Computation of Tight Bounds in Networks of Arbitrary Schedulers

Andreas Kiefer

Diplomarbeit

Computation of Tight Bounds in Networks of Arbitrary Schedulers

vorgelegt von

Andreas Kiefer

3. März 2009

Technische Universität Kaiserslautern
Fachbereich Informatik
AG disco | distributed computer systems lab

DISCO-D-015

Betreuer: Prof.Dr.-Ing. Jens B. Schmitt
Dipl.Inf. Nicos Gollan
Prüfer: Prof.Dr.-Ing. Jens B. Schmitt

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit mit dem Thema „Computation of Tight Bounds in Networks of Arbitrary Schedulers“ selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Kaiserslautern, den 3. März 2009

Andreas Kiefer

Mein Dank gilt:
Andrea, Christian, Malte
nicht nur für das Korrekturlesen
Prof. Schmitt und Nicos
für die gute Betreuung
Kalle, Marc, Michael
Natürlich meinen Eltern
u.v.m

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

1 Einleitung

Bevor der *Network Calculus* vor nunmehr 18 Jahren von Cruz [Cru91a] [Cru91b] vorgestellt wurde, konnte man für Netzwerke keine festen *Worst-Case* Szenarien berechnen. Erst mit der Entwicklung dieser Methode war es möglich, im Bereich *Quality of Service* maximale Grenzen für die *Performance Bounds* zu geben. Der Teilaspekt des *Delay Bounds* ist

Im Hinblick auf den immer wichtiger werdenden Echtzeitanspruch vieler Anwendungen, sind genaue Analysen äußerst wichtig. Eine beliebter werdende Anwendung ist die Nutzung des Internets um Multimedia-Anwendungen. Hier sind *Voice-over-IP (VoIP), Internet Protocol Television (IPTV),* Online-Spiele oder *Video Broadcasting* gängige Begriffe. Alle haben gemeinsam, dass nur eine möglichst geringe Verzögerung auftreten darf, damit keine visuellen oder auditiven Artefakte erfahrbar werden.

Eine andere, sehr interessante Anwendung ist die Realisierung eines Client-Server-Systems, mit dem Bandmitglieder von unterschiedlichen Orten aus über das Internet gemeinsam musizieren können [GDNW04]. Das System heißt *Network-centric Music Performance*.

Auch die steigende Verwendung von eingebetteten Systemen im alltäglichen Leben setzt eine gründliche Entwicklung voraus. Besonders, da diese immer günstiger werden und breitere Anwendungsmöglichkeiten finden. Die Automobilindustrie und Sensortechnik sind nur zwei von vielen Einsatzgebieten, in denen es auf Sicherheit, Zuverlässigkeit, Robustheit und Korrektheit der Systeme ankommt. Eine genaue Analyse der Komponenten und deren Zusammenspiel wird immer notwendiger.

Der Network Calculus verwendet zwei grundlegende Konzepte für die Netzwerk-Analyse. Der Datenverkehr wird durch eine *Arrival Curve* nach oben beschränkt. Die Garantien, die ein Server liefern kann, werden durch eine *Service Curve* beschrieben.

Für die Berechnung der Performance Bounds stützt sich der Network Calculus auf die *Min-Plus-Algebra*.

Ein aktuelles Problem ist die Frage nach der Berechnung von *Tight Bounds* im Bereich des Network Calculus. Lange war es nicht möglich, einen Tight Bound anzugeben. Für eine Abschätzung der Fehler und Abweichungen der bisherigen Berechnungen ist diese Grenze aber äußerst interessant.

Schmitt u.a. [SZF07] haben, unter der Voraussetzung von *Token-Bucket* beschränkten Arrival Curves und *Rate-Latency* garantierten Service Curves, dafür eine Berechnung entwickelt. Bei einer Verallgemeinerung auf stückweise lineare konkave Arrival und konvexe Service Curves ist dieser Bound nicht mehr Tight, weshalb die Berechnung *Optimisation-Based Bounding Method (OBA)* genannt wurde.

Die Methode zeigt, dass der Network Calculus ein grundlegendes Problem mit *Feed-Forward* Netzwerken hat. Die mathematischen Operationen verlangen eine Kommutativität, die in der Realität aber schwer umzusetzen ist, da sich Systeme nicht genauso leicht vertauschen lassen.

Die Lösung des Problems besteht darin, den Bereich der Min-Plus-Algebra zu verlassen und ein lineares Optimierungsproblem zu lösen. Dabei tritt allerdings eine kombinatorische Komplexität auf, die eine praktikable Berechnung für die meisten Szenarien unmöglich machen.

Ziel dieser Arbeit ist es, die auftretende Komplexität auf ein erträgliches Maß zu reduzieren. Ein bestehendes Software-Tool, in dem die Network Calculus Operationen schon implementiert sind, der *DISCO Network Calculator* [DIS08] wird zuerst mit der OBA erweitert. Zusätzlich werden Optimierungsverfahren implementiert, die den Suchraum nach geeigneten Lösungen überprüft.

Die Implementation ist nun Grundlage für Versuche, um erstens die Güte der Delay Bounds und zweitens die Laufzeit festzustellen.

2 Themenbezogene Arbeiten

Zurzeit gibt es vier verschiedene Software-Anwendungen, die sich mit der Umsetzung der Konzepte des Network Calculus beschäftigen. Dabei handelt es sich um den *DISCO Network Calculator (DNC)*, *Computational Issues in Network Calculus (COINC)*, den *Real-Time Calculus (RTC) Toolbox* und *Cyclic Network Calculus (CyNC)*. Diese Anwendungen können dazu beitragen, die Worst-Case-Analyse für die Berechnung, Dimensionierung und Analyse eines Netzwerk-Szenarios einem breiteren Publikum näherzubringen.

Die vier Anwendungen können in zwei unterschiedliche Gruppen eingeordnet werden, was ihre Verwendungsmöglichkeiten und Umsetzung der mathematischen Konzepte angeht. Die erste Gruppe befasst sich mit einer möglichst mathematisch korrekten Umsetzung des Formelwerks und den dabei auftretenden numerischen Problemen. Die zweite Gruppe dagegen spezialisiert sich auf die Übertragung des Network Calculus auf eingebettete Systeme.

