# Abschlussreport Fachprojekt Routingalgorithmen

TE_SR_WAN_simulation & TE_SR_experiments

Marcel Gerhardt
TU Dortmund

Ayoub Lahrach
TU Dortmund

Xiaoyu Zhou
TU Dortmund

## Zusammenfassung

In diesem Fachprojekt entwickeln und evaluieren wir zwei Ansätze zur Reduktion der maximalen Linkauslastung (MLU) in Weitverkehrsnetzen: (i) **TL_Inverse**, eine auslastungsadaptive Erweiterung der klassischen *inverse_capacity*-Heuristik, und (ii) eine kapazitätsbewusste Demand-First-Waypoints-Variante **(DFW)** auf Basis von *Demand-First-Waypoints* mit inverse_capacity-Initialisierung und robustem ECMP-Splitting. In *Projekt 1 (TE_SR_WAN_simulation)* untersuchen wir beide Verfahren auf den Topologien **Abilene**, **Géant** und **Germany50** und vergleichen sie jeweils mit ihren Baselines. Die Ergebnisse zeigen topologieabhängige Vorteile: **TL_Inverse** senkt die MLU insbesondere in gut vernetzten Netzen (z. B. Géant), während **DFW** in Szenarien mit hinreichender Pfaddiversität leichte Mediangewinne erzielt, jedoch mit höherer Streuung. In *Projekt 2 (TE_SR_experiments)* entwerfen wir eine kompakte, SR-fähige Topologie mit bewusstem Innen-Bottleneck und vergleichen **TL_Inverse** gegen eine statisch getunte Dijkstra-Baseline (**New_Weights**) sowie **DFW** gegen eine **Weights**-Baseline. Beide Experimente werten wir über Boxplots der MLU aus. Abschließend replizieren wir die Resultate einer anderen Gruppe für beide Projektteile erfolgreich. Die verwendeten und erstellten Repositories sind am Ende gelistet.

## 1 Einleitung & Ziele

Traffic Engineering (TE) zielt auf eine Lastverteilung, die Engpässe vermeidet und vorhandene Kapazitäten effizient nutzt. Eine zentrale Zielgröße ist die *maximale Linkauslastung (MLU)*, d. h. der maximale Quotient aus Last und Kapazität über alle Kanten. Dieser Report adressiert zwei komplementäre algorithmische Ansätze zur MLU-Reduktion: **(i)** *TL_Inverse* als auslastungsadaptive Erweiterung von *inverse_capacity* und **(ii)** eine kapazitätsbewusste Demand-First-Waypoints-Variante **(DFW)** mit *inverse_capacity*-Initialisierung und robustem ECMP-Splitting.

Im Mittelpunkt stehen die Entwicklung und Evaluation von **TL_Inverse** und **DFW** auf den Topologien **Abilene**, **Géant** und **Germany50**, jeweils im Vergleich zu ihren Baselines. Darauf aufbauend übertragen wir beide Ansätze in ein zweites, SR-fähiges Repository mit eigens entworfener Topologie, wobei **TL_Inverse** gegen **New_Weights** und **DFW** gegen eine **Weights**-Baseline antritt. Abschließend prüfen wir die Reproduzierbarkeit, indem wir Resultate einer anderen Projektgruppe nachvollziehen und einordnen.

## 2 Hintergrund

**Dijkstra / ECMP:** Kürzeste-Wege-Routing minimiert die Summe der Kantengewichte entlang eines Pfads. Existieren mehrere Pfade mit identischer Gesamtkostenlänge, aktiviert *Equal-Cost Multi-Path* (ECMP) eine gleichmäßige Aufteilung des Demand-Volumens auf diese Pfade. Formal: Für Gewichte $w$ und Demand $(s, t, d)$ bezeichne $a_{ij}(s, t; w) \in [0, 1]$ den ECMP-Anteil auf Link $(i, j)$; die linkweise Last ist $\ell_{ij}(w) = \sum_{(s,t,d)\in\mathcal{D}} d\, a_{ij}(s, t; w)$.

**inverse_capacity (IC):** Linkkosten werden proportional zum Kehrwert der Kapazität gesetzt, skaliert über die maximale Kapazität:

$$\text{base\_w}_{ij} = \frac{\max_{e\in E} c_e}{c_{ij}},$$

wodurch kapazitätsstarke Links in der Kürzeste-Wege-Metrik bevorzugt werden. IC dient bei uns sowohl als Baseline (ohne weitere Anpassung) als auch als Startmetrik für adaptive Verfahren.

**TL_Inverse:** Aufbauend auf IC werden die Gewichte *iterativ* relativ zur Basis an die beobachtete, normierte Auslastung $u_{ij} = \ell_{ij}/c_{ij}$ angepasst:

$$w_{ij}^{(k+1)} = \text{base\_w}_{ij}\left(1 + \alpha\, u_{ij}^{(k)}\right), \quad \alpha > 0.$$

Ablauf: Routen → Last messen → Gewichte anpassen (mehrere Iterationen, dann Fixierung). Die Skalierung *relativ zur Basis* begrenzt Übersteuerung und stabilisiert das Verfahren.

**Demand-First-Waypoints Prinzip:** Greedy-Heuristik mit *höchstens einem* Waypoint (Segment) pro Demand: Demands werden *absteigend nach Volumen* bearbeitet; für jeden Demand $(s, t, d)$ wird ein Kandidat $w \in V\backslash\{s, t\}$ gesucht, der die *maximale Linkauslastung (MLU)*

$$\text{MLU}(w) = \max_{(i,j)\in E} \frac{\ell_{ij}(w)}{c_{ij}}$$

möglichst senkt (Routing über $s \to w \to t$ mit Dijkstra+ECMP). Nur wenn die MLU *strikt* sinkt, wird $w$ akzeptiert; anschließend wird die Lastkarte aktualisiert.

**DFW-Variante (DFW):** Zwei gezielte Modifikationen: (i) *Kapazitätsbewusste Startmetrik* via IC (siehe oben) statt uniformer Kosten, damit die Waypoint-Suche auf Pfade mit mehr „Headroom" fokussiert. (ii) *Robuster ECMP-Split*: Sonderfall „keine gültigen Nachfolger" führt nicht zu Division durch Null; Pfadanteile werden deterministisch auf 0 gesetzt. Greedy-Schema, MLU-Kriterium und ECMP bleiben unverändert.

