technische universität
dortmund

Projekt 2

Simon Buschmann, Yannik Buchner - Least Loaded Link First (LLLF)
Nikita Podibko, Jan Draeger - Avarage Link Utilization/Random Load Aware
Johannes Heinrich, Malek Haoues Rhaiem - Average Path Length (APL)

TU Dortmund University - Fakultät für Informatik

Inhaltsverzeichnis

Genereller Ablauf

Was gab es für Probleme?

Probleme:

Lösung:

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt

Lösung:

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt

Lösung:

- *.topo.sh excecutable machen

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt

Lösung:

- *.topo.sh excecutable machen

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt

Lösung:

- *.topo.sh excecutable machen
- als root das Progam ausführen

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt
- Wie implementieren wir unseren Algorithmus?

Lösung:

- *.topo.sh excecutable machen
- als root das Progam ausführen

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt
- Wie implementieren wir unseren Algorithmus?

Lösung:

- *.topo.sh excecutable machen
- als root das Progam ausführen
- nutze Nanonet um topologie aus dem Projekt 1 von *.json zu *.topo.py zu *.topo.sh konvertieren

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt
- Wie implementieren wir unseren Algorithmus?
- Auch hier werden *.topo.py und *.topo.sh nicht erstellt

Lösung:

- *.topo.sh excecutable machen
- als root das Progam ausführen
- nutze Nanonet um topologie aus dem Projekt 1 von *.json zu *.topo.py zu *.topo.sh konvertieren

Was gab es für Probleme?

Probleme:

- throughput.json wird nicht erstellt
- flow.txt wird nicht erstellt
- Wie implementieren wir unseren Algorithmus?
- Auch hier werden *.topo.py und *.topo.sh nicht erstellt

Lösung:

- *.topo.sh excecutable machen
- als root das Progam ausführen
- nutze Nanonet um topologie aus dem Projekt 1 von *.json zu *.topo.py zu *.topo.sh konvertieren
- als root das Progam ausführen

Was gab es für Probleme?

Probleme:

Lösung:

Was gab es für Probleme?

Probleme:

- nuttcp not in Server/Client Mode error

Lösung:

Was gab es für Probleme?

Probleme:

- nuttcp not in Server/Client Mode error

Lösung:

- Zu geringe demands entfernen

Was gab es für Probleme?

Probleme:

- nuttcp not in Server/Client Mode error
- Wartezeit pro Test zu lange

Lösung:

- Zu geringe demands entfernen

Was gab es für Probleme?

Probleme:

- nuttcp not in Server/Client Mode error
- Wartezeit pro Test zu lange

Lösung:

- Zu geringe demands entfernen
- Abfrage, ob tests durchgelaufen sind

Least Loaded Link First

LLLF Algorithmus

Wie funktionierte der Algorithmus nochmal?

- Nutzt noch nicht stark benutzte Routen
- Wiederholtes Ausführen führt zu immer besserem Ergebnis

Wie sieht die Topologie aus?

- In Projekt 1 haben wir festgestellt, dass er besonders gut bei dichten Netzen funktioniert.
- Wir wählen eine sehr dichte Topologie.
- 11 Nodes, beinahe alle verbunden

