# Programmiermodelle und Unterstützung für umfassende Evaluationen im Umfeld von MPTCP Schedulern, Adaptionsentscheidungen und DASH Video Streaming[1]

Alexander Frömmgen[2]

**Abstract:** In der Dissertation wird aufgezeigt, dass die Analyse, die Umsetzung und die Evaluation von Kommunikationssystemen durch *i)* fehlende Abstraktionen und die resultierende Implementierungskomplexität sowie *ii)* die benötigten umfassenden Evaluationen für die Vielzahl an Konfigurationsmöglichkeiten und Netzwerkumgebungen erschwert werden.

Zur Überwindung dieser beiden Hindernisse präsentiert die Dissertation die folgenden Beiträge: *i)* drei Programmiermodelle für die Bereiche des Multipath TCP Paketschedulings, der adaptiven Kommunikationssysteme, sowie der Topologie-Adaption in Kommunikationssystemen, *ii)* 13 neue, ausführbare Multipath TCP Paketscheduler, sowie *iii)* ein wiederverwendbares Rahmenwerk zur nahtlosen Ausführung und Analyse umfassender Netzwerkexperimente.

## 1 Einführung

Kommunikationssysteme und das Internet sind in den letzten Jahren zu der zentralen Infrastruktur geworden. Dieser Beitrag gibt einen Überblick über die Dissertation des Autoren, welche sich der Weiterentwicklung heutiger Kommunikationssysteme, insbesondere mit Methoden und Werkzeugen zur Weiterentwicklung heutiger Kommunikationssysteme, beschäftigt.[3] In der Arbeit wird aufgezeigt, dass die Analyse, die Umsetzung und die Evaluation von Kommunikationssystemen durch *i)* fehlende Abstraktionen und die resultierende Implementierungskomplexität sowie *ii)* die benötigten umfassenden Evaluationen für die Vielzahl an Konfigurationsmöglichkeiten und Netzwerkumgebungen erschwert werden. Multipath TCP (MPTCP) [Fo13], das de facto Transportprotokoll zur Nutzung mehrerer Netzwerkpfade, stellt ein prominentes Beispiel dar. Innovationen im Bereich der Multipath TCP Paketscheduler werden durch die Implementierungskomplexität innerhalb des Linux-Kernels und den benötigten umfassenden Analysen für die Vielzahl an Anwendungen und Netzwerkumgebungen erschwert.

Zur Überwindung des ersten Hindernisses schlagen wir *ProgMP*, das erste Programmiermodell für Multipath TCP Paketscheduler, als Abstraktion für den Entwurf und die Entwicklung von Multipath TCP Paketschedulern vor [Fr17]. *ProgMP* beinhaltet eine aus-

[1] Englischer Originaltitel der Dissertation: Programming Models and Extensive Evaluation Support for MPTCP Scheduling, Adaptation Decisions, and DASH Video Streaming

[2] Die Dissertation ist im Rahmen der Tätigkeit im Sonderforschungsbereich MAKI an der TU Darmstadt an den Fachgebieten DVS und KOM entstanden. Kontakt: alexander.froemmgen@kom.tu-darmstadt.de

[3] Für eine detaillierte Darstellung der Beiträge und eine korrekte Würdigung der Grundlagen und der verwandten Arbeiten verweisen wir an dieser Stelle für das gesamte Dokument auf die Dissertationsschrift [Fr18a].

drucksstarke Spezifikationssprache und eine einfache Programmierschnittstelle zur Spezifikation ausführbarer Multipath TCP Paketscheduler. Wir zeigen die Stärken von *ProgMP* am Beispiel 13 neuer Paketscheduler mit unterschiedlichen Optimierungszielen auf [Fr17, FHK18]. Wir schlagen den ersten redundanten Multipath TCP Paketscheduler vor und zeigen, dass dieser die Latenz für Anwendungen mit strikten Latenz- und moderaten Durchsatzanforderungen signifikant reduziert [Fr16]. Wir nutzen *ProgMP* für eine detaillierte Analyse von Entwurfsentscheidungen für die Verwendung von Redundanz zur Abwägung von Latenz und Durchsatz. Des Weiteren schlagen wir Paketscheduler vor, welche feingranulare Durchsatz- oder Latenzziele einhalten, sowie die Interaktion mit darüberliegenden Protokollen wie beispielsweise HTTP/2 optimieren und dabei Pfadpräferenzen einhalten. Unsere detaillierten Evaluationen mittels Netzwerkemulation sowie Echtweltmessungen zeigen, dass *ProgMP* effiziente Schedulingentscheidungen und eine Vielzahl neuer, ausführbarer Multipath TCP Paketscheduler ermöglicht. Neben *ProgMP*, unserem Hauptbeitrag zur Überwindung fehlender Abstraktionen, stellen wir des Weiteren Programmiermodelle als Abstraktionen für die Adaptionsentscheidung adaptiver Kommunikationssysteme vor. So schlagen wir vor, Adaptionsentscheidungen mittels *event condition action* Regeln (Ereignis, Bedingung, Aktion) zu spezifizieren und diese Regeln basierend auf genetischer Programmierung in umfassenden Netzwerkexperimenten automatisch für eine gegebene Nutzenfunktion zu lernen [Fr15]. Schließlich stellen wir ein Programmiermodell für die Spezifikation von Topologie-Adaptionen in Kommunikationssystemen mittels Graphmustern in der Topologie vor [St16a] und zeigen, wie die darunterliegende Suche nach Graphmustern verteilt ausgeführt werden kann [St18].