**Topologien.** *Abilene* (klein, relativ homogen; geringe Pfaddiversität), *Géant* (mittelgroß, gut vernetzt; mehrere konkurrenzfähige Alternativrouten), *Germany50* (größer, heterogener; höhere Sensitivität für Parameter und Heuristiken).

## 3 Projekt 1 — TE_SR_WAN_simulation

### 3.1 Algorithmus TL_Inverse

Wir modellieren das Netzwerk als gerichteten Graphen $G = (V, E)$ mit Linkkapazitäten $c_{ij} > 0$ für $(i, j) \in E$ und einer Demandmenge $\mathcal{D} = \{(s, t, d)\}$, wobei $d$ das zu routende Verkehrsvolumen von Quelle $s$ zu Senke $t$ bezeichnet. Ausgehend von der klassischen *inverse_capacity*-Heuristik werden zunächst *Basisgewichte* definiert, die Links mit hoher Kapazität begünstigen:

$$\text{base\_w}_{ij} = \frac{\max_{e\in E} c_e}{c_{ij}} \quad \text{für alle } (i, j) \in E. \tag{1}$$

Betreuung / Lehrstuhl: Prof. Dr. Dr. Klaus-Tycho Förster  Datum: 22.09.2025.

Diese Basiskosten dienen als Ausgangspunkt eines iterativen, lastsensitiven Anpassungsverfahrens. Wir initialisieren $w_{ij}^{(0)} = \text{base\_w}_{ij}$ und berechnen für jede Iteration $k = 0, \ldots, K-1$ zunächst die kürzesten Wege relativ zu den aktuellen Gewichten $w^{(k)}$. Bei Mehrdeutigkeit (gleich lange Pfade) verwenden wir ECMP und teilen das Volumen $d$ eines Demands gleichmäßig auf alle gleichwertigen Pfade auf. Bezeichne $a_{ij}^{(k)}(s,t) \in [0,1]$ den Anteil des Demands $(s,t,d)$, der in Iteration $k$ über Link $(i,j)$ fließt (ECMP-Anteilssplitting). Daraus ergibt sich die linkweise *akkumulierte Last*

$$\ell_{ij}^{(k)} \;=\; \sum_{(s,t,d)\in\mathcal{D}} d \cdot a_{ij}^{(k)}(s,t), \tag{2}$$

und die *normierte Auslastung*

$$u_{ij}^{(k)} \;=\; \frac{\ell_{ij}^{(k)}}{c_{ij}}\,. \tag{3}$$

Im Anschluss werden die Linkgewichte proportional zur beobachteten Auslastung angepasst. Charakteristisch für *TL_Inverse* ist dabei, dass die Aktualisierung stets *relativ zur Basisgewichtung* (1) erfolgt und somit keine ungebundene Gewichtsausweitung über Iterationen hinweg stattfinden kann:

$$w_{ij}^{(k+1)} \;=\; \text{base\_w}_{ij}\,\bigl(1 + \alpha\, u_{ij}^{(k)}\bigr), \quad \alpha > 0. \tag{4}$$

Der Parameter $\alpha$ steuert die Reaktionsstärke des Verfahrens: Kleine Werte führen zu konservativen, große Werte zu aggressiveren Gewichtsanhebungen auf ausgelasteten Links. Nach $K$ Iterationen (in unseren Experimenten typischerweise $K = 3$) werden die finalen Gewichte $w_{ij}^{\star} = w_{ij}^{(K)}$ fixiert, und die endgültige Routenwahl erfolgt wieder per ECMP über kürzeste Pfade. Durch die Skalierung relativ zur Basis (1) bleibt das Verfahren numerisch stabil, während (4) gezielt jene Links penalisiert, deren normierte Auslastung (3) in der vorherigen Iteration erhöht war. Auf diese Weise können Hotspots schrittweise entschärft und Verkehrslasten auf weniger beanspruchte Teile der Topologie umverteilt werden.

## 3.2 Evaluationsdesign (TL_Inverse)

Zur Beurteilung von *TL_Inverse* vergleichen wir das Verfahren gegen die klassische *inverse_capacity* -Baseline auf drei etablierten WAN-Topologien: **Abilene** (klein, relativ homogen), **Géant** (mittelgroß, gut vernetzt) und **Germany50** (größer, heterogener). Als Verkehrsgrundlage dienen realistisch skalierte Demand-Matrizen; geroutet wird stets auf einem gerichteten Graphen $G = (V, E)$ mit Kapazitäten $c_{ij} > 0$ für $(i,j) \in E$. Für *TL_Inverse* werden die Gewichte aus der in Abschnitt 3.1 beschriebenen Iterationsprozedur gewonnen (typisch $K = 3$ Iterationen) und für die finale Bewertung fixiert. Die Baseline *inverse_capacity* entsteht aus den Basisgewichten $\text{base\_w}_{ij} = \max_{e\in E} c_e / c_{ij}$ ohne weitere Anpassung.

Die Routenwahl erfolgt deterministisch mittels kürzester Wege relativ zu den finalen Gewichten und *Equal-Cost Multi-Path* (ECMP): Existieren mehrere gleichlange Pfade, wird das Demand-Volumen $d$ eines Paares $(s,t,d)$ gleichmäßig auf diese Pfade verteilt. Bezeichne $a_{ij}(s,t;w) \in [0,1]$ den ECMP-Anteil des Demands $(s,t,d)$, der unter Gewichten $w$ über Link $(i,j)$ fließt. Dann ergibt sich die linkweise Last zu

$$\ell_{ij}(w) \;=\; \sum_{(s,t,d)\in\mathcal{D}} d \cdot a_{ij}(s,t;w)\,,$$

die *normierte Auslastung* zu $u_{ij}(w) = \ell_{ij}(w)/c_{ij}$ und als zentrale Zielgröße die *maximale Linkauslastung* (MLU)

$$\text{MLU}(w) \;=\; \max_{(i,j)\in E} u_{ij}(w)\,.$$

Für *TL_Inverse* variieren wir die Reaktionsstärke über $\alpha \in \{0.3,\ 0.5,\ 0.7\}$; die übrigen Parameter (Topologie, Kapazitäten, Demands, Routingmodell) bleiben zwischen den Algorithmen identisch. Ergebnisse berichten wir als Abbildungen (Balken/Boxplots) je Topologie und $\alpha$; auf Tabellen verzichten wir bewusst, da der Fokus auf der qualitativen Einordnung der MLU-Unterschiede liegt. Hinweise auf weitere Heuristiken (z. B. *DemandFirstWaypoints/GreedyWaypoints*) erscheinen in einzelnen Abbildungen nur als Kontext und sind nicht Teil der primären Bewertung.