Topologie

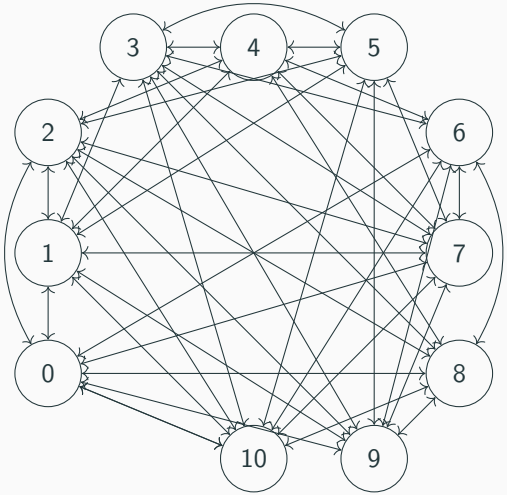


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

Topologie

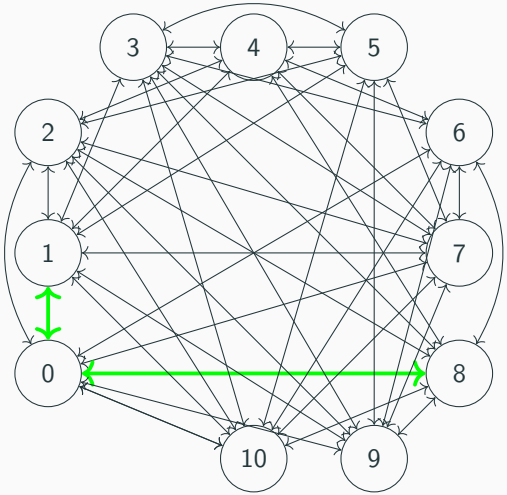


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$

Topologie

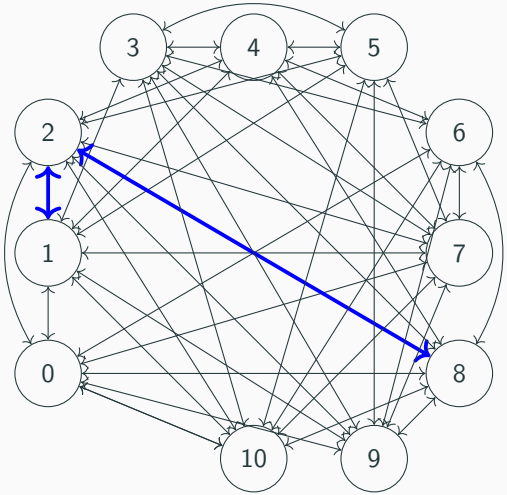


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$

Topologie

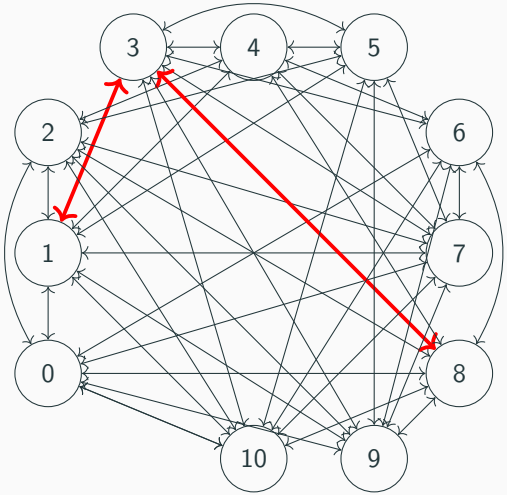


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$

Topologie

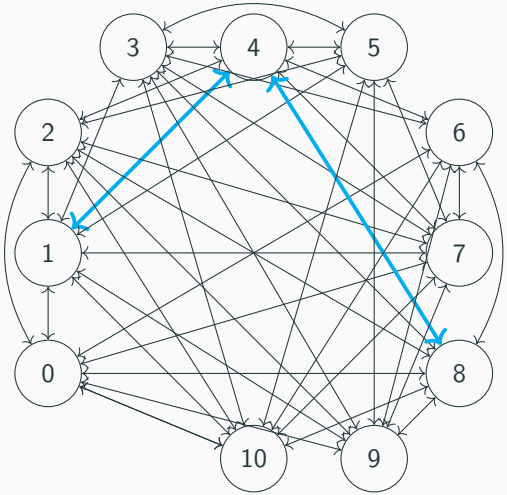


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$

Topologie

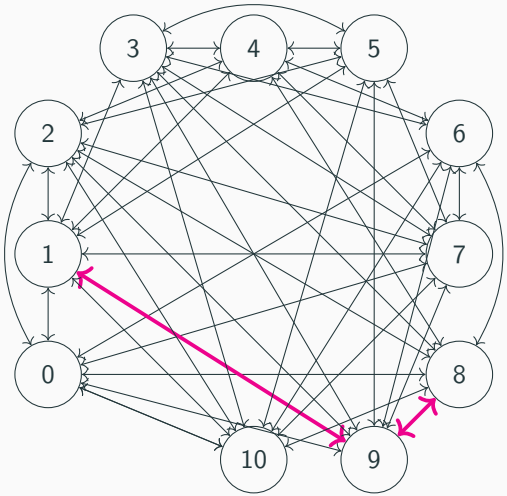


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$

Topologie

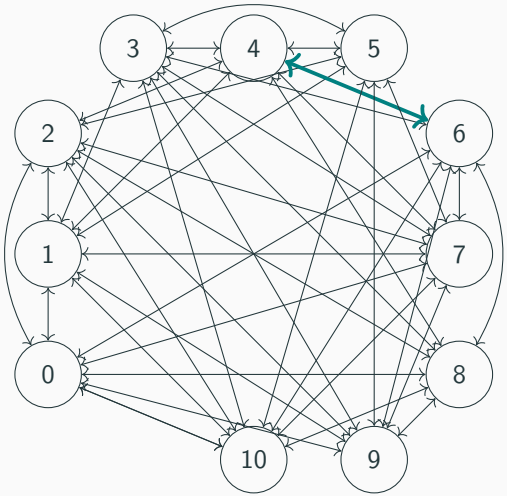


