TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Nachrichtentechnik

Actual Topics Projekt

zum Thema

Python basierte Simulation des Helfer-Knotens im drahtlos vermaschten Netzwerk

vorgelegt von Xiang, Zuo
im Studiengang Informationstechnik, Jg. 2014
geboren am 21.08.1991 in China

Betreuer: Dipl.-Inf. Frank Wilhelm
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. Dr. h.c. Frank H.P. Fitzek
Tag der Einreichung: 17. Januar 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Wegen der Eigenschaft vom drahtlos vermaschten Netzwerk(Wireless mesh network - WMN) könnten Knoten in der Nähe von dem Sender diese Übertragung mithören. In Anbetracht dessen, dass die drahtlose Kanäle normalerweise unter signifikanten Paketverlusten leiden. Könnte dieses Mithören eine potentielle Möglichkeit für die Verbesserung der Gesamtleistung durch opportunistische Sendung von empfangenen Paketen.

Andererseits wegen der Konkurrenz des Zugriffs auf drahtloses Kanal dürfen Sender und Helfer nicht zur gleichen Zeit Pakete zu transportieren. Dadurch könnte der Helfer nur die Sendezeit des Senders besetzen, wenn er keine neue Informationen dem Empfänger anbietet. [1]

Deshalb könnte die Untersuchung nach unterschiedlichen Sendestrategien des Helfer-Knoten ein wichtiges Thema sein.

1.2 Zielsetzung der Arbeit

Im Rahmen dieser Arbeit sollte ein Simulationsprogramm mit Kodo-Python aufgebaut, damit zwei unterschiedliche Strategien der Weitersendung am Helfer-Knoten verglichen werden könnten. Nämlich hier zuerst ohne Helfer und dann mit einem Helfer-Knoten, das den PlayNCool Algorithmus verwendet.

1.3 Struktureller Aufbau der Arbeit

Im Kapitel 2 werden das Szenario, genaue Arbeit sowie Werkzeug vorgestellt. Danach werden Simulationsergebnisse unterschiedlicher Parameter mit im Kapitel 3 durch Tabelle und Grafiken veranschaulicht. Abschließend werden im Kapitel 4 eine Zusammenfassung und einige Ausblicke in Bezug auf Weiterentwicklungen von dieser Arbeit sowie Verbesserungen der Simulation geliefert.

2 Methode

2.1 Aufgabenbeschreibung

2.1.1 Szenario der Simulation

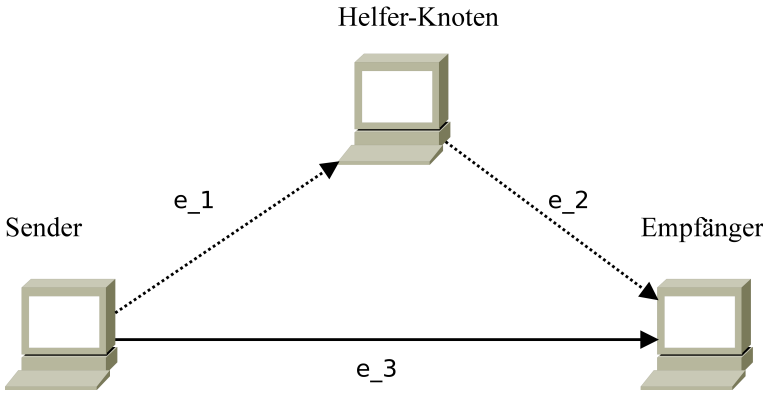


Abbildung 2.1: Szenario der Simulation

Weil in der Abbildung 2.1 beschriebenes Szenario der häufigste Fall in jeder Verbindung des vermaschten Netzwerks ist, sollte es in dieser Aufgabe detailliert analysiert und simuliert.

Die Paketverlustwahrscheinlichkeiten zwischen Sender, Helfer-Knoten sowie Empfänger werden jeweils durch e_1, e_2 sowie e_3 charakterisiert. Um die Analyse sowie Simulation zu vereinfachen, wird in dieser Aufgabe angenommen, dass Verbindungen des Helfer-Knotens symmetrisch sind($e_1 = e_2$).