Zur Überwindung des zweiten identifizierten Hindernisses haben wir das *MACI* Rahmenwerk zur Verwaltung, skalierbaren Ausführung und interaktiven Analyse von umfassenden Netzwerkexperimenten entwickelt [Fr18b]. Im Kern ist *MACI* eine Kombination und Integration etablierter Werkzeuge zur Förderung gründlicher, nahtloser Evaluationen während des gesamten Forschungsprozesses. Wir diskutieren unsere *MACI* Erfahrungen während *i)* der Entwicklung und Evaluation unserer vorgeschlagenen *ProgMP* Scheduler [Fr17, FHK18], *ii)* der Entwicklung einer Multipatherweiterung für das Transportprotokoll QUIC [Vi18] *iii)* der Analyse eines verteilten Protokolls zur Auffindung von Graphmustern in Topologien [St18], sowie *iv)* eines systematischen Vergleichs von *DASH Video Streaming* Implementierungen [St17]. Unsere Erfahrungen bestätigen, dass *MACI* wiederkehrende Aufgaben in der Evaluation diverser Kommunikationssysteme unterstützt und dadurch die Forschungseffizienz signifikant erhöht. Die Experimente mit *MACI* für *ProgMP*, die Multipatherweiterung für QUIC, die Topologiemustererkennung und die Videoübertragung mittels *DASH* gehen über eine Evaluation von *MACI* hinaus und stellen signifikante Beiträge zum Verständnis der jeweiligen Gebiete dar.

Im Folgenden gehen wir exemplarisch auf zwei Beiträge der Dissertation genauer ein: *i)* das *ProgMP* Programmiermodell für Multipath TCP Paketscheduler sowie *ii)* das *MACI* Rahmenwerk für umfassende Netzwerkexperimente.

# 2 ProgMP: Ein Programmiermodell für MPTCP Paketscheduler

## 2.1 Einleitung und Motivation

Das Transportprotokoll TCP liegt den meisten heutigen Kommunikationssystemen zugrunde und ermöglicht die Kommunikation in verteilten Systemen und Anwendungen. Multipath TCP ist eine kürzlich im `RFC 6824` vorgeschlagene TCP-Erweiterung, welche es mittels des Konzepts sogenannter *Subflows* ermöglicht mehrere Netzwerkpfade für eine logische MPTCP Transportprotokollverbindung zu verwenden [Fo13]. Es wurde gezeigt, dass Multipath TCP den Durchsatz und die Zuverlässigkeit der Transportprotokollverbindung erhöhen kann und dabei dynamisch auf Veränderungen und Schwankungen der verfügbaren Netzwerkresourcen reagiert.

Multipath TCP erhöht die Komplexität im Vergleich zu traditionellem TCP, da Pakete eines Datenstroms auf mehrere *Subflows* verteilt werden müssen, bevor diese auf der Empfangsseite wieder zusammengeführt werden (Abbildung 1). Dafür benötigt Multipath TCP eine Instanz, welche für jedes zu übertragende Datenpaket entscheidet, auf welchem *Subflow* bzw. Netzwerkpfad dieses übertragen werden soll. Diese Entscheidungsinstanz wird Paketscheduler oder auch vereinfacht Scheduler genannt. Die *Paketschedulingentscheidung* hat einen großen Einfluss auf die Leistungsmerkmale des Transportprotokolls. So können ungeeignete Entscheidungen die Vorteile von Multipath TCP verschwinden lassen oder sogar die Gesamtleistung verringern.