Die Software-Tools der letzten Gruppe sind eng verbunden mit der Performance-Analyse auf Systemebene von verteilten Echtzeit- und eingebetteten Systemen. Diese spezielle Anwendung zielt vor allem darauf ab, die direkte Anwendung des Network Calculus vor dem Benutzer zu verbergen und nur eine abstrakte Repräsentation der konkreten Berechnungen oder Kommunikationsressourcen zur Verfügung zu stellen.

Die folgenden Kapitel geben einen kurzen Überblick über die vier Software-Anwendungen und darüber, wie sie die mathematische Theorie des Network Calculus umsetzen, welche Vor- und Nachteile sie besitzen und auch die Unterschiede in deren Handhabung.

2.1 DISCO Network Calculator (DNC)

Die erste Anwendung, die hier vorgestellt wird, ist der *DISCO Network Calculator (DNC)* [DIS08]. Er steht als Open Source Java Bibliothek zur Verfügung und bietet eine Schnittstelle zu den mathematischen Network Calculus Verknüpfungen, sowie zusätzliche Funktionalität an [SZ06]. Diese zusätzliche Funktionalität beinhaltet unter anderem eine graphische Darstellung der Funktionen und verschiedene Analysemethoden, auf die weiter unten eingegangen wird. Zielsetzung ist, die Network Calculus Verknüpfungen zu automatisieren und die Analyse und Planung eines Netzwerks unter Worst-Case-Gesichtspunkten zu ermöglichen.

Die grundlegenden Verknüpfungen für die Min-Plus Algebra sind, bis auf den subadditiven Abschluss, alle umgesetzt. Die Funktionsmenge, mit der gearbeitet wird, beschränkt sich auf stückweise lineare konkave und konvexe Funktionen, was aber nicht notwendigerweise ein

Nachteil sein muss, da alle im Network Calculus gebräuchlichen Funktionen, durch stückweise lineare Funktionen angenähert werden können.

Zusätzlich zu der Anwendung der algebraischen Verknüpfungen auf die Funktionen können ganze Szenarien bezüglich der Delay und Backlog Bounds analysiert werden. Die Szenarien werden als Netzwerkgraph modelliert, auf dem der DNC dann arbeitet. Da der Network Calculus aber bekanntermaßen Probleme mit Flüssen hat, die bei der Berechnung voneinander abhängen, wurden zwei Lösungen implementiert, die diesen Nachteil ausgleichen. Zum einen handelt es sich um eine Methode, die von Charny und Le Boudec [CB00] entwickelt wurde, die mit diesen Abhängigkeiten umgehen kann, ohne das Szenario oder das zugrunde liegende Netzwerk in irgendeiner Form zu verändern. Der sogenannte *Charny-Bound* hängt von der Verkehrsauslastung und dem maximalen Hop-Count ab und hat leider ab einer bestimmten Auslastung mit explodierenden Grenzen zu kämpfen.

Die zweite, bessere Methode ist, die zugrunde liegende Netzwerk-Topologie auf ein *Feed-Forward* Netzwerk zu beschränken. Dazu wird ein graphentheoretischer Algorithmus verwendet, der das Routing in der allgemeinen Topologie übernimmt und somit zu einem Feed-Forward Netzwerk transformiert. Dieser Algorithmus nennt sich *Turn-Prohibition* [SKZ03]. Ihm liegt die Idee zugrunde, bestimmte Turns zwischen den Kanten eines Knotens zu verbieten und damit die Zyklen zu brechen.

Obwohl für die Berechnungen eine FIFO-Annahme getroffen werden kann, nimmt der DNC Abstand von der Implementierung spezifischer *Scheduling Policies*. Dies hat den Hintergrund, dass die eigentliche Scheduling Policy oberhalb der Aggregationsstufe nicht mehr genau auseinandergehalten werden kann. Der Grund liegt in einer möglichen Veränderung der Paketreihenfolge auf der Aggregationsstufe. Diese Ungewissheiten beim Multiplexing und einhergehend mit der Paketreihenfolge führen zur Annahme von beliebigen Schedulern, die Datenströme mittels *Blind Multiplexing* zusammenführen.

Um die Enge der berechneten Grenzen zu verbessern, wurden verschiedene Analysemethoden implementiert, die sich in der Art und Weise unterscheiden, wie die fundamentalen Network Calculus Eigenschaften und Verknüpfungen angewandt werden. Diese Analysen werden kurz aufgeführt, um eine Idee über deren Funktionsweise zu vermitteln.

Die *Fair-Queuing-Analyse (FQA)* nimmt getrennte Datenströme an und garantiert jedem einen fairen und gleichen Anteil am Service. Die *Total-Flow-Analyse (TFA)* basiert auf FIFO-Annahmen und addiert den Delay auf, den ein Datenstrom an jedem Server erhält. Eine etwas verfeinerte Analyse ist die *Separate-Flow-Analyse (SFA)* [KAT06], die das Verknüpfungstheorem ausnutzt. Hier wird zuerst der Service berechnet, der dem betrachteten Datenstrom an jedem Server zur Verfügung steht, wenn der Querverkehr Vorrang bekommt. Anschließend werden die einzelnen Service Curves mittels Konkatenation zusammengefasst und ergeben den Service, den der betrachtete Datenstrom erhält. Die *Pay-Multiplexing-Only-Once-Analyse (PMOO-SFA)* [SZM06] zeigt, wie die Verwendung des Verknüpfungstheorems und das Multiplexing von Datenströmen durchdachter und sorgfältiger ausgeführt werden können, in dem das Verknüpfungstheorem so früh wie möglich ausgeführt wird und der Preis des Multiplexings nur an den Eintrittsknoten des Querverkehrs und nicht auch noch an allen nachfolgenden Knoten gezahlt wird.

Diese Diplomarbeit beschäftigt sich mit der Erweiterung des DNC mit Algorithmen zum Berechnen von engen und optimierten Grenzen.