## 3.3 Ergebnisse (TL_Inverse)

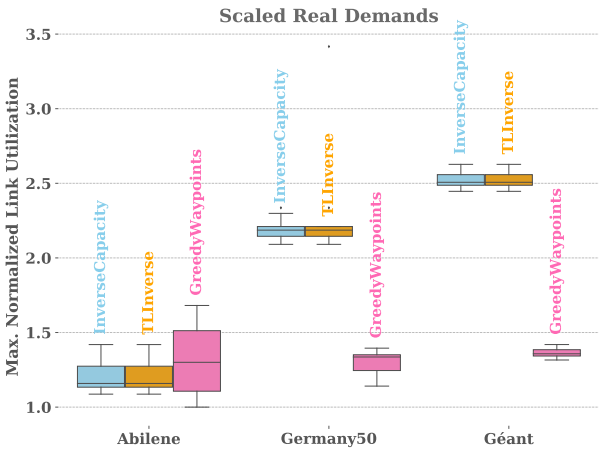


**Abbildung 1: Projekt 1 - TL_Inverse,** $\alpha = 0{,}3$

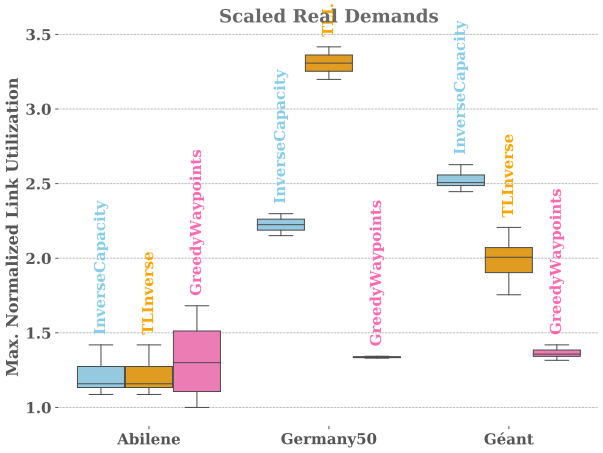


**Abbildung 2: Projekt 1 – TL_Inverse,** $\alpha = 0{,}5$

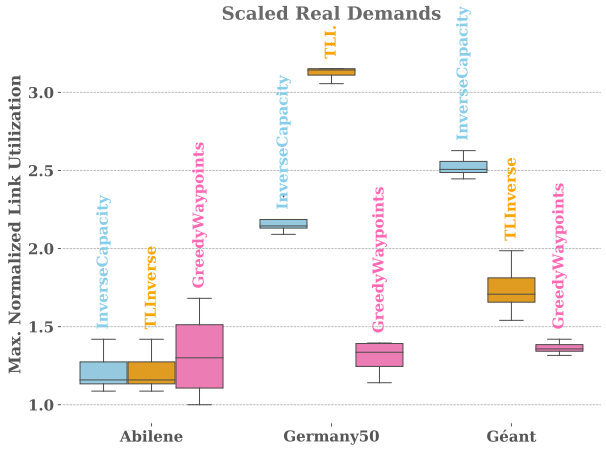


**Abbildung 3: Projekt 1 – TL_Inverse,** $\alpha = 0{,}7$

Über alle Topologien zeigt sich: Bei $\alpha = 0.3$ (vgl. Abb. 1) sind die Resultate praktisch identisch zur *inverse_capacity*-Baseline—der Strafterm ist zu schwach, um Pfade spürbar zu verlagern. Mit $\alpha = 0.5$ (vgl. Abb. 2) erzielt *TL_Inverse* einen guten Kompromiss: In **Géant** sinkt die MLU sichtbar, während in **Abilene** wegen geringer Pfadvielfalt kaum reagiert. Nur in **Germany50** kommt es zu einer erhöhten MLU, da durch die heterogene Struktur von **Germany50** neue Hotspots provoziert werden: Entlastete Kanten werden durch neu entstehende Engpässe auf Ausweichpfaden „ersetzt", sodass die MLU nicht mehr konsistent fällt. Erhöht man auf $\alpha = 0.7$(vgl. Abb. 3), sehen wir ähnliche Ergebnisse wie für $\alpha = 0.5$. In Summe empfiehlt sich $\alpha \approx 0.5$ zu wählen, um effektive Ergebnisse für die verschiedenen Topologien zu erzielen. Der Einsatz von *TL_Inverse* sollte jedoch je nach Topologie gut überdacht sein, da er zu deutlich erhöhter MLU führen kann.

## 3.4 Diskussion (TL_Inverse)

Die beobachteten Effekte von *TL_Inverse* lassen sich konsistent aus dem Wirkprinzip der Gewichtsaktualisierung in (4) und der resultierenden Routenwahl erklären. Indem die Linkkosten proportional zur *normierten* Auslastung $u_{ij}$ steigen, werden in der jeweils nächsten Iteration genau jene Kanten unattraktiver, die zuvor Engpässe gebildet haben. Der zentrale Stabilisator ist dabei die Skalierung *relativ zur Basis* (*vgl.* (1)): Da $w_{ij}^{(k+1)}$ stets aus base_w$_{ij}$ abgeleitet wird, kann die Metrik nicht ungebunden „davonlaufen", sondern bleibt numerisch gebändigt. In Verbindung mit ECMP führt dies zu einer schrittweisen, aber kontrollierten Pfadverlagerung weg von Hotspots hin zu weniger belasteten Teilen der Topologie.

Gleichzeitig erklärt dieses Zusammenspiel die $\alpha$-Sensitivität: Für kleine Werte ($\alpha = 0{,}3$) ist die Modulation in (4) so schwach, dass die induzierten Kostenunterschiede die Kürzeste-Pfade-Struktur praktisch nicht verändern; das Ergebnis fällt identisch zur *inverse_capacity*-Baseline aus. Bei moderaten Werten ($\alpha = 0{,}5$) entsteht der intendierte Trade-off: Überlastete Links werden hinreichend penalisiert, ohne dass alternative Pfade übermäßig verteuert oder neue Ungleichgewichte erzeugt werden — insbesondere in gut vernetzten Graphen wie **Géant** senkt dies die MLU spürbar. Für hohe Werte ($\alpha = 0{,}7$) kann die Anpassung hingegen *übersteuern*: ECMP-Ties und diskrete Pfadwechsel führen dann zu nichtlinearen Sprüngen in der Lastverteilung, sodass Entlastung auf einer Kante durch neu entstehende Hotspots auf Ausweichrouten „bezahlt" wird (**Germany50**). Das Verfahren bleibt zwar stabil (durch die Basis-Skalierung), optimiert die MLU aber nicht mehr monoton.