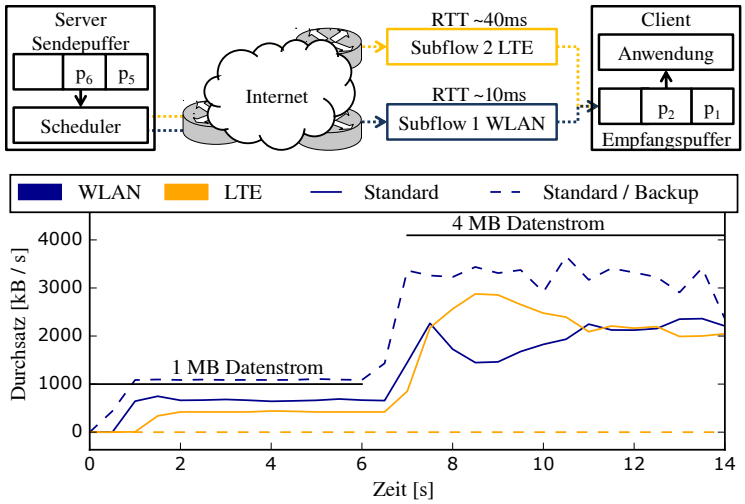


Abb. 1: **Bedarf an flexibleren Schedulern:** Versuchsaufbau und reproduzierbare Messergebnisse für eine interaktive Netzwerkanwendung mit dem MPTCP Standardscheduler *MinRTT* unter Verwendung eines WLAN- und eines LTE-*Subflows*. Weder der Standardscheduler noch der *Backup Modus* des Standardschedulers ermöglichen es, die Videobitrate von 4MB/s sicherzustellen und dabei präferenzbewusst den WLAN-*Subflow* auszuschöpfen bevor der LTE-*Subflow* verwendet wird.

Die heutigen MPTCP Implementierungen sind zumeist auf hohen Durchsatz optimiert. Viele Anwendungen benötigen jedoch für die jeweiligen Anforderungen und Charakteristika angepasste Schedulingstrategien. So haben Anwendungen beispielsweise unterschiedliche Anforderugnen bezüglich der erfahrenden Netzwerklatenz und dem erreichten Durchsatz und viele Anwender präferieren die Verwendung von WLAN über LTE für mobile Anwendungen. Bis zur Erstellung dieser Dissertation fehlten sowohl in der Forschung als auch in den Implementierungen Konzepte und Methoden zur Spezifikation und Verwendung optimierter anwendungs- und präferenzbewusster MPTCP Paketscheduler.

Abbildung 1 illustriert diese Limitationen anhand einer exemplarischen Messung einer interaktiven Videoübertragung. Für die Messung wurde eine MPTCP Verbindung mit einem WLAN- und einem LTE-*Subflow* zwischen einer AWS Serverinstanz und einem Laptop verwendet. Die ersten sechs Sekunden des Videostroms sind mit 1MB/s kodiert, der übrige Videostrom mit 4MB/s. Obwohl der 1MB/s Datenstrom vollständig auf dem WLAN-*Subflow* mit 10ms Umlaufzeit übertragen werden könnte, beobachten wir, dass der heutige Standardscheduler *MinRTT* ungefähr 30% des 1MB/s Stroms über den LTE-*Subflow* mit 40ms Umlaufzeit übertragt. Dies ist eine Folge der Durchsatz- und Ressourcenausgleichsoptimierungen des *MinRTT* Schedulers im Zusammenspiel mit der Staukontrolle und einer TCP-Optimierung für kleine Paketpuffer. Eine weitere Messung unter Verwendung des *Backup Modus* für den LTE-*Subflow* zeigt, dass dieser Modus den benötigten Durchsatz für den 4MB/s Strom nicht bereitstellen kann, da er effektiv zu einer Deaktivierung des LTE-*Subflows* führt.

Dieses symptomatische Beispiel zeigt das Fehlen eines einzelnen, optimalen Schedulers für alle Anwendungsfälle auf. Stattdessen benötigt MPTCP optimierte Paketscheduler zur Berücksichtigung von Anwendungsanforderungen und Pfadpräferenzen. Gleichzeitig beobachten wir, dass die Entwicklung, die Evaluation und die Verwendung neuer Multipath TCP Scheduler durch die hohe Implementierungskomplexität, etwa in der Standardimplementierung im Linux-Kernel, behindert wird. Entsprechend finden sich viele Vorschläge für optimierte Scheduler, welche nicht in einer Echtweltumgebung evaluiert wurden bzw. komplexe Änderungen an der zugrundeliegenden MPTCP Implementierung vermeiden.[4] Der Bedarf an optimierten Schedulern und die fehlenden Konzepte zur Erstellung dieser zeigt die Notwendigkeit an Abstraktionen zum effizienten Entwurf und zur Umsetzung von Multipath TCP Schedulern auf.