2.1.2 Simulationsarbeit

Wie in der Einleitung erwähnt, dass die heuristische Weitersendung des Helfer-Knotens zur Verbesserung der Netzwerkleistung beitragen könnte. Zwei Situationen weiden in dieser Aufgabe mit Kodo-Python simuliert und auch verglichen.

(1) Datenübertragung mit RLNC ohne Helfer-Knoten

(2) Datenübertragung mit RLNC mit Helfer-Knoten

Bei der zweiten Situation gibt es viele unterschiedliche Strategien, in dieser Aufgabe wird der PlayNCool Algorithmus[1] mit Kodo-Python implementiert und simuliert.

Weil zufällige Netzwerkkodierung bei den Simulationen verwendet wird, können Sender, Helfer-Knoten und Empfänger auch als Encoder, Recoder sowie Decoder genannt.

Wenn das Helfer-Knoten Packte zu senden beginnt, liegen verschiedene Varianten für die Disposition zwischen Sender und Helfer vor. In dieser Simulation wird eine vereinfachte Annäherung von TDMA verwendet, nämlich Sender und Helfer besetzen das Kanal alternativ.

Bei jeder Simulation werden Paketverlustwahrscheinlichkeiten(nämlich e_1, e_2, e_3) als Variablen eingestellt. Um einen Vergleich durchzuführen, wird die Anzahl von Paketen bekommen, die insgesamt und auch nur von dem Helfer gesendet werden. Danach werden Daten mehrerer Simulationen verarbeitet und in Grafiken mit Konfidenzintervallen dargestellt. In dieser Arbeit wird der Test für bestimmten Parameter jeweils 50 mal durchgeführt und eine Student-Verteilung bei der Schätzung verwendet.

2.2 Algorithmus

2.2.1 RLNC

RLNC(Random Linear Networkcoding) ist eine einfache aber leistungsfähige Kodierungsschema.[2] Zusammenfassend senden Knoten im Netzwerk zufällige lineare Kombinationen von empfangenen Paketen, um z.B. den Durchsatz zu erhöhen. Laut der Forschung bietet RLNC wegen der Fähigkeit von Rekodierung große Vorteile bei Multi-Hop-Netzwerks. [3]

2.2.2 PlayNCool

PlayNCool ist ein opportunistisches Protokoll mit Netzwerkkodierung für lokale Optimierung vom Rounting im drahtlos vermaschten Netzwerk. [1]

Eine wichtige Leistungsmerkmal des Algorithmus ist, dass das Helfer-Knoten nicht sofort sondern nach gewisser Zeit Pakete transportiert, um nützlichere Informationen dem Empfänger anzubieten. Wegen der Vermeidung von Konkurrenz des drahtlosen Kanals an der Anfangsphase, die Leistung der Verbindung wird verbessert.

Für die Erzeugung der kodierten Pakte mit der Generationsgröße g wird RLNC verwendet. Die Anzahl der Paketen am Helfer-Knoten, die von dem Sender transportiert werden, wird als r definiert. Dieser Wert sollte als der Schwellenwert für den Anfang der Paketübertragung zwischen Helfer und Empfänger.

Laut der PlayNCool Politik wird r durch folgende Formel berechnet. [1]

$$r = -g \cdot \frac{C}{D - (1 - e_3) \cdot C} \tag{2.1}$$

mit

$$C = -1 + e_2 + e_3 - e_1 \cdot e_3 \tag{2.2}$$

$$D = (2 - e_3 - e_2) \cdot (e_3 - e_1 \cdot e_3) \tag{2.3}$$

2.3 Vorstellung der Simulationswerkzeuge

Die Simulationscode sowie Dokumentationen steht im Repository(`https://github.com/stevelorenz/acto15-helper`) zur Verfügung.