2.2 Computational Issues in Network Calculus (COINC)

Wie der Name der Anwendung schon vermuten lässt, wurden hier Algorithmen entwickelt, um die Network Calculus Verknüpfungen berechnen zu können. Diese Algorithmen sind in eine in C++ programmierte Anwendung eingeflossen [BCG$^+$08], die eine textbasierte Benutzerschnittstelle liefert. Die Ausgabe der Ergebnisse erfolgt in eine LaTeX Datei.

Der wissenschaftliche Hintergrund dieser Software lag im Bestreben, Algorithmen zu entwickeln, die sämtliche grundlegenden Network Calculus Verknüpfungen computergestützt berechnen kann [BET07]. Dies sind Minimum, Addition, Faltung, Entfaltung, Subadditive Hülle, sowie Maximum und Subtraktion. Dabei liegt das Hauptaugenmerk auf der Entwicklung möglichst mathematisch korrekter Algorithmen und der Suche nach einer möglichst effizienten Umsetzung dieser Algorithmen.

Die einzige Stelle, an der von der mathematischen Theorie abgewichen wird, ist die Menge der betrachteten Funktionen. Dies geschieht deshalb, um so allgemeine Algorithmen wie möglichst zu bekommen. Im Network Calculus werden Funktionen mit Wertebereich $\mathbb{R}_{min} = \mathbb{R} \cup \{+\infty\}$ betrachtet, wobei in COINC dies für diskrete Umgebungen folgendermaßen definiert wird: $f : \mathbb{N} \to \overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$, und für den kontinuierlichen Fall $f : \mathbb{R}^+ \to \overline{\mathbb{R}}$. Weiterhin wird eine Klasse von Funktionen definiert, die nicht notwendigerweise steigend sind, aber abgeschlossen bezüglich der elementaren Verknüpfungen. Dies sind die als *Ultimately Pseudo-Periodic* bezeichneten Funktionen, definiert durch $\exists T \in X = \mathbb{N}$ oder $\mathbb{R}^+, \exists (c,d) \in \mathbb{R} \times X \setminus \{0\}, \forall t > T, f(t+d) = f(t) + c$. Das bedeutet, dass eine Funktion ab einem bestimmten Punkt nur noch einen periodischen Teil besitzt. Dabei muss aber mit dem Problem gekämpft werden, dass sich nummerische Ungenauigkeiten auf den periodischen Anteil dahingehend auswirken, dass die Länge des periodischen Anteils leicht explodieren kann.

Neben diesen Unterschieden wird auch die Komplexität der entwickelten Algorithmen betrachtet. Hierbei ist besonders interessant, dass die Komplexität der subadditiven Hülle nicht genau quantifiziert und nur vermutet wird, dass es sich um exponentiellen Aufwand handelt. Grund dafür ist die Ausgabe bei wachsender Problemgröße und ein zugrunde liegendes NP-vollständiges Problem. Eine genauere Beschreibung der Algorithmen und deren Komplexität findet sich in [BET07].

Zurzeit wird an der nächsten Version dieser Anwendung gearbeitet, die als Bibliothek zur Einbindung in Scilab [INR08], einer Open Source Plattform für numerische Berechnungen, gedacht ist [BCG$^+$08].

Diese Zusammenfassung lässt die naheliegende Vermutung aufkommen, dass der primäre und wohl auch einzige Schwerpunkt in der Entwicklung von Algorithmen für die Network Calculus Verknüpfungen liegt. Die Möglichkeit ganze Szenarien zu simulieren und zu analysieren

wird als nebensächlich betrachtet und ist also darauf beschränkt, sämtliche Formeln dafür aufzuschreiben.

2.3 Real-Time Calculus (RTC) Toolbox

Wie anfangs erwähnt, gibt es zwei Programme die sich mit der Anwendung des Network Calculus auf eingebettete Systeme beschäftigen. Eines davon ist die *RTC Toolbox* [WT06]. Der Kern dieses Programms ist in Java implementiert, aber der Quellcode ist nicht öffentlich zugänglich, so dass über die mathematische Korrektheit und Komplexität der Implementierung nur spekuliert werden kann.

Die Abbildung des Network Calculus auf ein eingebettetes System verlangt eine Anpassung der Konzepte, die hier kurz angerissen werden. Diese Anpassung wurde in [TCN00] unter dem Namen *Real-Time Calculus* veröffentlicht. Insbesondere der Aspekt der Echtzeitanforderungen verlangt eine entsprechende Modellierung.

Die Datenströme werden als Ereignisstrom und nicht als stetigen Funktionen angenommen. Die Ereignisse stellen eine Treppenfunktion dar, mit Unstetigkeitsstellen an den Paket-Ankunftszeiten. Obwohl eine Treppenfunktion durch stückweise lineare Funktionen begrenzt werden kann, ist diese stärker mit dem Zeitverlauf der Datenströme in eingebetteten Systemen verknüpft.

Der nächste Unterschied liegt in der Modellierung einer eingebetteten Architektur. Dafür wird ein abstraktes *Service-Element* verwendet, das in Abbildung 2.1 beschrieben wird. Im Gegensatz zu dem elementaren Network Calculus wird ein Datenstrom bzw. Ereignisstrom R im Real-Time Calculus durch zwei Arrival Curves α_U und α_L sowohl nach oben, als auch nach unten beschränkt. Diese zusätzliche Beschränkung erlaubt es, Verkehr zu modellieren, der kontinuierlich oder periodisch gesendet wird, wie dies bei Statusmeldungen oder Sensordaten der Fall ist, oder der einen Jitter aufweist.

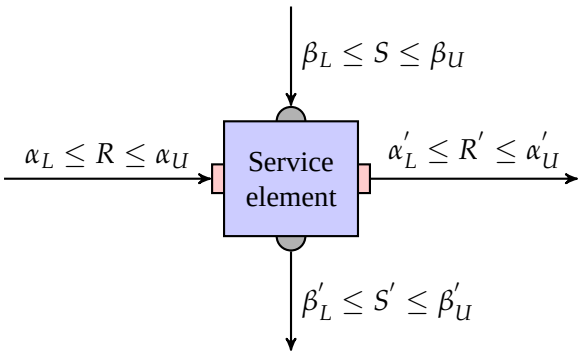


Abbildung 2.1: Real-Time Calculus service element [Per06].