### 2.2 Das ProgMP Programmiermodell

Im Rahmen der Dissertation schlagen wir *ProgMP*, das erste Programmiermodell für den Entwurf und die Umsetzung von Multipath TCP Schedulern [Fr17], vor.[5] *ProgMP* besitzt eine Schedulerspezifikationssprache, welche es Kommunikationssystemforschern und Anwendungsentwicklern ermöglicht, Multipath TCP Scheduler auf einem hohen Abstraktionsniveau zu beschreiben. Neben der Spezifikationssprache beinhaltet *ProgMP* eine Programmierschnittstelle, welche es Anwendungen erlaubt, zur Laufzeit Informatio-

[4] Die Dissertation [Fr18a] beinhaltet eine Übersicht über 27 Paketscheduler im weiteren Multipath TCP Umfeld.
[5] Dokumentation, Implementierung und Beispiele zu *ProgMP* sind auf `https:/progmp.net` verfügbar.

nen und Hinweise an den Scheduler zu übergeben. Schließlich stellen wir eine effiziente Ausführungsumgebung für die spezifizierten Scheduler in der Multipath TCP Linux Kernelimplementierung bereit. Somit schließt *ProgMP* die Lücke zwischen einer einfachen, ausdrucksstarken Beschreibung der Multipath TCP Scheduler und der Erprobung und Nutzung von Schedulerinnvoationen in realen Anwendungen. Im Folgenden stellen wir weitere Details von *ProgMP* vor.

**Spezifikation von Schedulern** Die Spezifikationssprache von *ProgMP* ist eine domänenspezifische Sprache, welche Ausdrucksstärke, Verständlichkeit und Ausführbarkeit abwägt. Es sind alle relevanten Entitäten aus der Paketschedulingdomäne in der Sprache enthalten. So gibt es in der Sprache die drei Paketpuffer *i)* `Q` für die zu sendenden Pakete, *ii)* `QU` für die sich in der Übertragung befindenden Pakete, sowie *iii)* `RQ` für Pakete welche erneut übertragen werden müssen. Die Sprache bietet einfache Kontrollflussoperationen sowie deklarative Sprachkonstrukte zur Auswahl der Pakete aus den Puffern beziehungsweise des zu nutzenden Subflows aus der Menge aller Subflows `SUBFLOWS`. Das implizite, statische Typsystem reduziert im Zusammenspiel mit unveränderlichen Variablen mögliche Seiteneffekte während der Ausführung und ermöglicht eine effiziente Ausführung sowie eine einfache Fehlerbehandlung. Darüber hinaus kann eine einzelne Schedulerausführung kein, ein, oder mehrere Pakete unterschiedlichen Subflows zuweisen. Dies vereinfacht viele Schedulerspezifikationen erheblich und reduziert die Komplexität und somit die Fehlerwahrscheinlichkeit im Bereich der Zustandsverwaltung für die Schedulerentwicklerin.

Abbildung 2 stellt zur Illustration der Schedulerspezifikationssprache Auszüge aus zwei alternativen Schedulern zur Verwendung von Redundanz dar. Beide Scheduler nehmen an, dass `sbfCandidates` eine Menge an potenziellen Subflowkandidaten beinhaltet. Die erste Alternative (links) sendet (`PUSH`t) das nächste zu sendende Paket (`Q.TOP`) auf allen Subflowkandidaten und entfernt das Paket aus dem Puffer falls zumindest ein Subflowkandidat vorhanden ist. Die zweite Alternative (rechts) nutzt zwei unterschiedliche Strategien in Abhängigkeit vom Zustand des Sendepuffers. Befinden sich ungesendete Pakete in diesem (`!Q.EMPTY`) bevorzugt der Scheduler diese neuen Pakete über bereits gesendete Pakete und überträgt sie auf dem Subflow mit der minimalen Umlaufzeit (`RTT`). Sind alle Pakete

```
1 /* Alternative 1 für Redundanz */
2 ...
3 IF(!sbfCandidates.EMPTY) {
4   FOREACH(VAR sbf IN sbfCandidates) {
5     sbf.PUSH(Q.TOP);
6   }
7   DROP(Q.POP());
8 }
```

```
1  /* Alternative 2 für Redundanz */
2  ...
3  IF (!Q.EMPTY) {
4    sbfCandidates.MIN(sbf => sbf.RTT).PUSH(Q.POP());
5    RETURN;
6  }
7
8  VAR packetCandidate = QU.FILTER(packet =>
9    !sbfCandidates.FILTER(sbf =>
10     !packet.SENT_ON(sbf)).EMPTY).TOP;
11 sbfCandidates.FILTER(sbf =>
12   !packetCandidate.SENT_ON(sbf)).MIN(sbf => sbf.RTT).
13     PUSH(packetCandidate);
14 }
```