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$

theoretische MLU: 0.5

Topologie

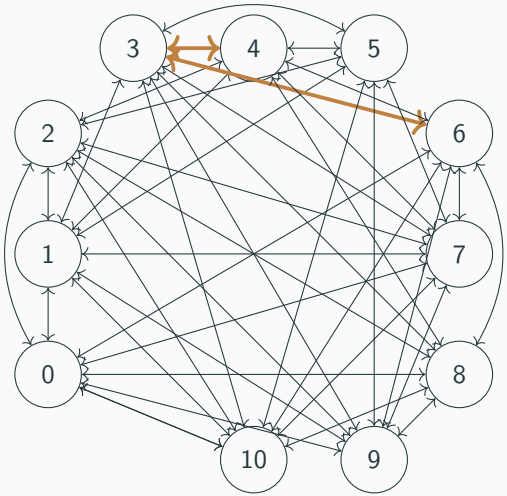


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$

theoretische MLU: 0.5

Topologie

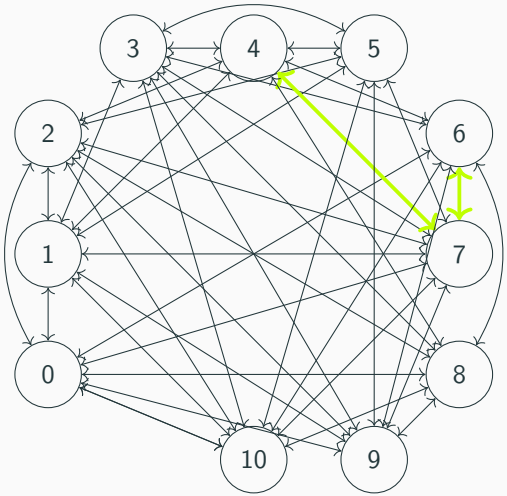


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$

theoretische MLU: 0.5

Topologie

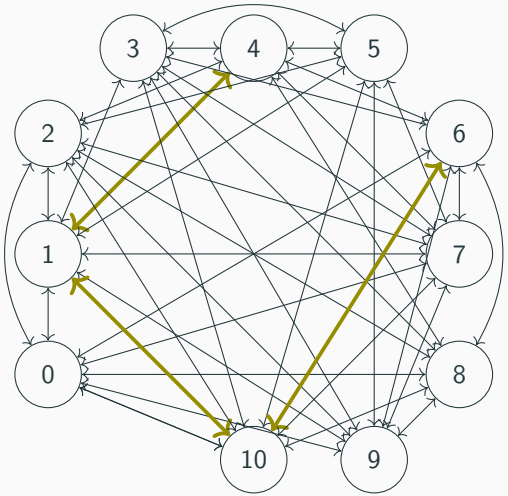


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$

theoretische MLU: 0.5

Topologie

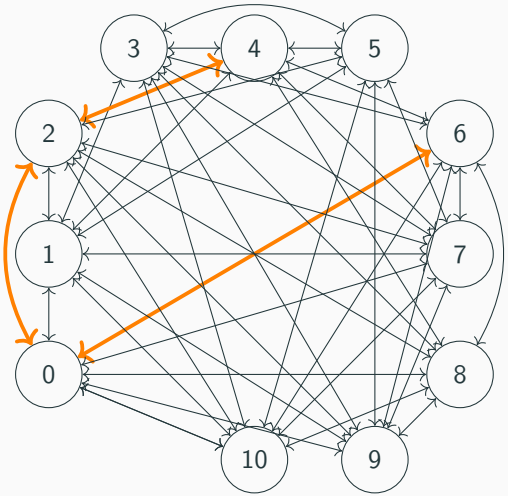


Figure 1: LLLF

Alle Kanten: capazity=2.0

Demands:

- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$

theoretische MLU: 0.5

Results

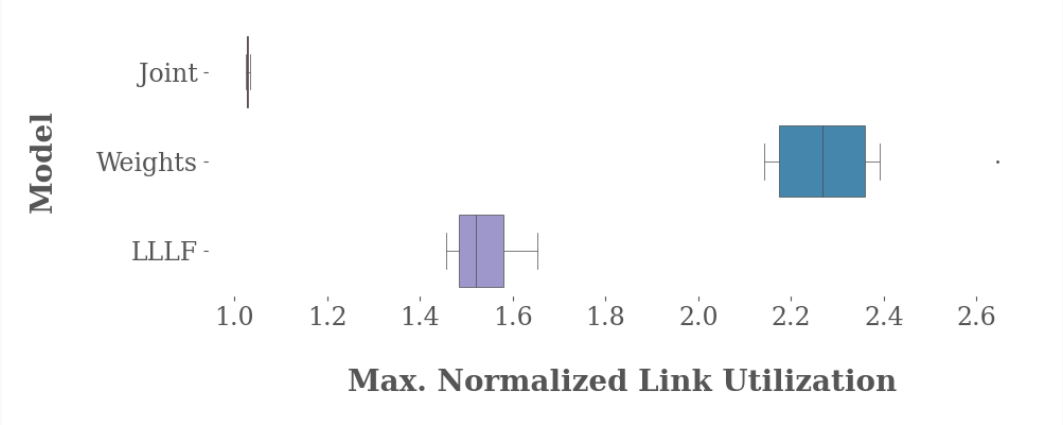


Figure 2: Results

Warum?