Die im Service-Element S verfügbaren Ressourcen sind durch eine *Maximum* und *Minimum Service Curve* β_L, β_U wie gehabt begrenzt, aber nicht verbrauchte Ressourcen werden über

den Ausgang $S^{'}$ an den Eingang weiterer Service-Elemente übergeben. Mit diesem Vorgehen können verschiedene Scheduling Policies modelliert werden, wie *Preemptive Fixed Priority (FP), Generalised Processor Sharing (GPS), Time Division Multiple Access (TDMA)* und *Earliest Deadline First (EDF)* [Wan06]. Durch dieses veränderte Verhalten müssen die Ankunftsbedingungen wie folgt angepasst werden:

$$\alpha_L(t-s) \leq R(t) - R(s) \leq \alpha_U(t-s) \quad \forall s,t : 0 \leq s \leq t$$
$$\beta_L(t-s) \leq S(t) - S(s) \leq \beta_U(t-s) \quad \forall s,t : 0 \leq s \leq t$$

Daraus ergeben sich dann neue Formeln für die *Stream* und *Resource Output Bounds* [CKT03]:

$$\alpha_L^{'}(t) = \min\{\inf_{0\leq s\leq t}\{\sup_{\lambda\geq 0}\{\alpha_L(s+\lambda) - \beta_U(\lambda)\} + \beta_L(t-s)\}, \beta_L(t)\}$$
$$\alpha_U^{'}(t) = \min\{\sup_{\lambda\geq 0}\{\inf_{0\leq s\leq \lambda+t}\{\alpha_U(s) + \beta_U(\lambda+t-s)\} - \beta_L(\lambda)\}, \beta_U(t)\}$$
$$\beta_L^{'}(t) = \sup_{0\leq\lambda\leq t}\{\beta_L(\lambda) - \alpha_U(\lambda)\}$$
$$\beta_U^{'}(t) = \max\{\inf_{\lambda\geq t}\{\beta_U(\lambda) - \alpha_L(\lambda)\}, 0\}$$

Die Toolbox ist für die Anwendung in MATLAB [Mat08] gedacht und besitzt, neben einer Schnittstelle für die Anbindung des Java-Kerns, eine Bibliothek für eine *Modulare Performance Analyse (MPA)* [Wan06]. Für die Modellierung eines eingebetteten Systems ist das Schreiben eines MATLAB-Programms notwendig, da momentan noch keine andere graphische Eingabemöglichkeit besteht. Eine Einbettung in Simulink [Sim08] ist in Planung, dadurch würde dem Benutzer erlaubt werden, Systeme mit Hilfe graphischer Blöcke zu erstellen, zu simulieren und zu berechnen [Wan06].

Zurzeit wird der Anwender beim Erstellen des Modells durch verschiedene Hardware-Komponenten unterstützt, die schon als Block-Modelle vorliegen. Diese Blöcke kapseln die Real-Time Calculus Verknüpfungen und können so zur modularen Performance-Analyse von komplexeren Queuing-Networks eingesetzt werden.

Eine genauere Betrachtung der Real-Time Calculus Verknüpfungen zeigt, dass die meisten Min-Plus- und Max-Plus-Verknüpfungen implementiert sind, bis auf die subadditive Hülle. Innerhalb des Java-Kerns werden *Variability Characterization Curves (VCC)* verwendet. Dieses Konzept wurde für die bessere Handhabung von Datenströmen, mit einer Variabilität in der Ausführungszeit oder in der Eingabe-Ausgabe Geschwindigkeit, entwickelt [MZCW04]. Dies ist zum Beispiel in der Dekodierung von MPEG-2-Daten variabler Länge der Fall.

Eine VCC v ist ein Tupel $(v_L(t), v_U(t))$, wobei $v_L(t)$ eine untere und $v_U(t)$ eine obere Schranke für alle Kurven vorsieht, d.h. Funktionen die sowohl stückweise lineare und nicht stückweise lineare Funktionen beinhalten. Aber nur Funktionen mit einer endlichen Darstellung können behandelt werden, also Funktionen, die ab einem bestimmten Punkt ein periodisches Verhalten besitzen. Weiterhin ist ihr Definitionsbereich auf $\mathbb{R}^+$ beschränkt, sie müssen aber weder notwendigerweise steigend noch positiv sein [BET08].

Diese Anwendung ist auch eine der Performance-Analysemethoden für verteilte, eingebettete Systeme, die Perathoner in seiner Masterarbeit verglichen und bewertet hat [Per06].

2.4 Cyclic Network Calculus (CyNC)

Die letzte hier vorgestellte Anwendung, die das Konzept des Network Calculus auf Echtzeit-Systeme anwendet, ist der *Cyclic Network Calculus (CyNC)* [Sch08, SSH07]. Er ist vor allem für eingebettete Systeme mit komplexen Prozessabhängigkeiten gedacht. Grundlage für die Programmierung und Verwendung sind MATLAB [Mat08] und Simulink [Sim08].

CyNC bietet eine Bibliothek mit verschiedenen Blöcken, die eine einfache Verwendung für das graphische Modellieren und Analysieren von Systemen möglich machen. Im Gegensatz zu den bisherigen Anwendungen wird vom Benutzer keine Implementierung in irgendeiner Programmiersprache verlangt, sondern nur das graphische modellieren mit vorgefertigten Bausteinen.

Um dem Namen Cyclic Network Calculus gerecht zu werden, stellen Schiøler u.a. in [SNLJ05] eine Möglichkeit vor, auch Netzwerke mit zyklischen Abhängigkeiten unter den Datenflüssen zu berechnen. Dies scheint der wesentliche Unterschied zu dem Real-Time-Calculus zu sein, denn auch hier wird das in Abbildung 2.1 gezeigte Service-Element als Grundlage verwendet.