2.3.1 Kodo

Kodo ist eine kommerzielle Bibliothek für Netzwerkkodierung entwickelt von Steinwurf. Kodo ermöglicht die Verwendung von RLNC mit hoher Leistung in einer Vielzahl von Produkten und Applikationen. Kodo ist mit C++ geschrieben und stellt viele Anbindungen für andere Programmiersprachen zur Verfügung, wie Python, Java usw. Mehr Informationen sind auf der Webseite(`http://steinwurf.com/kodo/`) verfügbar[4]

In meiner Simulation wird binäre RLNC-Kodierung mit Hilfe der Kodo-Python realisiert. Informationen von Kodo-Python steht auf der Website(`https://github.com/steinwurf/kodo-python`) zur Verfügung. [5]

2.3.2 Matplotlib

Für die Erstellung der Grafiken wird Python-Bibliothek Matplotlib verwendet. Damit kann die Daten in schöner graphische Darstellung schnell erstellen. Eine Vielzahl von Beispielen steht auf der Galerie zur Verfügung. In dieser Arbeit wird Errorbar für Grafiken mit Konfidenzintervallen benutzt.(`http://matplotlib.org/1.2.1/examples/pylab_examples/errorbar_demo.html`)

3 Ergebnisse

3.1 Simulationsparameter

In Simulationen könnten $e_2(e_1 = e_2)$ und auch e_3 als Variable betrachtet werden. Um einen Vergleich zu machen, nimmt e_2 jeweils einen bestimmten Wert(hier 30%, 60% und 90%) und variiert e_3 von 0% bis 95% mit dem Abstand von 5%.

Als Kriterien für die Durchführung der Übertragung werden zwei Kenngröße betrachtet, nämlich die gesamte gesandte Pakte und die Paket, die vom Helfer-Knoten gesandt werden. Damit wird der Beitrag des Helfers zur Übertragung bildhaft dargestellt.

3.2 Simulationsergebnisse

In folgenden Tabellen und Grafiken werden Simulationsergebnisse gezeigt und analysiert. Um Linien deutlicher zu zeigen, stehen bei jeder Simulation Abbildungen mit halbem Intervall von e_3 zur Verfügung.

Zuerst wird e_2 als 30% angenommen und die Ergebnisse werden in Tabelle 3.1, Abbildung 3.1 sowie 3.2 gezeigt:

Daraus zeigt es sich zuerst, dass der Mittelwert der Situation ohne Helfer eine offensichtliche steigende Tendenz aufweist. Insbesondere nach 85%, wächst der Wert von 678.74 bis 2035.38 über 1023.36 relativ schnell. Im Vergleich dazu nimmt der gesamte Mittelwert der Situation mit Helfer mit einer niedrigen und stabilen Geschwindigkeit zu. Beim schlechtesten Fall, müsste der Sender ohne Helfer ungefähr siebenfach so viele Pakte transportieren als mit dem Helfer. Wenn es um die empirische Standardabweichung zu kommen, stellen Werte eine gleiche Tendenz wie Mittelwert dar. Nämlich während die Situation ohne Helfer einen deutlichen Wachstum zeigt, bleibt sie mit Helfer beinahe unverändert bei schlechten Fällen.

Die Anzahl der gesandten Paketen vom Helfer vergrößert sich mit der ansteigenden Paketverlustrate. Damit kann es vermutet werden, dass der Helfer immer mehr zur der Übertragung beiträgt. Außerdem wird es in der Abbildung 3.2(a) gezeigt, dass es einen Schnittpunkt(hier ungefähr um 30%) zwischen zwei Linien(ohne Helfer und gesamte Anzahl mit Helfer) gibt.

Danach hat die Übertragung mit Helfer immer eine bessere Leistung. Deshalb könnte dieser Wert als einen Schwellenwert betrachtet werden, ob ein Helfer für die Übertragung eingestellt werden sollte.

Jetzt wird e_2 auf 60% vergrößert und die Ergebnisse werden in Tabelle 3.2, Abbildung 3.3 sowie 3.4 gezeigt:

Daraus kann man erfahren, dass im Vergleich zu einer besseren Verbindung des Helfers, fangt der Helfer nur nach größerem Wert von e_3(hier ungefähr nach 40%)an, Paket zu transportieren. Und der Schnittpunkt wird auch nach ungefähr 60% verschoben.

Deshalb kann Schlussfolgerungen daraus gezogen, dass in diesem Szenario(Symmetrie von e_1 und e_2) die Übertragungsleistung offensichtlich verbessert durch die Einstellung des Helfer-Knotens wird, wenn die Paketverlustrate zwischen Sender und Empfänger(e_3) relativ schlechter als die zwischen Helfer und Empfänger($e_1 = e_2$).