Was könnte das Problem sein?

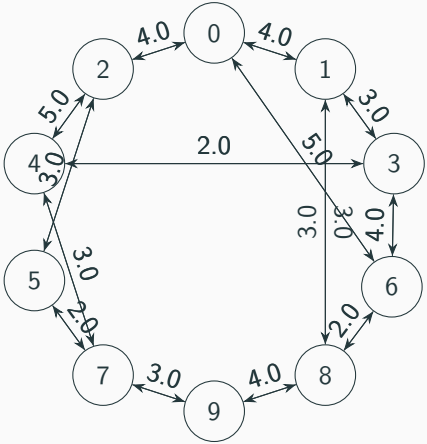
- Die ersten paar tests haben irgendwie Fehler:
  - nuttcp-t: v6.1.2: Error: connect: Connection refused
  - nuttcp-r: v6.1.2: Error: bind: Address already in use
- Auch könnte es Probleme bei mehreren Demands mit selben Start und Ziel geben.
- Vielleicht war die Topologie auch zu komplex

Randomized Load Aware

RandomizedLoadAware

Topologie-Eigenschaft	**Warum schlechter für `RandomizedLoadAware`?**
Geringe Pfadvielfalt	Kein Auswahlspielraum für Pfadverteilung.
Zentrale Engpässe	Kann nicht umleiten, wenn es keine Alternativen gibt.
Ungenaue oder unvollständige Kapazitäten	Bewertungsfunktion wird verzerrt.
Zu hohe Netzgröße	Pfaderzeugung / Laufzeit steigt exponentiell.