Um die Zyklen aufzulösen, wird ein Fixpunktproblem mit einem iterativen Verfahren gelöst. Ein Nachteil dieser Methode ist, dass die Komplexität und Korrektheit der zugrunde liegenden numerischen Iteration und linearen Zerlegung nicht ausführlich untersucht wurde [SNLJ05, BET08]. Auch die Untersuchung der Stabilität und der Stabilitätsgrenzen wurde für zukünftige Forschung offen gelassen [SNLJ05].

Die zur Verfügung stehenden Blöcke, die in [SSH07] beschrieben sind, können in drei Hauptgruppen unterteilt werden: Generatoren, Netzwerk-Elemente und zusätzliche Werkzeuge. Die Generatoren bieten die Möglichkeit Bedingungen für die Datenströme und Ressourcen festzulegen.

Die Netzwerk-Elemente sind insgesamt sechs verschiedene Blöcke, zum einen das schon bekannte Service-Element, sowie fünf andere Schedulingverfahren um Ressourcen zu verteilen und darauf zuzugreifen. Dies sind im Besonderen *Fixed Priority (FP)*, *Round Robin (RR)*, *FIFO*, *Time Division Multiple Access (TDMA)* und *Earlies Deadline First (EDF)*.

Die letzte Gruppe beinhaltet Blöcke für verschiedene andere Effekte. So finden sich hier Blöcke um die Effekte von Bursts durch *Packetising* und *Scaling*, oder auch *Flow-Convergence* zu modellieren. Weiterhin gibt es einen Block für die *Inf-Plus-Faltung* um Service-Elemente in Reihe zu schalten. Zu guter Letzt wurde auch an die Berechnung von Delay und Backlog Bound gedacht, sowie an die graphische Darstellung der Ergebnisse.

Zwar wurde die Funktionalität der aktuellen Version schon in [SSH07] beschrieben, bislang ist aber leider nur die vorherige Version unter [Sch08] öffentlich zugänglich.

Die von CyNC verwendeten Funktionen sind über $\mathbb{R}^+$ definiert und sind Treppenfunktionen bis zu einem bestimmten Punkt, ab dem sie affin sind.

3 Optimisation-Based Bounding Method

Momentan gibt es verschiedene Analysemethoden, den Network Calculus auf Feed-Forward Netzwerke anzuwenden, wenn *Arbitrary Multiplexing* angenommen wird. Das bedeutet, dass alle anderen Datenströme gegenüber dem betrachteten vorrangig behandelt werden. Jede dieser Methoden verfolgt unterschiedliche Ansätze, die vom Network Calculus definierten Verknüpfungen anzuwenden. Die verfügbaren Analysen geben verschiedene Möglichkeiten, die Bounds zu verbessern.

Die Unterschiede treten bei den verwendeten Verknüpfungen, als auch bei der Reihenfolge der Anwendung auf. Eine kurze Übersicht wurde schon in Kapitel 2.1 gegeben. Auf zwei dieser Methoden wird im Folgenden noch etwas genauer eingegangen, da sie für die Entwicklung einer neuen Methode zum Berechnen eines *Tight Bounds* wichtig sind und auch als Vergleichsbasis für die Ergebnisse dienen.

Das folgende Unterkapitel fasst die wichtigsten der bisherigen Ansätze kurz zusammen und schildert den Grund, den der Network Calculus mit Tight Bounds besitzt. Darauf aufbauend beschreiben die weiteren Unterkapitel die von Schmitt u.a. neu entwickelte *Optimisation-Based Bounding Methode* [SZF07].

3.1 Bisherige Ansätze

Ein wichtiger Punkt ist, dass die hier vorgestellte Methode zur Berechnung von Tight oder optimierten Bounds *Strict Service Curves* voraussetzt.

Definition 1 (Strict Service Curve)**.** *Sei $\beta \in \mathcal{F}$. Ein System $\mathcal{S}$ garantiert einem Datenstrom eine Strict Service Curve β, wenn während jeder Backlogged-Period der Zeitdauer u der Ausgang des Datenstroms mindestens gleich $\beta(u)$ ist.*

Jede Strict Service Curve ist also auch eine Service Curve, aber nicht umgekehrt. Die Service Curves vieler Scheduler sind strikt. Ein Beispiel dafür ist der Scheduler, der mit Hilfe von Rate-Latency-Funktionen *Generalized Processor Sharing* nachbildet. Strict Service Curves spielen bei der Entwicklung von Tight Bounds eine entscheidende Rolle, weil sie es ermöglichen, die maximale Backlogged-Period des Systems zu beschränken. Diese Schranke ist gerade der Schnittpunkt $\overline{d}$ zwischen Arrival und Service Curve, also $\alpha(\overline{d}) = \beta(\overline{d})$. Deshalb

sind Burst-Delay-Funktionen problematisch, da sie die Eigenschaft der Striktheit nicht erfüllen. Ohne die Erfüllung der Striktheit kann die Vorgehensweise zwar angewandt werden, die Ergebnisse sind aber nicht aussagekräftig.

Die Gemeinsamkeit der einzelnen Methoden besteht darin, dass die netzinternen Beschränkungen jedes einzelnen Datenstroms des Querverkehrs berechnet werden. Für Feed-Forward Netzwerke muss dafür einfach nur der Output Bound aus Theorem 1 angewandt werden, wobei sich das Multiplexing der Datenströme aus deren Addition ergibt.