Abb. 2: Ausschnitte aus zwei *ProgMP* Schedulern, welche Pakete redundant auf mehreren Pfaden übertragen. Der linke Scheduler überträgt das Paket auf allen *Subflows*, welche zum Zeitpunkt der Entscheidung verfügbar sind. Der rechte Scheduler hingegen überträgt nur dann Pakete redundant wenn keine ungesendeten Pakete vorhanden sind.

zumindest auf einem *Subflow* übertragen nutzt der Scheduler Redundanz und übertragt die Pakete erneut auf einem anderen Subflow. Die beiden Beispiele zeigen, dass *ProgMP* es ermöglicht, komplexe Schedulingentscheidungen präzise in wenigen Zeilen zu spezifizieren. Im Vergleich dazu benötigt der traditionelle Ansatz mit Linux-Kernelmodulen 251 Zeilen C Programmcode für den redundanten Scheduler.

**Die Ausführungsumgebung** Wir haben eine *ProgMP*-Ausführungsumgebung im Multipath TCP Linux-Kernel implementiert und evaluiert. Die Ausführungsumgebung besteht aus einem Interpreter sowie einem eBPF-basierten Just-in-Time Compiler zur Sprachausführung, einer Implementierung der Pufferabstraktion und einer Umsetzung der Fehlerbehandlungsabstraktion. Darüber hinaus wird Domänenwissen genutzt, um die Schedulerausführung zur Laufzeit zu optimieren. So kann ein Scheduler etwa zur Laufzeit unter der Annahme einer festen Anzahl an *Subflows* optimiert und erneut kompiliert werden.

**Vorgestellte und Evaluierte Scheduler** Im Rahmen der Dissertation haben wir *ProgMP* zur Analyse und Verbesserung etablierter sowie zum systematischen Entwurf und zur Evaluation neuer MPTCP Scheduler verwendet. Die betrachteten Scheduler (Tabelle 1) optimieren dabei Leistungseigenschaften entsprechend unterschiedlicher Anwendungsanforderungen und Nutzer- und Pfadpräferenzen. Die Vorstellung und Evaluation der Scheduler stellt nicht nur eine Evaluation der Ausdrucksstärke, Tragweite und Anwendbarkeit von *ProgMP* dar, sondern ist darüber hinaus ein fundamentaler Beitrag zum Verständnis des Entwurfsraums an MPTCP Schedulern und eine signifikante Steigerung der MPTCP Leistung in vielen Anwendungsgebieten.

| | | Diskutierte Varianten | Anzahl Zeilen mit Präfix |
|---|---|---|---|
| **Etablierte Scheduler** | Präferenzen | | |
| Standard *MinRTT* | | 1 | 15 |
| *Backup Modus* | Binär | 1 | 15 + 7 = 22 |
| Round robin | Binär | 2 | 21 und 35 |
| Redundant[6] | Binär | 1 | 21 |
| **Neue Scheduler und Funktionen** | | | |
| Aktives Ausprobieren für zeitnahe RTT-Abschätzungen | | 3 | 31, 34 und 38 |
| Abwägen von Latenz und Ressourceneinsatz mittels Redundanz | | 3 | 17, 21 und 24 |
| Übertragungszeitminimierung in heterogenen Umgebungen | | 1 | 28 |
| Präferenzbewusstes Einhalten tolerierbarer RTTs | | 2 | 23 und 26 |
| Präferenzbewusstes Bereitstellen des benötigten Durchsatzes | | 2 | 22 und 35 |
| Einwegeverzögerungsbewusstes Scheduling | | 1 | 15 |
| Adaptives Scheduling für HTTP/2 | | 1 | 26 |

Tab. 1: Mit *ProgMP* analysierter Entwurfsraum an Multipath TCP Schedulern. Da die meisten Scheduler den gleichen Präfix von 11 Zeilen nutzen, werden effektiv zwischen 4 und 27 weitere Spezifikationszeilen benötigt. Die *ProgMP*-Spezifikationen der etablierten Scheduler benötigten 3,6% bis 12.4% der Anzahl an Zeilen der C-basierten Kernelmodule.