Topologie



Ergebnisse

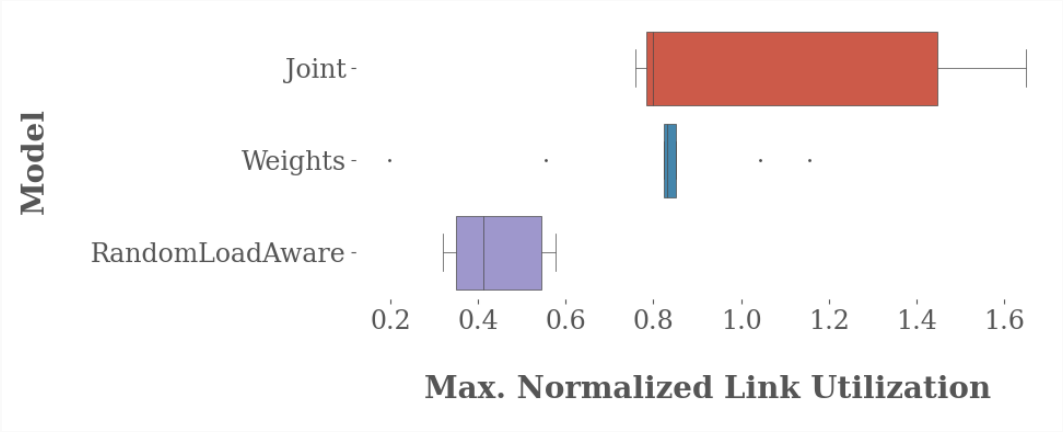


Figure 3: Results

Ergebnisse

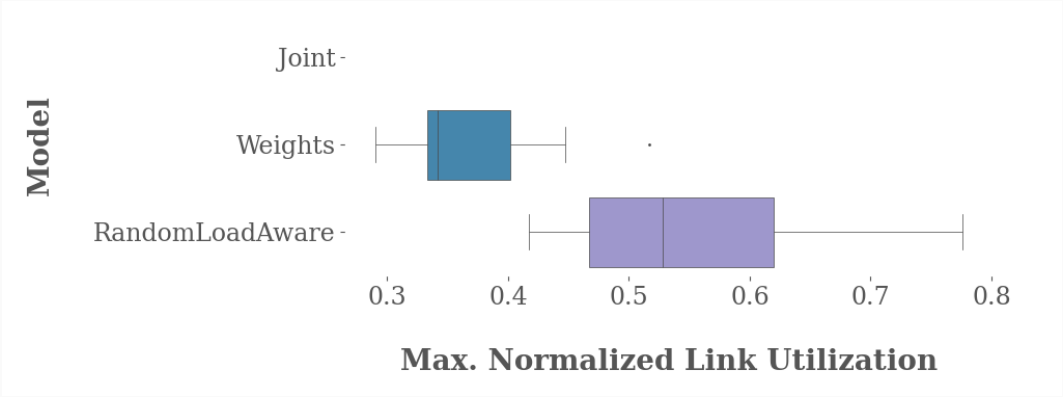
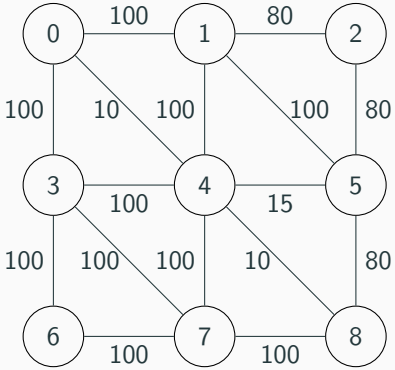