Die Topologie spielt dabei eine doppelte Rolle. In kleinen, relativ homogenen Netzen mit geringer Pfadvielfalt (**Abilene**) fehlen Alternativen, sodass selbst sinnvolle Gewichtsmodulationen kaum Wirkung entfalten. In mittelgroßen, gut vernetzten Netzen (**Géant**) existieren mehrere konkurrenzfähige, nur leicht längere Routen; hier entfaltet *TL_Inverse* den größten Nutzen. In größeren und heterogeneren Netzen (**Germany50**) steigen die Risiken von „Lastverschiebungen": Kapazitätsunterschiede und längere alternative Wege können dazu führen, dass eine lokale Penalität global unerwünschte Nebenwirkungen erzeugt, wenn $\alpha$ zu groß gewählt wird.

Als methodische Grenzen bleiben drei Punkte: Erstens ist der Strafterm in (4) linear in $u_{ij}$; nichtlineare Varianten (z. B. saturierende oder stückweise lineare Formen) könnten Überreaktionen dämpfen, ohne den Reaktionsgrad bei moderater Auslastung zu verlieren. Zweitens optimiert *TL_Inverse* die MLU *indirekt* über Gewichtssteuerung und ist damit empfindlich gegenüber ECMP-Tie-Breaking und diskreten Pfadwechseln; eine zusätzliche Regularisierung (z. B. Pfadlängen- oder Schwankungsterm) könnte hier Robustheit schaffen. Drittens betrachten wir ausschließlich die MLU als Zielgröße; in Praxis-Setups sind ergänzende Kriterien (Fairness, Pfadlänge, Latenzvariabilität) relevant und können Zielkonflikte mit der MLU aufdecken. Insgesamt legt die Analyse nahe, $\alpha$ konservativ zu wählen (typisch um $\alpha \approx 0{,}5$) und die Topologieeigenschaften (Pfaddiversität, Kapazitätsheterogenität)bei der Parametrisierung explizit zu berücksichtigen.

## 3.5 Algorithmus demand_first_waypoints

Aufbauend auf der bekannten Demand-First-Waypoints-Heuristik verfolgen wir dasselbe Greedy-Prinzip, lassen jedoch zwei gezielte Modifikationen einfließen. Erstens verwenden wir statt uniformer Startkosten die kapazitätsbewussten Basisgewichte aus (1), d. h.

$$w_{ij}^{(0)} = \text{base\_w}_{ij} = \max_{e \in E} c_e / c_{ij}$$

Damit bevorzugt die Kürzeste-Wege-Metrik bereits vor der Waypoint-Suche Links mit höherem Headroom. Zweitens wird der ECMP-Split in seltenen Randfällen numerisch robust gemacht (kein Divide-by-Zero, wohldefinierte Null-Anteile, falls temporär keine gültigen Nachfolger vorliegen).

Der eigentliche Greedy-Schritt bleibt unverändert: Die Demands $(s, t, d) \in \mathcal{D}$ werden absteigend nach Volumen verarbeitet, und es wird höchstens ein Waypoint $w \in V \setminus \{s, t\}$ zugelassen. Für jeden Kandidaten wird das Routing über $s \to w \to t$ (mit Dijkstra+ECMP unter $w^{(0)}$) simuliert und die resultierende maximale Linkauslastung (MLU, vgl. Abschnitt 3.2) bewertet. Wir wählen

$$w^\star = \arg \min_{w \in V \setminus \{s,t\}} \text{MLU}\big(\text{Routen}(s \to w \to t;\, w^{(0)})\big) \qquad (5)$$

und übernehmen $w^\star$ nur, wenn die MLU *strict* sinkt; anschließend wird die Lastkarte aktualisiert und der nächste Demand bearbeitet. Die Kombination aus kapazitätsbewusster Initialisierung und konservativem „ein Segment pro Demand"-Schema zielt darauf ab,

Hotspots effektiv zu entschärfen, ohne die Pfadstruktur aggressiv zu destabilisieren.

## 3.6 Evaluationsdesign (DFW)

Die Bewertung unserer Anpassung von `demand_first_waypoints` erfolgt auf denselben Topologien wie zuvor (Abilene, Géant, Germany50) mit identischen, realistisch skalierten Demand-Sätzen; Graph- und Kapazitätsmodell entsprechen Abschnitt 3.1. Als Referenz dient die bekannte Demand-First-Waypoints-Baseline, gegen die wir ausschließlich die beiden Änderungen aus Abschnitt 3.5 testen (inverse_capacity-Startgewichte statt uniformer Kosten; robuster ECMP-Split). Das Greedy-Schema bleibt ansonsten unverändert (ein Waypoint pro Demand, nur wenn die MLU sinkt).

Die Routenwahl geschieht deterministisch über Dijkstra+ECMP: Für jeden Demand werden nach der Greedy-Entscheidung die kürzesten Wege relativ zu den *fixen* Startgewichten verwendet; die Lastberechnung und die Zielgröße *MLU* folgen der Definition aus Abschnitt 3.2. Da das Verfahren ohne iterative Gewichtsanpassung arbeitet, genügt ein einzelner Greedy-Durchlauf über alle Demands („demands first“).

Wir berichten die Ergebnisse als Boxplots der MLU je Topologie (ohne Tabellen); die vertikale Achse ist – analog zu den TL_Inverse-Plots – bewusst gezoomt, um kleine, aber systematische Unterschiede sichtbar zu machen.