Tabelle 3.1: Simulationsergebnisse mit $e_2 = 30\%$

	Mittelwert			Empirische Standardabweichung		
e_3(%)	Ohne Helfer	gesamt mit Helfer	nur vom Helfer	Ohne Helfer	gesamt mit Helfer	nur vom Helfer
0	100.00	100.00	0.00	0.00	0.00	0.00
5	107.42	107.34	0.00	2.78	2.99	0.00
10	112.14	112.52	0.00	4.10	4.10	0.00
15	120.14	120.68	0.98	5.07	5.15	2.31
20	128.78	126.10	3.64	6.30	6.04	3.98
25	135.72	134.58	9.82	7.02	7.68	4.91
30	147.78	145.42	18.24	9.53	7.79	5.51
35	155.64	154.44	26.94	8.22	9.08	5.11
40	171.12	163.40	34.54	9.83	10.18	5.36
45	184.18	174.00	41.60	13.49	11.08	6.36
50	201.22	182.44	49.14	12.34	12.08	6.33
55	226.82	192.78	59.36	15.22	14.02	7.29
60	255.98	200.36	66.70	21.57	12.46	6.47
65	285.30	213.96	76.74	24.43	11.55	5.87
70	346.46	222.96	85.60	25.45	12.97	6.57
75	410.56	239.40	97.20	36.52	12.88	6.30
80	505.92	249.86	106.38	39.23	11.55	5.53
85	678.64	261.86	117.00	65.48	13.46	6.41
90	1023.36	277.60	130.10	91.24	12.87	6.41
95	2035.38	296.10	144.42	203.36	14.53	6.98

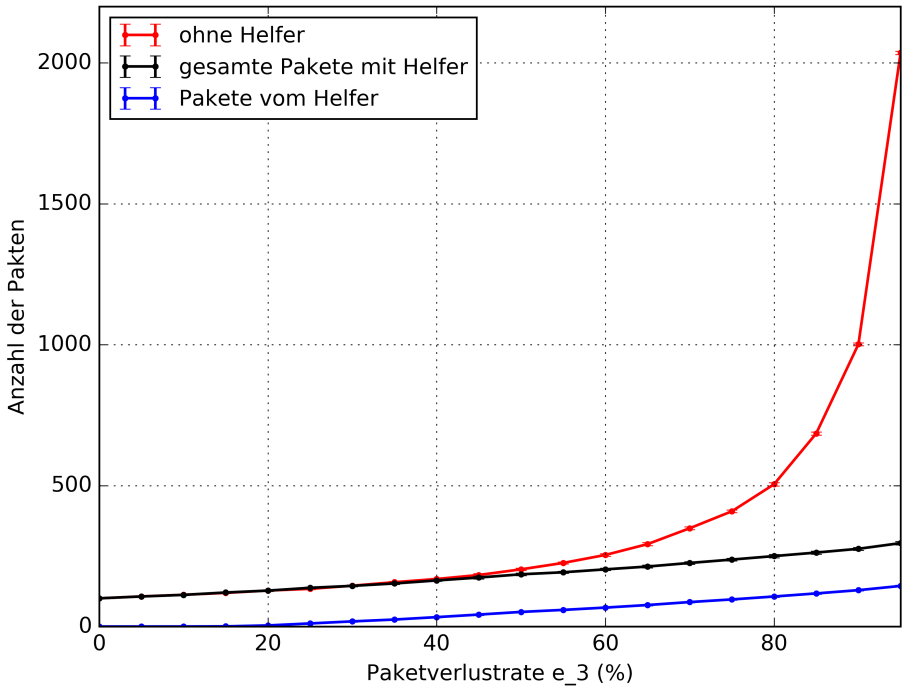
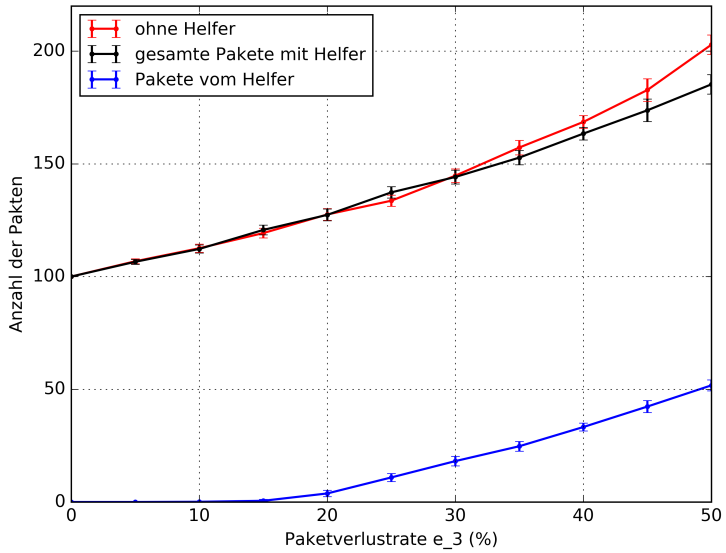