Figure 4: Results

Average Path Length

Topologie


0
1
2
3
4
5
6
7
8
100
80
100
10
100
100
80
100
15
100
100
100
10
80
100
100

Vorteile der Topologie

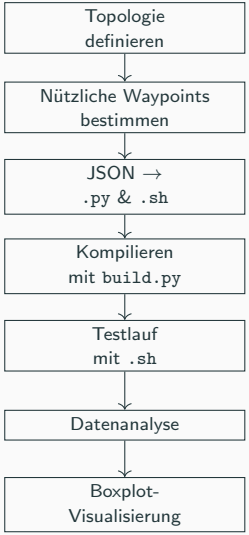
- **Pfad Diversität**: Durch viele Pfadmöglichkeiten
- **Tradeoff Visualisierung**: Visualisiert gut tradeoff zwischen Pfadlänge (Diagonale Wege) und MLU (Alternative Wege)
- **Waypoint Nutzwert**: Auch durch Pfad Diversität
- **Vorteilhafte Umwege**: Durch Kapazitätssetzung

Nachteile/Probleme

- **Speziallisiert:** Nur Kompetetiv gegen andere Algorithmen unter speziellen Umständen
- **Nicht verallgemeinbar:** Ergebnisse sind auf unterschiedlichen Topologien
- **Nichts neues:** Für gute Testergebnisse niedriges Lambda nötig $\rightarrow$ wird zu DemandWaypoints Algorithmus

Vorgehensweise

1. **Topologie-Modellierung:** Definition der Netzwerkstruktur im JSON-Format mit Knoten, gerichteten Links, Kapazitäten, Gewichten und Demands.
2. **APL-Optimierungslogik:** Bewertung möglicher Waypoints basierend auf Hop-Anzahl, Betweenness und Volumen → Generierung der `waypoint_chance_map`.
3. **Konvertierung:** Umwandlung in `.topo.py` und `.topo.sh` mit angepasstem Skript.
4. **Kompilierung und Test:** Build via `build.py`, Ausführung via `nanonet_batch.py`.
5. **Datenerhebung:** Analyse von `*.throughput.json`, `flow_X-Y.txt`.
6. **Visualisierung:** Erstellung von Boxplots mit `plot_results.py`.