## 3.7 Ergebnisse (DFW)

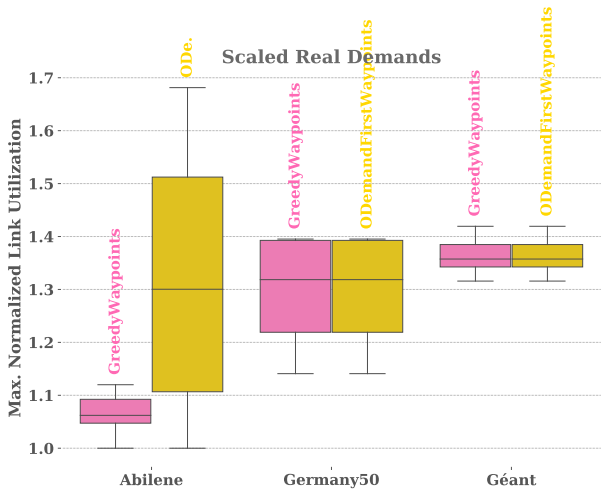


**Abbildung 4: Projekt 1 – DFW**

Über alle Topologien zeigt sich ein konsistentes Muster (vgl. Abb. 4; die y-Achse ist gezoomt): In **Abilene** fallen die Unterschiede gering aus, was der niedrigen Pfaddiversität entspricht. In **Géant** profitiert unsere Variante von der kapazitätsbewussten Startmetrik und erzielt häufig eine niedrigere MLU als die Baseline. Für **Germany50** hängt der Effekt stärker von der Demandlage ab; Greedy-Entscheidungen können vereinzelt Last verlagern und damit neue Hotspots auf Ausweichpfaden erzeugen, sodass die Vorteile nicht in allen Seeds konsistent sind. Insgesamt liefert *GreedyWaypoints* leichte Mediangewinne bei moderater Pfadvielfalt, während in sehr kleinen oder sehr heterogenen Netzen die Robustheit der Verbesserungen begrenzt bleibt.

## 3.8 Diskussion (DFW)

Die Befunde lassen sich aus dem Zusammenspiel von kapazitätsbewusster Startmetrik und Greedy-Auswahl („ein Segment pro Demand“) erklären. Durch die Initialisierung $w_{ij}^{(0)} = \text{base\_w}_{ij}$ aus (1) verschiebt sich die Kürzeste-Wege-Metrik bereits vor der Waypoint-Suche in Richtung kapazitätsstarker Links; der erste akzeptierte Waypoint wirkt daher häufig *entlastend*, ohne schwache Kanten unnötig zu beanspruchen. Diese Vorstrukturierung entfaltet ihren größten Nutzen dort, wo mehrere konkurrenzfähige Alternativpfade existieren (Géant): Lokale Umlenkungen führen dann zu einer echten Abflachung der Spitzenlast (MLU), ohne die Pfadlängen signifikant zu erhöhen.

Grenzen zeigen sich an beiden Enden des Spektrums: In Netzen mit geringer Pfadvielfalt (Abilene) fehlen hinreichend gute Alternativen, sodass die Waypoint-Suche nur selten eine spürbare Verbesserung gegenüber der Baseline erzielt. In größeren, heterogeneren Topologien (Germany50) kann die Greedy-Entscheidung Last *verlagern* statt sie zu neutralisieren, insbesondere wenn ECMP-Ties diskrete Sprünge in der Routenwahl auslösen. Das erklärt die beobachtete Streuung: Einzelne Seeds profitieren deutlich, andere erzeugen neue Hotspots auf Ausweichpfaden.

Methodisch ist die *Single-Waypoint-Beschränkung* pro Demand der zentrale Trade-off zwischen Einfachheit und Optimalkraft. Zwei natürliche Erweiterungen bieten sich an: (i) ein zweites Segment, das nur bei weiterer MLU-Reduktion aktiviert wird (stufenweise Greedy-Suche), und (ii) eine hybride Kopplung mit gewichtsbasierter Penalität (z. B. TL_Inverse) zur Dämpfung von Überreaktionen bei ECMP-Ties. Zudem ließe sich die Auswahl regelbasiert regularisieren (z. B. Mindestentlastungsschwelle, sanfte Strafkosten für Pfadlängen-Inflation), um Varianz zu reduzieren, ohne den medianen Gewinn in pfaddiversen Netzen zu verlieren.

# 4 Projekt 2 — TE_SR_experiments

## 4.1 Zielsetzung & Topologie (TL_Inverse)

Dieses Kapitel überträgt die in Projekt 1 untersuchten Ideen in eine nanonet-basierte Laufumgebung und betrachtet zwei Vergleichsstrecken: *TL_Inverse vs. New_Weights* sowie *DFW vs. Weights_2*. Für **TL_Inverse vs. New_Weights** nutzen wir einen gerichteten Graphen mit $|V| = 6$ Knoten ($V = \{0, 1, 2, 3, 4, 5\}$) und paarweise gerichteten Kanten. Die Kapazitäten sind heterogen: die innere Verbindung $0 \leftrightarrow 3$ ist mit $c_{03} = c_{30} = 5$ der *Bottleneck*, alle übrigen Links tragen $c_{ij} = 10$ (Außenring als Ausweichstruktur). Die Demandmenge ist

$$\mathcal{D} = \{(0, 4, 10),\ (0, 5, 5),\ (2, 4, 5),\ (1, 5, 5)\},$$

wodurch mehrere Alternativrouten mit geringen Kostenunterschieden, aber potenziell unterschiedlicher Auslastung unter ECMP entstehen.

## 4.2 Implementierung (TL_Inverse)

Die Pipeline ist reproduzierbar und identisch zur in Projekt 1 verwendeten Toolchain: Aus einer JSON-Beschreibung (Knoten, Kanten mit Kapazität, Demands) erzeugt traffic_engineering_json2nanonet_v2_tl.py ein ausführbares `.topo.py`. Danach generiert build.py daraus das `.topo.sh`, das Linux-Namespaces, Veth-Paare und Routingtabellen aufsetzt und die Messläufe startet. Die Routen werden in dijkstra_computed() materialisiert (Dijkstra/ECMP, optional SR-Segmente). Pro Demand starten parallele `nuttcp`-Ströme (`NSTREAMS=32`) über `TIME=300` s; eine Startverzögerung von ca. 2 min stellt die Initialisierung sicher. ECMP ist via Hash-Policy aktiv. Um vorzeitige Terminierungen auf leistungsschwacher VM zu vermeiden, wurde die Aufräumphase konservativ verzögert (z. B. `sleep(self.TIME + 60)`), sodass `throughput.py -s` die Auswertung zuverlässig abschließt.

*Abbildung der Verfahren:* **New_Weights** setzt *statisch* erhöhte Kosten auf dem Innenlink $0 \leftrightarrow 3$ (doppelte Linkkosten) und erzwingt per Segment Routing (SR) gewünschte Wege um den Engpass. **TL_Inverse** startet mit gleichmäßigen Gewichten, führt eine vorgelagerte Lastschätzung aus und passt die Linkkosten proportional zur normierten Auslastung an (vgl. (4)); die finalen Pfade werden als Multipath-Konfiguration mit Anteilsgewichten in die Topologie geschrieben. Alle übrigen Parameter (Topologie, Kapazitäten, Demands, Messdauer) sind zwischen den Varianten identisch.