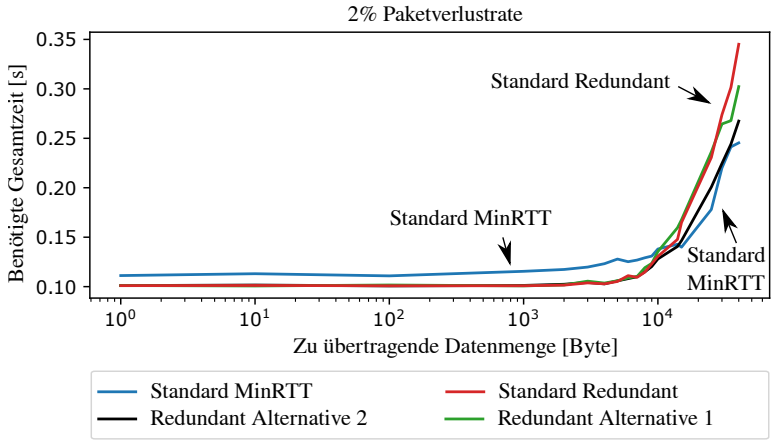


Abb. 3: Durchschnittlich benötigte Übertragungszeit in Abhängigkeit der zu übertragenden Datenmenge in jeweils 20 Experimenten mit Mininet, zwei homogenen *Subflows* und 2% Paketverlustrate.

Abbildung 3 zeigt exemplarisch einen experimentellen Vergleich der in Abbildung 2 vorgestellten Redundanzalternativen sowie des Standard *MinRTT* und redundanten Schedulers. Die Messung zeigt, dass alle drei Redundanzalternativen für geringe Datenmengen einen Vorteil gegenüber dem Standardscheduler haben. Für größere Datenmengen hingegen unterscheiden sich die Alternativen. Hier ist der Standardscheduler und Alternative 2 den beiden anderen redundanten Schedulern überlegen. Die exemplarische Messung zeigt, dass Scheduler durch die direkte Ausführbarkeit von *ProgMP*-Spezifikationen empirisch evaluiert und analysiert werden können und somit Schedulerinnovationen signifikant vereinfacht werden.

## 3 MACI: Ein Rahmenwerk für die wiederkehrenden Anforderungen umfassender Netzwerkexperimente

Die Forschung an Kommunikationssystemen ist auf Netzwerkexperimente angewiesen. Entsprechend wurden innerhalb der Forschungsgemeinschaft Methoden und Werkzeuge für kontrollierte und reproduzierbare Netzwerkexperimente entwickelt – etwa Netzwerksimulatoren und deren zugrundeliegenden Netzwerkmodellen. So ist eine Vielzahl an Simulatoren und Emulatoren verfügbar, welche auf unterschiedliche Anwendungen, unterliegende Abstraktionen und Netzwerkmodelle ausgerichtet sind. Experimente mit diesen Ausführungsumgebungen sind die Grundlage für den Entwurf und die Entwicklung heutiger Kommunikationssysteme und ermöglichen eine frühe und wiederkehrende Rückmeldung und Analyse der Systeme.

[6] Der redundante Scheduler wurde im Rahmen der Dissertation entwickelt [Fr16]. Er wird hier dennoch als *etabliert* klassifiziert, da er ursprünglich ohne *ProgMP* entwickelt wurde und einer von drei verfügbaren Schedulern für den MPTCP Linux-Kernel ist (`https://github.com/multipath-tcp/mptcp/blob/mptcp_v0.94/net/mptcp/mptcp_redundant.c`).

Im Rahmen der Dissertation haben wir beobachtet, dass Forscher und Entwickler wiederkehrend unterstützende Infrastruktur und Werkzeuge entwickeln, um die Ausführung und die Auswertung von Experimenten zu automatisieren. Die Entwicklung dieser Werkzeuge startet üblicherweise für jedes Forschungsprojekt von Neuem. Die Entwicklung beginnt meist mit kleinen Hilfsprogrammen, welche im Verlauf des Forschungsprojekts immer umfassender werden. Insgesamt nehmen sie schließlich einen wesentlichen Anteil am Gesamtaufwand des Forschungsprojekts ein. Auch wenn die Entwicklung dieser Werkzeuge eine geringe Komplexität aufweist, lenkt sie von der eigentlichen Forschungsarbeit ab und verzögert das Projekt.

In der Dissertation identifizieren wir die *wiederkehrenden* Anforderungen von und Aufgaben für netzwerkexperimentbasierte Studien, etwa *i)* die Spezifikation, die Verwaltung und die Dokumentation der Experimente und ihrer abhängigen und unabhängigen Parameter, *ii)* die skalierbare, parallele Ausführung umfassender Experimentstudien, sowie *iii)* die interaktive Analyse der Experimentergebnisse basierend auf den zuvor spezifizierten Parametern. Darüber hinaus argumentieren wir, dass ein integrierter Ansatz für diese Aufgaben die (Forschungs-) Effizienz deutlich erhöht.