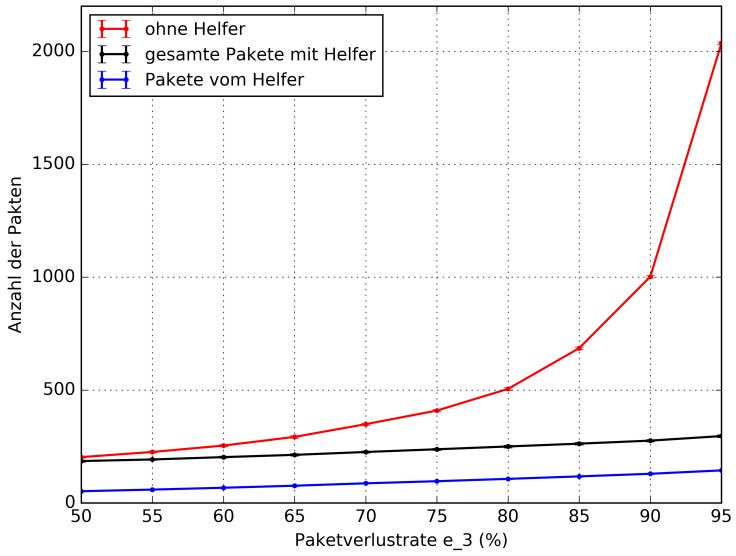
Abbildung 3.1: Simulationsergebnisse mit $e_2 = 30\%$

Tabelle 3.2: Simulationsergebnisse mit $e_2 = 60\%$

	Mittelwert			Empirische Standardabweichung		
$e_3(\%)$	Ohne Helfer	gesamt mit Helfer	nur vom Helfer	Ohne Helfer	gesamt mit Helfer	nur vom Helfer
0	100.00	100.00	0.00	0.00	0.00	0.00
5	107.28	106.44	0.00	3.80	3.47	0.00
10	112.88	112.78	0.00	4.14	4.53	0.00
15	119.26	119.36	0.00	5.03	6.07	0.00
20	127.10	127.42	0.00	6.45	6.04	0.00
25	136.40	136.78	0.00	6.85	7.44	0.00
30	143.94	144.28	0.00	8.39	9.13	0.00
35	155.70	156.12	0.00	8.65	9.05	0.00
40	169.66	169.66	0.88	10.12	11.82	2.90
45	184.06	186.42	2.90	12.98	13.09	5.25
50	201.44	208.14	16.94	15.79	15.61	11.46
55	222.68	226.86	29.38	16.08	17.88	12.62
60	252.92	254.80	51.08	17.73	20.37	12.71
65	295.98	285.74	69.50	24.11	22.08	14.44
70	339.70	310.16	90.02	32.00	21.36	11.90
75	408.36	338.68	114.06	37.88	27.80	15.61
80	519.02	367.08	135.60	49.39	29.14	15.18
85	673.08	402.90	164.98	52.05	28.63	15.01
90	1039.84	445.94	197.58	101.47	29.32	15.64
95	2016.22	495.54	234.82	212.14	34.33	16.45