Theorem 1 (Performance Bounds)**.** *Gegeben sei ein System $\mathcal{S}$, dass eine Service Curve β besitzt. Angenommen, ein Datenstrom F, der durch eine Arrival Curve α begrenzt wird, durchläuft das System. Dann können folgende Performance Bounds berechnet werden:*
Backlog: $\forall t : b(t) \leq (\alpha \oslash \beta)(0) =: v(\alpha, \beta)$
Delay: $\forall t : d(t) \leq \inf\{t \geq 0 : (\alpha \oslash \beta)(-t) \leq 0\} =: h(\alpha, \beta)$
Output (Arrival Curve α' für F'): $\alpha' = \alpha \oslash \beta$

Zur Erläuterung werden die Methoden an dem Beispiel in Abbildung 3.1 erklärt. An diesem Beispiel können schon die Ursachen der Probleme für die Bestimmung von Tight Bounds verdeutlicht werden, die die bisherigen Methoden haben. Im Folgenden sind die Arrival Curves durch Token-Buckets $\alpha_i = \gamma_{r_i,b_i}$ beschränkt und für den garantierten Service sind die Rate-Latencies $\beta_i = \beta_{R_i,T_i}$, für $i = 1, 2$, soweit nicht anders angegeben

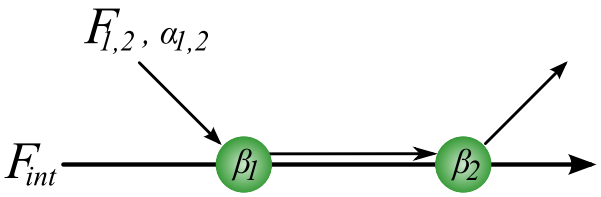


Abbildung 3.1: Simple two nodes, two flows scenario.

Separated Flow Analysis (SFA)

Der betrachtete Datenstrom wird getrennt von dem Querverkehr behandelt, wie schon der Name der Methode andeutet.

Theorem 2 (Left-Over Service Curve unter Arbitrary Multiplexing)**.** *Gegeben sei ein Knoten, der zwei Datenströme 1 und 2 in beliebiger Reihenfolge multiplext. Angenommen, der Knoten garantiert der Vereinigung der beiden Ströme eine Strict Minimum Service Curve β und Strom 2 wird durch eine Arrival Curve α_2 begrenzt. Dann ist*

$$\beta_1(t) := [\beta(t) - \alpha_2(t)]^+$$

eine Service Curve für Datenstrom 1, wenn $\beta_1 \in \mathcal{F}$. Man nennt β_1 auch die Left-Over Service Curve für den betrachteten Datenstrom.

Die Separierung vom Querverkehr ist sinnvoll, weil dadurch der Left-Over Service an jedem einzelnen Knoten für den betrachteten Datenstorm nun mit dem nachfolgenden Theorem 2 berechnet werden kann. Der Querverkehr wird also aufsummiert und gegenüber dem betrachteten Datenstrom vorrangig behandelt. Die Striktheit der Service Curve ist für Theorem 2 notwendig, wie das Beispiel aus [BT01] zeigt.

Die einzelnen Left-Over Service Curves, die sich dann für jeden Knoten auf dem Weg des betrachteten Datenstroms ergeben, werden dann mit Hilfe des Theorems 3 zu einer Service Curve zusammengefasst.

Theorem 3 (Verknüpfungstheorem für Tandemsysteme)**.** *Gegeben sei ein Datenstrom, der nacheinander zwei Systeme $\mathcal{S}_1$ und $\mathcal{S}_1$ durchläuft. Angenommen, $\mathcal{S}_i$ garantiert dem Datenstrom eine Service Curve β_i, $i = 1, 2$. Dann garantiert die Verknüpfung der beiden Systeme dem Datenstrom die Service Curve $\beta_1 \otimes \beta_2$.*

Die Anwendung der Theoreme auf das Beispiel in Abbildung 3.1 ergibt für den Delay Bound folgende Berechnung:

$$\begin{aligned} d^{\mathtt{SFA}} &= h\left(\alpha_1, [\beta_1 - \alpha_2]^+ \otimes [\beta_2 - (\alpha_2 \oslash \beta_1)]^+\right) \\ &= T_1 + T_2 + \frac{b_1}{(R_1 \wedge R_2) - r_2} + \frac{b_2 + r_2 T_1}{R_1 - r_2} + \frac{b_2 + r_2(T_1 + T_2)}{R_2 - r_2} \end{aligned} \tag{3.1}$$

An dieser Stelle ist anzumerken, dass die horizontale Abweichung für die Berechnung des End-to-End Delay Bounds benutzt werden kann, da für jeden Datenstrom eine FIFO-Reihenfolge angenommen wird (*FIFO-per-microflow*).

Daraus ergibt sich eine weitere Beobachtung. Die Bounds können nicht Tight sein, da durch die Erhöhung der Burstiness um $r_2 T_1$ durch den Term $\alpha_2 \oslash \beta_1$ in Formel 3.1, die Erhöhung der Burstiness nur einen theoretischen Verlauf des Datenstroms beschreibt. Der Service an Knoten 1 müsste zuerst bis T_1 verzögern und dann unendlich schnell die ankommenden Daten bearbeiten.

Diese Beobachtung wird auch dadurch bestätigt, dass die Burstiness b_2 in der Berechnung für $d^{\mathtt{SFA}}$ zweimal auftaucht und der Preis des Multiplexings mit dem Querverkehr zweimal bezahlt wird. Allerdings „sieht“ der Knoten 2 ja nur den gemeinsamen Datenstrom und der Querverkehr hat keine Möglichkeit, den betrachteten Datenstrom erneut zu überholen.

Pay Multiplexing Only Once SFA (PMOO-SFA)

Wie eben schon bemerkt, fallen die Kosten des Multiplexing an jedem Knoten an. Diesem Nachteil wird bei der PMOO-SFA dadurch begegnet, indem die Reihenfolge der Verknüpfungen geschickt vertauscht wird. Die grundlegende Idee ist, das Verknüpfungstheorem so früh wie möglich anzuwenden, um so die Kosten des Multiplexing nur einmal bezahlen zu müssen, nämlich am Eintrittsknoten des jeweiligen Querverkehrs. Die nachfolgenden Knoten sehen dann also nur noch den gemeinsamen Datenstrom.