Basierend auf diesen Beobachtungen präsentieren wir *MACI*[7], das erste maßgeschneiderte Rahmenwerk um die zuvor identifizierten wiederkehrenden Anforderungen und Aufgaben zu erfüllen [Fr18b]. *MACI* ist ein Rahmenwerk für die nahtlose Verwaltung, skalierbare Ausführung und interaktive Auswertung umfassender Netzwerkexperimente. Die nahtlose Integration dieser Funktionen ermöglicht es beispielsweise, die Ergebnisse der Experimente ohne manuelle Aufbereitung der Daten visuell darzustellen und basierend auf den damit gewonnen Einsichten weitere Experimentkonfigurationen in wenigen Schritten zu starten. *MACI* entstand als Ergebnis und basierend auf den Erfahrungen aus mehreren Forschungsprojekten [St16b, Fr15, Kh16].

*MACI* stellt eine geschickte Integration und Kombination etablierter Werkzeuge dar um eine gründliche, experimentbasierte Evaluation während des gesamten Forschungsprozesses zu fördern. *MACI* wendet beispielsweise die Konzepte der interaktiven Datenanalyse aus dem Bereich des Business Intelligence und des Data Science auf Netzwerkexperimente an. *MACI* folgt dem Zeitgeist der agilen Softwareentwicklung und der kontinuierlichen Integration in der Softwareentwicklung, indem es Hindernisse für kurze Iterationszyklen eliminiert. Dabei wird die zugrundeliegende Ausführungsumgebung für das einzelne Netzwerkexperiment als austauschbare Blackbox betrachtet um eine Vielzahl an etablierten Ausführungsumgebungen, etwa Netzwerksimulatoren und Emulatoren, zu unterstüzen.

Wir haben *MACI* erfolgreich für die in Tabelle 2 dargestellten Forschungsprojekte verwendet. Diese nutzen verschiedene Ausführungsumgebungen und befassen sich mit unterschiedlichen Protokollen auf diversen Netzwerkschichten. Dies zeigt, das *MACI* für diverse Forschungsprojekte im Bereich der Kommunikationssysteme anwendbar ist. *MACI* hat dabei alle wiederkehrenden Aufgaben im Bereich der experimentellen Evaluation signifikant vereinfacht und es uns somit ermöglicht, uns auf die eigentlichen Forschungsfragen zu fokussieren.

[7] *MACI* ist auf `https://maci-research.net` öffentlich verfügbar.

| | Ausführungsumgebung | Schicht/Protokoll |
|---|---|---|
| **Von Bachelor- und Masterstudenten genutzt** | | |
| Nachbildung einer bekannten MPTCP Experimentstudie | Mininet | Transport/MPTCP |
| Entwicklung einer Multipath-Erweiterung für QUIC [Vi18] | Mininet | Transport/MPQUIC |
| Lernen und Evaluieren von Staukontrollen für QUIC | Mininet | Transport/QUIC |
| **Von Doktoranden genutzt** | | |
| Entwicklung neuer MPTCP Scheduler [Fr17, FHK18] | Mininet | Transport/MPTCP |
| Analyse einer verteilten Topologiemustererkennung [St18] | Java | Diverse |
| Analyse und Vergleich von DASH Implementierungen [St17] | Mininet | Anwendung/DASH |

Tab. 2: Übersicht über bisherige *MACI*-Verwendungen.

Exemplarisch möchten wir hier den experimentellen Vergleich der unterschiedlichen Redundanzalternativen für Multipath TCP Scheduler mittels *MACI* nennen. Neben der direkten visuellen Analyse (Abbildung 3 basiert auf einer der direkt verfügbaren Visualisierungen) haben wir von der skalierbaren, parallelen Ausführung der Experimente profitiert. So basiert die genannte Abbildung auf 30.720 Experimenten, von denen jedes fast eine Minute benötigt. Mit *MACI* waren wir in der Lage, diese Experimente mit 20 AWS Instanzen an einem Tag für 95$ durchzuführen. Eine sequenzielle Ausführung hätte hingegen über 20 Tage und die selben finanziellen Aufwendungen benötigt.

Schließlich stellen wir fest, dass die gemeinsame Verwendung eines Werkzeugs und das dadurch geförderte Folgen einer gemeinsamen Evaluationsmethode die Zusammenarbeit in den Forschungsprojekten deutlich vereinfacht hat. Darüber hinaus hat das Bereitstellen eines Werkzeugs und einer bewährten Methoden Bachelor- und Masterstudenten bei ihren ersten Forschungsschritten unterstützt.