(a) e_3 von 0 bis 50%



(b) e_3 von 50 bis 95%

Abbildung 3.2: Simulationsergebnisse verschiedener Intervalle mit $e_2 = 30\%$

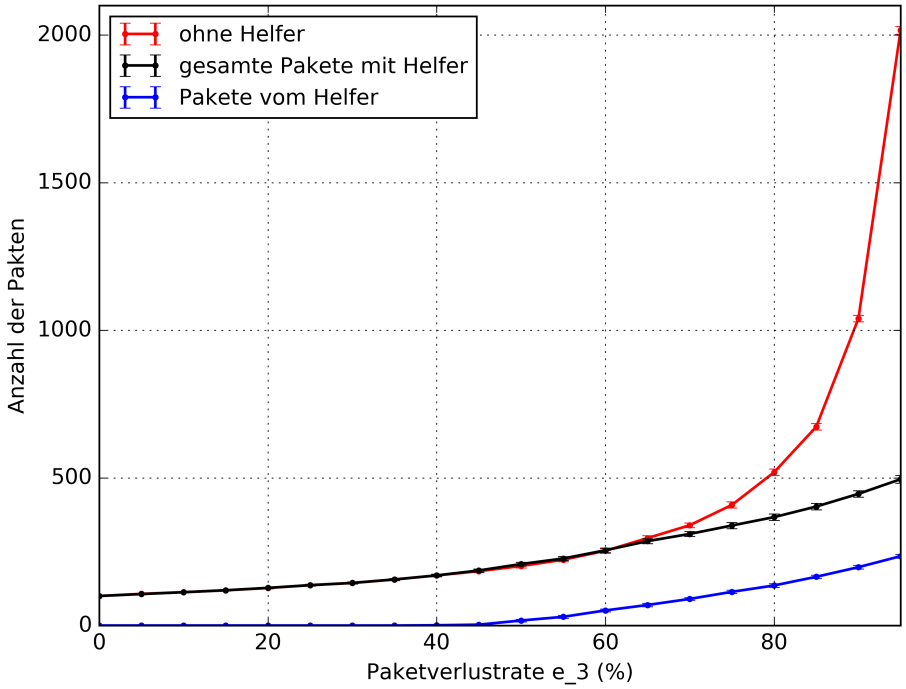
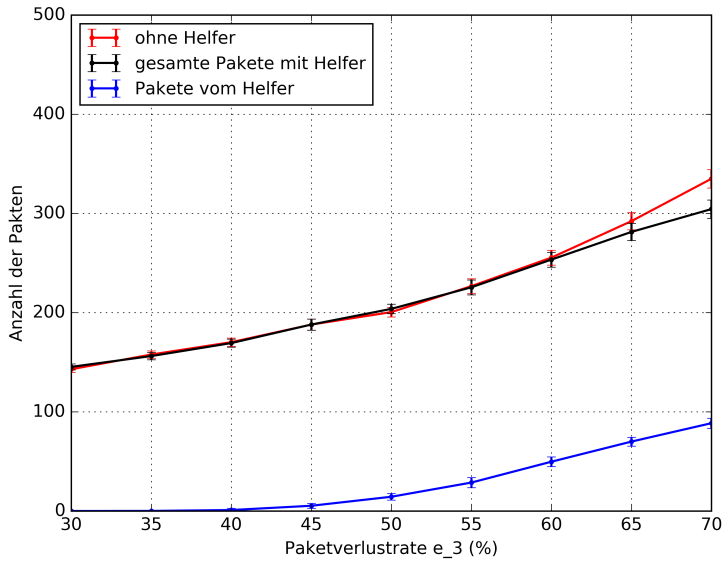
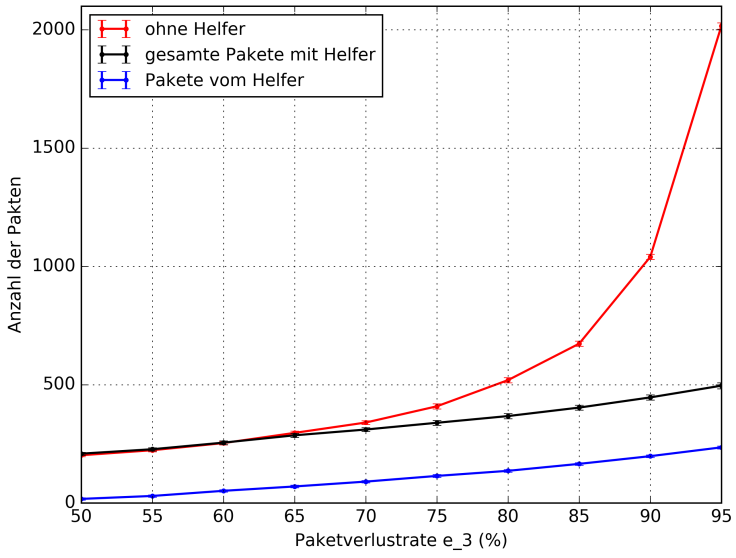