## 4.3 Baselines & Metrik (TL_Inverse)

Verglichen werden **New_Weights** (deterministische SR-Policy mit manuell verteuertem Bottleneck) und **TL_Inverse** (datengetriebene, lastadaptive Gewichtssteuerung). Die Routenwahl erfolgt in beiden Fällen über Dijkstra mit ECMP. Zielgröße ist die *maximale Linkauslastung (MLU)*; Definition und Lastberechnung entsprechen Abschnitt 3.2. Weitere Kriterien (Fairness, Pfadlänge, Jitter) optimieren wir hier bewusst nicht, um den Einfluss der Routenwahl auf Hotspot-Bildung isoliert zu betrachten.

## 4.4 Ergebnisse & Einordnung (TL_Inverse)

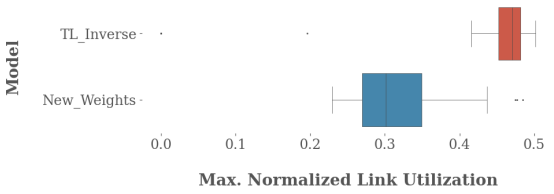


**Abbildung 5: Projekt 2 — TL_Inverse.**

Die Ergebnisse in Figure 5 zeigen einen *leicht* niedrigeren Median und eine geringere Streuung für *New_Weights* gegenüber *TL_Inverse*. Die gezoomte y-Achse verdeutlicht, dass die absoluten Differenzen klein sind; beide Verfahren liegen in einem engen MLU-Band. Der Vorteil der Baseline erklärt sich durch die Kombination aus *statisch verteuertem Innenlink* und *SR-Policy*, die die Umgehung des Bottlenecks deterministisch erzwingt. *TL_Inverse* verteilt Last adaptiv, kann bei geringer Pfaddiversität jedoch neue Belastungsschwerpunkte auf Ausweichpfaden erzeugen und den Median nicht weiter senken. Insgesamt deuten die Resultate darauf hin, dass manuell getunte SR-Policies in kleinen, gut kontrollierbaren Topologien robuste leichte Vorteile liefern, während adaptive Verfahren ihren Mehrwert eher in dynamischeren oder variablen Situationen entfalten.

## 4.5 Zielsetzung & Topologie (DFW vs. Weights_2)

Für die zweite Vergleichsstrecke konstruieren wir eine kompakte, SR-fähige Topologie mit **fünf Knoten** (0–4) und zwei Kapazitätsklassen: **80 Mbit/s** auf den Außenkanten ($0 \leftrightarrow 1$, $0 \leftrightarrow 3$, $1 \leftrightarrow 4$, $3 \leftrightarrow 4$) sowie **40 Mbit/s** auf allen Kanten über *Knoten 2* ($0 \leftrightarrow 2$, $1 \leftrightarrow 2$, $2 \leftrightarrow 3$, $2 \leftrightarrow 4$). Die Demands sind symmetrisch und je **10 Mbit/s**: $0 \rightarrow 4$, $4 \rightarrow 0$, $1 \rightarrow 3$, $3 \rightarrow 1$. Knoten 2 bildet damit einen Innen-Hub mit geringerer Kapazität; über den Außenring stehen belastbare Alternativpfade bereit.

## 4.6 Implementierung (DFW vs. Weights_2)

Die Artefakte werden analog erzeugt (JSON → `.topo.py` → `.topo.sh`); Mess- und Systemparameter entsprechen Strecke A (`TIME=300` s, `NSTREAMS=32`, Startoffset ≈ 2 min, ECMP aktiv; konservative Terminierung zur Vermeidung abgebrochener Läufe). In **Weights_2** sind die *Linkkosten uniform* (z. B. 1000), SR-Waypoints werden *nicht* gesetzt; die Routen entstehen rein durch Dijkstra/ECMP. In **DFW** werden die *Linkkosten kapazitätsbewusst* gesetzt (z. B. 2000 auf 40 Mbit/s, 1000 auf 80 Mbit/s); zusätzlich erzwingen wir pro Demand *einen SR-Waypoint* (z. B. $0 \rightarrow 4$ via 1, $4 \rightarrow 0$ via 3, $1 \rightarrow 3$ via 4, $3 \rightarrow 1$ via 0) mittels `encap seg6 ... segs {w}`. Damit spiegelt Strecke B den methodischen Unterschied der Idee wider: kapazitätsbewusste Vorstrukturierung plus wohldosiertes, deterministisches Segment im Vergleich zu einer uniformen Weights-Baseline.

## 4.7 Baselines & Metrik (DFW vs. Weights_2)

Verglichen werden **DFW** (GreedyWaypoints, kapazitätsbewusste Kosten, fester Single-Waypoint pro Demand) und **Weights_2** (uniforme Kosten, keine Waypoints). Als Zielgröße verwenden wir die MLU aus Abschnitt 3.2; alle übrigen Parameter (Demands, Seeds, Messdauer) sind zwischen den Varianten identisch. Ergebnisse berichten wir als Boxplots, um Median und Streuung gegenüberzustellen.

## 4.8 Ergebnisse & Einordnung (DFW vs. Weights_2)

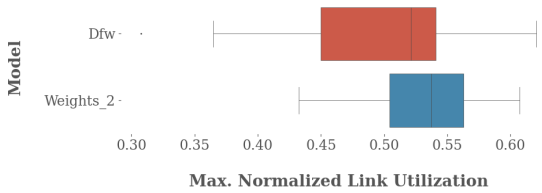


**Abbildung 6: Projekt 2 — DFW.**

Die Median-MLU beider Varianten liegt nahe beieinander; *DFW* ist im Median leicht im Vorteil, zeigt jedoch eine größere Streuung. *Weights_2* liefert eine engere Box (stabiler), liegt dafür minimal höher. Die absoluten Unterschiede sind klein (gezoomte Achse). Interpretatorisch passt das Bild zur Konstruktion: Die Kombination aus kapazitätsbewusster Metrik und erzwungenen SR-Segmenten umgeht den Innen-Bottleneck zuverlässig, erhöht aber die Varianz abhängig von ECMP-Aufteilung und Segmentwahl. Die uniforme Baseline ist robuster, jedoch nicht immer median-optimal. Zusammen mit Strecke A ergibt sich damit ein konsistentes Gesamtbild: Auf kleinen, gezielt designten Topologien können deterministische SR-Policies einen stabilen Vorteil liefern, während adaptive und/oder kapazitätsbewusste Strategien leichte Mediangewinne erzielen, deren Robustheit mit der Pfaddiversität und der konkreten Demandlage variiert.