Ergebnisse

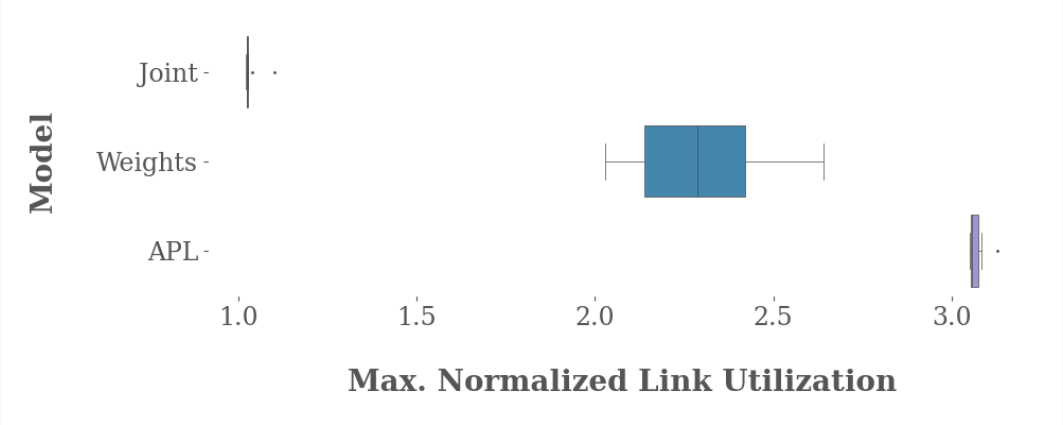


Figure 5: Ergebnisse APL

Ergebnisse

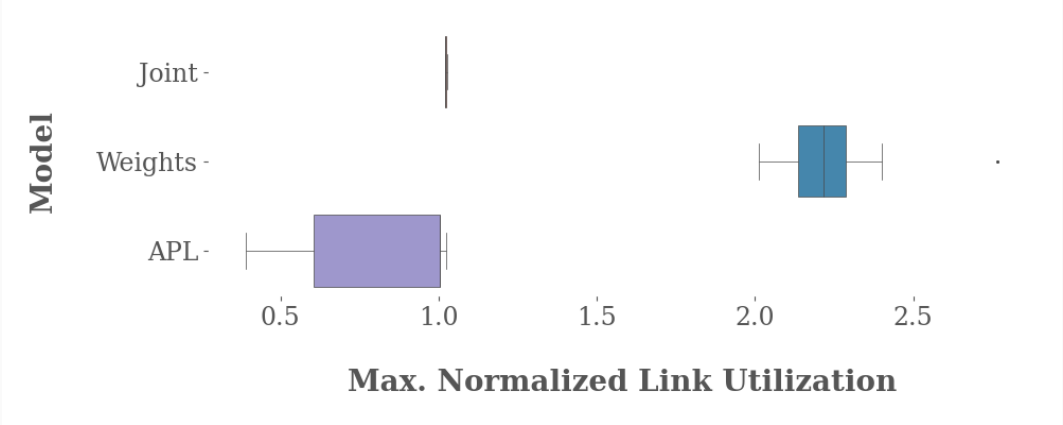


Figure 6: Ergebnisse APL

Fazit

- Sorgt für geringe Latenzen
- Kann andere Algorithmen in richtige Richtung neigen
- Aber optimiert nicht nur die MLU
- **Für meiste Netze am sinnvollsten Regelmäßig kürzeste Pfade $\rightarrow$ MLU optimieren**
- **Algorithmus nur in sequentieller Nutzung mit anderen Algorithmen, die MLU optimieren, sinnvoll**

Reproduktion

Reproduktion

Wie lief die Reproduktion?

- Der Code lief ohne Probleme durch
- Unsere Ergebnisse sind aber aus irgendeinem Grund besser
  - Vielleicht weil mein Computer besser ist

Reproduktion Ergebnisse

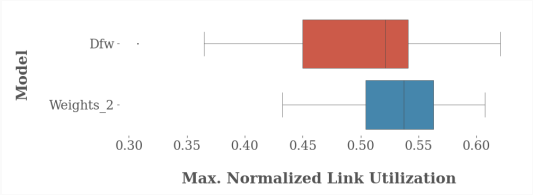


Figure 7: Originial 1

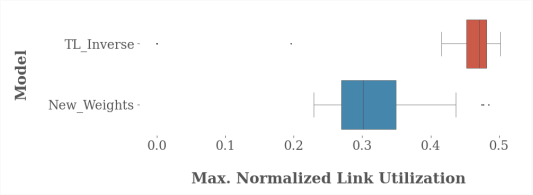


Figure 9: Originial 2

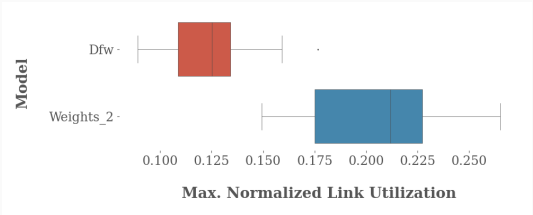


Figure 8: Reproduktion 1

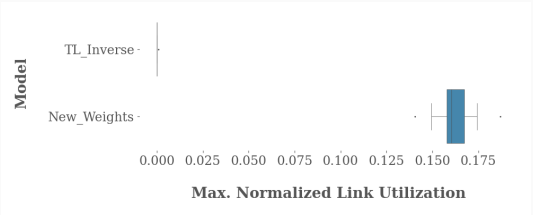


Figure 10: Reproduktion 2

Fragen?

Noch Fragen?