Abbildung 3.3: Simulationsergebnisse mit $e_2 = 60\%$



(a) e_3 von 30 bis 70%



(b) e_3 von 50 bis 95%

Abbildung 3.4: Simulationsergebnisse verschiedener Intervalle mit $e_2 = 60\%$

4 Zusammenfassung und Aussicht

4.1 Zusammenfassung

Das Ziel dieser Arbeit ist der Aufbau sowie Analyse der Simulation für das in Kapitel 2.1.1 beschriebene Szenario mit Hilfe vom Kodo-Python und die Ergebnisse mit Tabellen und Grafiken darstellen.

Bei der Situation mit dem Helfer-Knoten liegen viele Strategien vor, in dieser Aufgabe wird der auf RLNC basierte PlayNCool Algorithmus implementiert und simuliert.

Aus den Ergebnissen könnte einige Schlussfolgerung gezogen werden:

In diesem Szenario(Symmetrie von e_1 und e_2) die Übertragungsleistung(hier weniger gebrauchte Pakete und auch Schwankungen) wird offensichtlich verbessert durch die Einstellung des Helfer-Knotens, wenn die Paketverlustrate zwischen Sender und Empfänger(e_3) relativ schlechter als die zwischen Helfer und Empfänger($e_1 = e_2$).

Die Simulationscode sowie Dokumentationen steht im Repository(`https://github.com/stevelorenz/acto15-helper`) zur Verfügung.

4.2 Aussichten

Um die Simulation zu vereinfachen, werden einige nicht allgemeine Voraussetzung in dieser Arbeit angenommen. Diese könnten als weitere Arbeiten betrachtet.

4.2.1 Unsymmetrische Paketverlustrate des Helfer-Knotens

In dieser Simulation nimmt e_1 den gleichen Wert als e_2, deshalb wird die Anzahl der empfangenen Paketen am Helfer als Schwellenwert für den Anfang der Übertragung. Aber wenn diese Symmetrie nicht vorliegt, muss man überlegen und testen, ob dieser Kriterium weiter gilt. Es gibt z.B. Ausnahmezustand: Obwohl die Verbindung zwischen Helfer und Empfänger relativ gut ist, aber wegen der schlechten Übertragung zwischen Sender und Helfer sind die Pakete vom Helfer linear abhängig und haben damit keinen Beitrag zur Leistung.

4.2.2 Übertragung mehrere Generationen von Paketen

In dieser Simulation wird nur eine Generation von Paketen gesandt. Wenn die transportierte Informationen ziemlich groß ist, mehrere Generation sind notwendig. Damit entsteht viele komplexere Probleme, die bei der Simulation berücksichtigt werden sollten. Zum Beispiel, wie sollte der Sender sich verhalten, wenn er auf die Empfangsbestätigung der letzten Generation vom Empfänger warten.

Literaturverzeichnis

[1] Peyman Pahlevani, Daniel E Lucani, Morten V Pedersen und Frank HP Fitzek. „Playncool: opportunistic network coding for local optimization of routing in wireless mesh networks“. In: *Globecom Workshops (GC Wkshps), 2013 IEEE*. IEEE. 2013, S. 812–817 (siehe S. 1, 4, 5).

[2] *Linear network coding*. Wikipedia. URL: `https://en.wikipedia.org/wiki/Linear_network_coding` (siehe S. 4).

[3] *Introduction to Network Coding*. Steinwurf, 2012. URL: `http://kodo-docs.steinwurf.com/en/latest/nc_intro.html` (siehe S. 4).

[4] *Introduction to Kodo*. Steinwurf. URL: `http://steinwurf.com/kodo/` (siehe S. 5).

[5] *Github Repository of Kodo-Python*. Steinwurf. URL: `https://github.com/steinwurf/kodo-python` (siehe S. 5).