Bezogen auf das Beispiel in Abbildung 3.1 bedeutet dies, dass zuerst die Faltung des Verknüpfungstheorems auf die beiden Knoten angewandt wird und anschließend das Multiplexing auf den daraus resultierenden Knoten. Oder in Formeln ausgedrückt:

$$\begin{aligned} d^{\text{PMOO}} &= h\left(\alpha_1, [(\beta_1 \otimes \beta_2 - \alpha_2)]^+\right) \\ &= T_1 + T_2 + \frac{b_1 + b_2 + r_2(T_1 + T_2)}{(R_1 \wedge R_2) - r_2} \end{aligned} \tag{3.2}$$

Auf den ersten Blick scheint die PMOO-SFA bessere Bounds liefern zu können, da die Formel 3.2 eine leicht nachvollziehbare Form besitzt, in der die Latencies T_1, T_2 der Knoten aufsummiert werden und die Burst Terme b_1, b_2, sowie die Erhöhung der Burstiness $r_2(T_1 + T_2)$ an der geringeren Service Rate der Knoten anfallen. Jedoch offenbart eine genauere Betrachtung ein überraschendes Ergebnis: die SFA kann nämlich unter Umständen bessere Schranken als die PMOO-SFA liefern. Dieser Spezialfall tritt auf, wenn die Parameter $b_2 = 0$ und $T_1 = 0$ gesetzt werden. Dann verkürzen sich die Formeln zu

$$\begin{aligned} d^{\text{SFA}} &= T_2 + \frac{b_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{R_2 - r_2} \\ d^{\text{PMOO}} &= T_2 + \frac{b_1 + r_2 T_2}{(R_1 \wedge R_2) - r_2} \end{aligned}$$

und für die Differenz ergibt sich nun

$$d^{\text{PMOO}} - d^{\text{SFA}} = r_2 T_2 \left(\frac{1}{(R_1 \wedge R_2) - r_2} - \frac{1}{R_2 - r_2} \right) \geq 0$$

Durch Veränderung der Rate des zweiten Servers R_2 kann diese Differenz nun beliebig groß gemacht werden. Obwohl dieser Spezialfall vielleicht eher die Ausnahme ist, zeigt sich doch deutlich, dass die Parameterwahl einen entscheidenden Einfluss auf das Ergebnis hat. So würde die Wahl von $T_2 = 0$ dazu führen, dass die PMOO-SFA auf jeden Fall bessere Bounds liefert. Mit diesen Beispielen zeigt sich, dass die PMOO-SFA keine Tight Bounds liefern kann.

Die Frage, die sich jetzt stellt ist, wo die Ursachen hierfür liegen. Die SFA entspricht keinem realistischen zeitlichen Verlauf des Datenflusses, die PMOO-SFA dagegen ist abhängig von der Parameterwahl, wie in dem harmlos aussehenden Beispiel weiter oben gezeigt wurde. Bei genauerer Betrachtung der PMOO-SFA ergibt sich aus Formel 3.2 die Schlussfolgerung, dass die Erhöhung der Burstiness von Fluss 2 aufgrund der Latency des Knotens 2 $r_2 T_2$ vom Minimum der Raten R_1 und R_2 verarbeitet wird. Gerade dies führt zu dem Kernproblem, das die PMOO-SFA hat. Eine Erhöhung jener Burstiness kann nur an dem Knoten 2 anfallen und nur von diesem und möglichen nachfolgenden Knoten verarbeitet werden, aber niemals an Knoten 1. Gerade diese physikalische Tatsache kann die PMOO-SFA nicht abbilden, die Kommutativität der Faltung verhindert dies erfolgreich und wichtige topologische Informationen werden einfach verschluckt.

Aus algebraischer Sicht des Network Calculus ist die Reihenfolge der Operanden für die Min-Plus-Faltung egal, da Kommutativität Eigenschaft eines Dioids ist. Für reale Datenströme ist

die Reihenfolge dagegen nicht gleichgültig. Hier zeigt sich, dass der Network Calculus von Anfang an schon eine Eigenschaft besitzt, die bei der Anwendung auf reale Szenarien Probleme bereitet und entsprechend Kenntnis desselben, sowie Fingerspitzengefühl bei der Modellierung eines konkreten Systems erfordert. Deswegen werden alle Multiplexing-Verfahren, die nicht explizit FIFO voraussetzen, das gleiche Problem mit sich bringen, selbst wenn mehr Informationen gegeben sind, als durch Arbitrary Multiplexing vorausgesetzt werden.

Dies kann aus physikalischer Sicht folgendermaßen erklärt werden. Ohne die FIFO-Annahme ist es möglich, dass an einem Knoten erstmals hinzukommenden Daten des Querverkehrs gegenüber dem betrachteten Datenstrom vorrangig behandelt werden. Diese „Überholung“ führt zu einer Erhöhung der Burstiness des Querverkehrs, die frühestens an dem Knoten anfallen kann, an dem der betrachtete Datenstrom überholt wurde.

3.2 Ein neuer Ansatz

Im vorigen Kapitel wurde auf das Problem eingegangen, das der Network Calculus schon aufgrund seiner algebraischen Struktur mit sich bringt – nämlich, dass die Kommutativität der Faltung zu einem gravierenden Informationsverlust führt – und somit die bisherigen Ansätze, einen Tight Bound zu liefern, ohne eine FIFO-Annahme scheitern. Die weiteren Kapitel beschäftigen sich mit der neuen *Optimisation-Based Bounding Methode*, durch die dieser Informationsverlust ausgeglichen wird und dadurch unter bestimmten Voraussetzungen einen Tight Bound liefern kann.

Die verlorengegangenen Informationen über die Burstiness, die ein Datenstrom auf dem Weg von einem zum nächsten Knoten mitnimmt werden durch zusätzliche Variablen gespeichert. Mit der Kenntnis über die Ankunftsbedingungen der Datenströme und den Service Garantien der einzelnen Knoten kann dann ein Optimierungsproblem formuliert werden.

Im weiteren Verlauf des Kapitels wird diese neue Methode anhand eines Beispiels erläutert. Ein geeignetes Beispiel, das alle wesentlichen Probleme in sich vereinigt, ist das Szenario in Abbildung 3.2. Dafür werden die Formeln für allgemeine Arrival und Service Curves aufgestellt, und später dann im Speziellen für Token-Bucket beschränkte Datenströme und Rate-Latency Server gelöst.

In den nachfolgenden Kapiteln werden dann die komplexeren Formeln für eine allgemeinere Netzwerkdarstellung präsentiert, sowie auf stückweise lineare konkave Arrival und konvexe Service Curves verallgemeinert werden.