## 5 Limitierungen & Validität

Die Aussagekraft der Ergebnisse wird durch mehrere Validitätsaspekte begrenzt, die wir im Folgenden transparent machen und, wo möglich, mit getroffenen Gegenmaßnahmen verknüpfen.

*Interne Validität.* Die Implementierungen orientieren sich strikt an den in Kapitel 3 eingeführten Definitionen: *TL_Inverse* aktualisiert Gewichte relativ zur Basis ((1)) gemäß (4); die Auswertung erfolgt nach fixierten $K$ Iterationen. Für *DFW* (unsere Anpassung von `demand_first_waypoints`) verwenden wir kapazitätsbewusste Startgewichte ($w_{ij}^{(0)}$ = base_w$_{ij}$) sowie einen robusten ECMP-Split, der Divide-by-Zero vermeidet. Gleichwohl verbleiben Risiken: Wir haben keine *formale Konvergenz-/Laufzeitanalyse* für *TL_Inverse* durchgeführt (nur $K$=3 in den Experimenten), und bei *DFW* erzwingt die Single-Waypoint-Beschränkung eine Heuristik, die lokale Optima begünstigen kann. In *Projekt 2* traten auf der genutzten VM vorzeitige Terminierungen auf; zur Absicherung wurde die Aufräumphase konservativ verzögert (`sleep(self.TIME + 60)`), damit alle Läufe vollständig in `throughput.py -s` einfließen. Diese Maßnahme reduziert Messabbrüche, eliminiert jedoch nicht mögliche *System-Noise* (Scheduling, CPU-Steal) der VM. Sämtliche Läufe nutzen identische Parameter (z. B. `NSTREAMS=32`, `TIME=300 s`, einheitliche ECMP-Hash-Policy), um Implementationsartefakte von algorithmischen Effekten zu trennen.

*Konstruktvalidität.* Wir fokussieren bewusst die *maximale Linkauslastung (MLU)* als Zielgröße. Diese konservative Kenngröße erfasst Hotspots unmittelbar, blendet jedoch andere für den Betrieb relevante Aspekte (Pfadlänge, Latenz/Jitter, Fairness) aus. Insbesondere bei *TL_Inverse* kann eine MLU-Verbesserung mit leichten Pfadlängen-Zunahmen einhergehen; bei *DFW* kann die lokale Umlenkung Pfade selektiv verlängern. Auch die ECMP-Semantik (gleichmäßige Aufteilung bei gleichen Kosten) ist Teil des Konstrukts: Tie-Breaking und diskrete Pfadwechsel können Lastsprünge verursachen, die sich in der MLU widerspiegeln, ohne dass die Gesamtnetzqualität zwingend schlechter wäre. Unsere qualitative Interpretation macht diese Grenzen transparent, ersetzt aber keine Multi-Ziel-Optimierung.

*Externe Validität.* Die drei WAN-Topologien (Abilene, Géant, Germany50) decken ein Spektrum aus Größe, Konnektivität und Heterogenität ab, bleiben aber Repräsentanten; die Übertragbarkeit auf größere ISP-Backbones, Traffic-Diurnalität oder hochvariable Demand-Matrizen ist nicht automatisch gegeben. Die in *Projekt 2* entworfenen Topologien sind *absichtlich klein* und SR-fähig, um Mechanismen isoliert sichtbar zu machen; daraus folgen klare, aber kontextgebundene Aussagen. Zudem setzt *DFW* in *Projekt 2* feste Waypoints je Demand ein—eine starke Annahme, die in produktiven Netzen durch Policy-/SR-Constraints, Tailoring und Operationspraxis limitiert sein kann.

*Statistische Schlussfolgerungsvalidität.* Wir berichten Ergebnisse primär über Boxplots (Median/Streuung) und verzichten auf formale Signifikanztests oder umfangreiche Seed-Studien. Die y-Achsen wurden *gezoomt*, um kleine, aber systematische Unterschiede sichtbar zu machen; dadurch wirken Abstände optisch größer als in absoluten Werten. Insbesondere in *Projekt 2* sind die Differenzen zwischen Baseline und Variante *klein*; unsere Einordnung bleibt daher *qualitativ* und betont Trends statt harter Schwellen.

*Maßnahmen zur Absicherung.* Wir halten die Vergleichsbedingungen zwischen Baselines und Varianten konstant (Topologie, Kapazitäten, Demands, `NSTREAMS`, Messdauer, ECMP-Policy). Für *TL_Inverse* variieren wir $\alpha \in \{0.3, 0.5, 0.7\}$, um Sensitivität sichtbar zu machen; für *DFW* beschränken wir Änderungen strikt auf Startmetrik und ECMP-Robustheit. Die erfolgreiche *Replikation* fremder Ergebnisse in beiden Projekten stützt die korrekte Umsetzung der Toolchains und erhöht die Nachvollziehbarkeit. Alle verwendeten Artefakte (JSON/`.topo.py`/`.topo.sh`, Plots) sind versioniert abgelegt und im Quellenanhang referenziert.

*Implikationen.* Die Resultate sind *valide im Sinne des gewählten Konstrukts* (MLU unter Dijkstra+ECMP/SR) und der betrachteten Topologien. Für breitere Generalisierungen empfehlen sich (i) zusätzliche Zielfunktionen (Pfadlänge, Fairness, Jitter), (ii) Sensitivitätsstudien über Demands/Seeds und (iii) größere, realitätsnahe Topologien. Methodisch bieten sich als Erweiterungen nichtlineare Strafterme bzw. adaptive $\alpha$ (für *TL_Inverse*) sowie Multi-Segment-Greedy oder Hybride (Waypoint + Gewichtsmodulation) für *DFW* an, um Varianz zu dämpfen, ohne Median-Vorteile zu verlieren.

