eine ca. 300MB große Datei heruntergeladen, einmal mit Proxy und einmal ohne und die CPU Auslastung wurde gegenübergestellt. Ein Auszug aus den Ergebnissen wird hier dargestellt:

Ohne Proxy	**Mit Proxy**
	

In dieser Stichprobe ist zu sehen, dass die CPU Auslastung sehr schwankend ist, aber mit aktivem Proxy kaum erhöht ist. Die Übertragungsrate ist in beiden Fällen in etwa bei 50Mbps, was der Bandbreite des Anschlusses entspricht. Durch die Messungen wurde festgestellt, dass durch die Verwendung des Proxys kein Einbruch der Verbindungsgeschwindigkeit merkbar ist. Auch die CPU Auslastung wird durch die Verwendung kaum erhöht.

3.2 Anpassung des Proxys

Wie schon oben beschrieben muss der Proxy nur insofern angepasst werden, dass er aus einer Konfigurationsdatei die vorhergesehenen IP-Adresse - Konfiguaration Paare ausgelesen und für die

entsprechenden Verbindungen gesetzt werden. Für die Anpassung ist also der einzige Berührungspunkt die Datei in der die Verbindung zum Ziel aufgebaut wird, genauer die Stelle an der das Socket erstellt wird. Diese Implementierung ist in der Datei sockssrv.c in der Methode *connect_socks_target* zu finden. An dieser Stelle wird dann die Datei eingelesen und die jeweiligen IP-Adressen in der Datei werden mit dem aktuellen Ziel verglichen. Wenn eine passende Adresse in der Datei gespeichert ist, dann werden die abgespeicherten Einstellungen verwendet. Gibt es keinen Treffer, dann werden die Standarteinstellungen verwendet.

3.3 Implementieren der REST Schnittstelle

Für die Implementierung der REST-Schnittstelle wurde Python mit dem Framework Flask verwendet. Alles was das Programm machen muss, ist die verschiedenen Requests entgegenzuehmen und entsprechend zu Antworten und gegebenenfalls die Datei zu verändern. Der Inhalt eines POST-Requests ist im JSON Format wie folgt:

```
{
    "enable": "{0|1}",
    "ip": "<dns-name>",
    "path_manager": "{default|fullmesh|ndiffports|binder}",
    "scheduler": "{default|roundrobin|redundant}"
}
```