## Literaturverzeichnis


[FHK18] Frömmgen, Alexander; Heuschkel, Jens; Koldehofe, Boris: Multipath TCP Scheduling for Thin Streams: Active Probing and One-way Delay-awareness. In: Proceedings of the IEEE International Conference on Communications (ICC). 2018.

[Fo13] Ford, Alan; Raiciu, Costin; Handley, Mark; Bonaventure, Olivier: TCP Extensions for Multipath Operation with Multiple Addresses, RFC 6824, IETF. 2013.

[Fr15] Frömmgen, Alexander; Rehner, Robert; Lehn, Max; Buchmann, Alejandro: Fossa: Learning ECA Rules for Adaptive Distributed Systems. In: Proceedings of the IEEE International Conference on Autonomic Computing (ICAC). S. 207–210, 2015.

[Fr16] Frömmgen, Alexander; Erbshäußer, Tobias; Zimmermann, Torsten; Wehrle, Klaus; Buchmann, Alejandro: ReMP TCP: Low Latency Multipath TCP. In: Proceedings of the IEEE International Conference on Communications (ICC). 2016.

[Fr17] Frömmgen, Alexander; Rizk, Amr; Erbshäußer, Tobias; Weller, Max; Koldehofe, Boris; Buchmann, Alejandro; Steinmetz, Ralf: A Programming Model for Application-defined

Multipath TCP Scheduling. In: Proceedings of the ACM/IFIP/USENIX Middleware Conference, **Best Paper Award**, `https://progmp.net`. ACM, S. 134–146, 2017.

[Fr18a] Frömmgen, Alexander: Programming Models and Extensive Evaluation Support for MPTCP Scheduling, Adaptation Decisions, and DASH Video Streaming, Dissertation, TU Darmstadt. 2018.

[Fr18b] Frömmgen, Alexander; Stohr, Denny; Rizk, Amr; Koldehofe, Boris: Dont Repeat Yourself: Seamless Execution and Analysis of Extensive Network Experiments. In: Proceedings of the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), `https://maci-research.net`. S. 20–26, 2018.

[Kh16] Khuda Bukhsh, Wasiur; Rizk, Amr; Frömmgen, Alexander; Koeppl, Heinz: Optimizing Stochastic Scheduling in Fork-Join Queuing Models: Bounds and Applications. In: Proceedings of the IEEE INFOCOM. 2016.

[St16a] Stein, Michael; Frömmgen, Alexander; Kluge, Roland; Löffler, Frank; Schürr, Andy; Buchmann, Alejandro; Mühlhäuser, Max: TARL: Modeling Topology Adaptations for Networking Applications. In: Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). ACM, S. 57–63, 2016.

[St16b] Stohr, Denny; Frömmgen, Alexander; Fornoff, Jan; Zink, Michael; Buchmann, Alejandro; Effelsberg, Wolfgang: QoE Analysis of DASH Cross-Layer Dependencies by Extensive Network Emulation. In: Proceedings of the SIGCOMM Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE). ACM, S. 25–30, 2016.

[St17] Stohr, Denny; Frömmgen[8], Alexander; Rizk, Amr; Zink, Michael; Steinmetz, Ralf; Effelsberg, Wolfgang: Where are the Sweet Spots?: A Systematic Approach to Reproducible DASH Player Comparisons. In: Proceedings of the ACM Conference on Multimedia (MM), `https://maci-research.net/dash`. S. 1113–1121, 2017.

[St18] Stein, Michael; Frömmgen, Alexander; Kluge, Roland; Lin, Wang; Wilberg, Augustin; Koldehofe, Boris; Mühlhäuser, Max: Scaling Topology Pattern Matching: A Distributed Approach. In: Proceedings of the ACM/SIGAPP Symposium on Applied Computing (SAC). 2018.

[Vi18] Viernickel, Tobias; Frömmgen, Alexander; Rizk, Amr; Koldehofe, Boris; Steinmetz, Ralf: Multipath QUIC: A Deployable Multipath Transport Protocol. In: Proceedings of the IEEE International Conference on Communications (ICC). 2018.

**Alexander Frömmgen** schloss im Jahr 2013 das Informatik-Studium an der Technischen Universität Darmstadt ab. Anschließend war Herr Frömmgen an den Lehrstühlen von Prof. Alejandro Buchmann und von Prof. Ralf Steinmetz als wissenschaftlicher Mitarbeiter im Sonderforschungsbereich MAKI tätig. Im Jahr 2018 wurde er am Fachbereich Informatik der TU Darmstadt zum Dr.-Ing. (mit Auszeichnung) promoviert. Herr Frömmgen arbeitet nun bei Google im Umfeld interaktiver Kommunikationsanwendungen.

[8] Die beiden ersten Autoren haben in gleichem Maße beigetragen.