## 6 Fazit

**Fazit.** Die beiden betrachteten Ansätze adressieren die Reduktion der maximalen Linkauslastung (MLU) komplementär: *TL_Inverse* erweitert die inverse_capacity-Metrik um eine auslastungsadaptive Komponente und zeigt in mittelgroßen, gut vernetzten Netzen (Géant) konsistente Vorteile, während in sehr kleinen, homogenen Topologien (Abilene) die Pfaddiversität zu gering ist, um spürbare Effekte zu entfalten. In heterogeneren Netzen (Germany50) ist die Wahl von $\alpha$ maßgeblich; moderat gewählte Werte (ca. $\alpha \approx 0{,}5$) balancieren Entlastung und Stabilität am zuverlässigsten. Im nanonet-Setting (Projekt 2) liegt *New_Weights* leicht vorn, da die Kombination aus manuell verteuertem Bottleneck und deterministischer SR-Policy die Umgehung des Engpasses erzwingt und auf der bewusst kleinen Topologie sehr gut zur Demandlage passt; die absoluten Unterschiede bleiben gleichwohl klein. Unsere angepasste Waypoint-Heuristik *DFW* profitiert von der kapazitätsbewussten Startmetrik: In pfaddiversen Szenarien liefert sie leichte Mediangewinne (Projekt 1/Géant), während sie in Projekt 2 gegenüber der uniformen Weights-Baseline einen minimalen Medianvorteil bei

höherer Streuung zeigt. Insgesamt ergibt sich ein konsistentes Bild: deterministische, manuell getunte SR-Policies liefern in kleinen, gut kontrollierbaren Topologien robuste, leicht bessere Mediane; adaptive bzw. kapazitäts bewusste Strategien reagieren datengetrieben und können in geeigneten Topologien MLU-Vorteile erzielen, allerdings mit stärkerer Varianz. Die erfolgreichen Replikationen stützen die korrekte Umsetzung und erhöhen die Nachvollziehbarkeit der Befunde.

# 7 Replikation (fremde Ergebnisse)

## 7.1 Projekt 1 — TE_SR_WAN_simulation

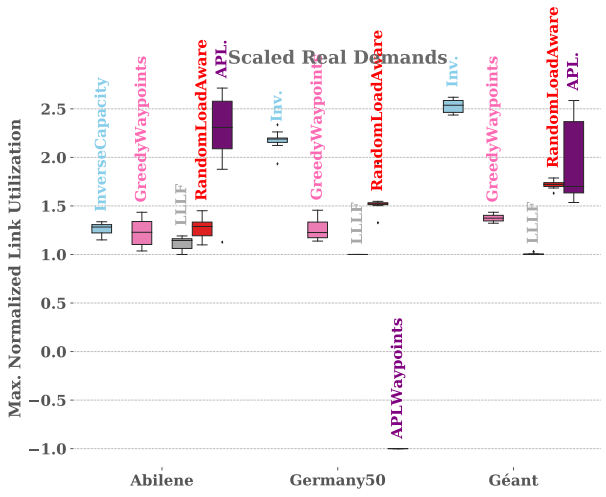


**Abbildung 7: Projekt 1 — Replikation: MLU über Abilene, Géant und Germany50 (fremde Ergebnisse, repliziert).**

Zur Überprüfung der Nachvollziehbarkeit haben wir die Resultate einer anderen Gruppe für **Abilene**, **Géant** und **Germany50** repliziert (vgl. Abb. 7). Nach kleineren Startschwierigkeiten (Umgebung/Pfade, initiale Skriptaufrufe) liefen die Messungen vollständig durch und konnten ausgewertet werden. (vgl. Abb. 7)

## 7.2 Projekt 2 — TE_SR_experiments (Gruppe 3)

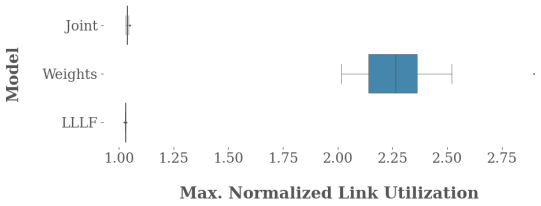


**Abbildung 8: Projekt 2 — Replikation: Ergebnisse Gruppe 3 (Boxplot-Übersicht).**

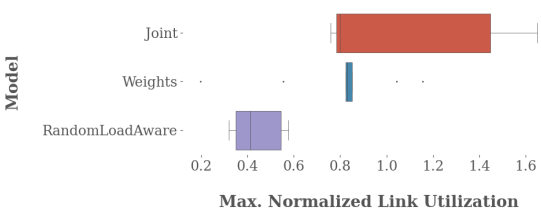


**Abbildung 9: Projekt 2 — Replikation: Graph RLA (Darstellung der Resultate).**

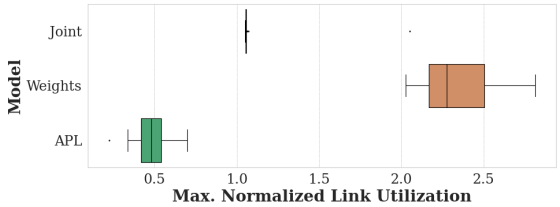


**Abbildung 10: Projekt 2 — Replikation: Result_02 (Boxplot-Detail).**

Die Replikation der Ergebnisse von **Gruppe 3** im *TE_SR_experiments*-Setup konnte *weitgehend problemlos* über `nanonet_batch` durchgeführt werden. Auf der genutzten VM traten jedoch vereinzelt vorzeitige Terminierungen auf; zur Absicherung wurde die Aufräumphase konservativ verzögert (`sleep(self.TIME + 60)` in `nanonet_batch`), sodass `throughput.py -s` vollständige Messergebnisse verarbeiten konnte. Die replizierten Boxplots und Graphdarstellungen (Abb. 8–10) reproduzieren die qualitativen Muster der Originalaussagen. Insgesamt stützen die Replikationen die korrekte Umsetzung der Toolchain und erhöhen die Nachvollziehbarkeit der im Report gezogenen Schlüsse.

# Literatur / Quellen

### Basis/Frameworks

- **TE_SR_WAN_simulation:** https://github.com/tfenz/TE_SR_WAN_simulation
- **TE_SR_experiments_2021:** https://github.com/nikolaussuess/TE_SR_experiments_2021
- **Nanonet:** https://github.com/nikolaussuess/nanonet

### Eigene Repositories (Ergebnisse/Implementierung)

- **Projekt 1:** https://github.com/Fisch96/Results_fachprojekt_group_4
- **Projekt 2:** https://github.com/Fisch96/TE_SR_experiments_2021-Gruppe4

### Replizierte Gruppen/Repos

- **Gruppe 3 Projekt 1:** https://github.com/Makeandbreak09/TE_SR_WAN_simulation
- **Gruppe 3 Projekt 2:** https://github.com/Makeandbreak09/TE_SR_experiments_2021

**Temporary page!**

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.