Das Szenario ist wie in Abbildung 3.2 angelegt. Der Service jedes Knotens wird garantiert durch eine Strict Service Curve β_k, $k = 1, 2, 3$.

Sei $0 \leq t_0 \leq t_1 \leq t_2 \leq t_3$, so dass t_{k-1} der Start der letzten Backlogged-Period am Knoten k, bevor t_k. Aufgrund der Forderung, dass die Service Curve strikt sein muss, lässt sich aus der Eigenschaft der Service Curve für jeden Knoten $k = 1, 2, 3$ die folgende Formel aufschreiben

$$F^{(k)}(t_k) - F^{(k-1)}(t_{k-1}) \geq \beta_k(t_k - t_{k-1})$$

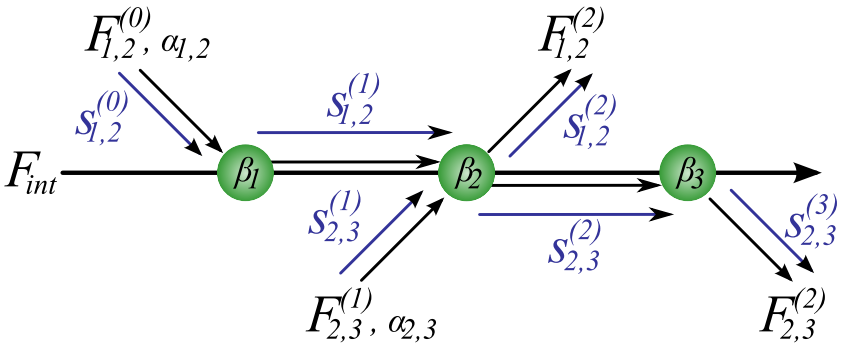


Abbildung 3.2: Overlapping interference scenario.

wobei $F^{(k)}$ den gesamten Datenstrom angibt, der an Knoten $k+1$ ankommt.

Wegen der Monotonie der Eingangs- und Ausgangsfunktionen und der Beziehung $F_i^{(k)}(t_k) \geq F_i^{(k)}(t_{k-1}) = F_i^{(k-1)}(t_{k-1})$ für die jeweiligen t_k können sich mit der sich daraus ergebenden Ungleichung $F_i^{(k)}(t_k) - F_i^{(k-1)}(t_{k-1}) \geq 0$ folgende Formeln herleiten lassen:

$$
\begin{aligned}
F_1^{(1)}(t_1) - F_1^{(0)}(t_0) &\geq \left[\beta_1(t_1 - t_0) - \left(F_2^{(1)}(t_1) - F_2^{(0)}(t_0)\right)\right]^+ \\
F_1^{(2)}(t_2) - F_1^{(1)}(t_1) &\geq \left[\beta_2(t_2 - t_1) - \left(F_3^{(2)}(t_2) - F_3^{(1)}(t_1)\right) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+ \\
F_1^{(3)}(t_3) - F_1^{(2)}(t_2) &\geq \left[\beta_3(t_3 - t_2) - \left(F_3^{(3)}(t_3) - F_3^{(2)}(t_2)\right)\right]^+
\end{aligned}
$$

Aufaddiert ergeben diese Formeln

$$
\begin{aligned}
F_1^{(3)}(t_3) - F_1^{(0)}(t_0) \geq &\left[\beta_1(t_1 - t_0) - \left(F_2^{(1)}(t_1) - F_2^{(0)}(t_0)\right)\right]^+ \\
&+ \left[\beta_2(t_2 - t_1) - \left(F_3^{(2)}(t_2) - F_3^{(1)}(t_1)\right) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+ \\
&+ \left[\beta_3(t_3 - t_2) - \left(F_3^{(3)}(t_3) - F_3^{(2)}(t_2)\right)\right]^+
\end{aligned}
$$

Aufgrund der Ankunftsbedingungen ergibt sich

$$
\begin{aligned}
F_2^{(1)}(t_1) - F_2^{(0)}(t_0) \leq F_2^{(0)}(t_1) - F_2^{(0)}(t_0) \leq \alpha_2(t_1 - t_0) \\
F_2^{(2)}(t_2) - F_2^{(0)}(t_0) \leq F_2^{(0)}(t_2) - F_2^{(0)}(t_0) \leq \alpha_2(t_2 - t_0) \\
F_3^{(2)}(t_2) - F_3^{(1)}(t_1) \leq F_3^{(1)}(t_2) - F_3^{(1)}(t_1) \leq \alpha_3(t_2 - t_1) \\
F_3^{(3)}(t_3) - F_3^{(1)}(t_1) \leq F_3^{(1)}(t_3) - F_3^{(1)}(t_1) \leq \alpha_3(t_3 - t_1)
\end{aligned}
$$

An diesem Punkt erfolgt der entscheidende Schritt: die Einführung der Schlupfvariablen. Da-

mit kann die obige Ungleichung umgeformt werden.

$$\begin{aligned}
F_2^{(1)}(t_1) - F_2^{(0)}(t_0) &= \alpha_2(t_1 - t_0) - s_2^{(1)} && s_2^{(1)} \geq 0 \\
F_2^{(2)}(t_2) - F_2^{(1)}(t_1) &\leq \alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)} \\
F_3^{(2)}(t_2) - F_3^{(1)}(t_1) &= \alpha_3(t_2 - t_1) - s_3^{(2)} && s_3^{(2)} \geq 0 \\
F_3^{(3)}(t_3) - F_3^{(2)}(t_2) &\leq \alpha_3(t_3 - t_1) - \alpha_3(t_2 - t_1) + s_3^{(2)}
\end{aligned}$$

Der Sinn und Zweck der Schlupfvariablen $s_i^{(k)}$ ist es, das Wissen über die übertragene Burstiness wieder mit in die Berechnungen einfließen zu lassen. Schließlich geht es bei dieser neuen Methode darum herauszufinden, an welchem Knoten die Burstiness eines Datenstroms erhöht wird. Die Schlupfvariablen geben an, wie viel der angesammelten Burstiness des Datenstroms i vom Knoten k an den folgenden weitergeleitet wird. Damit sind diese gerade die Variablen der Zielfunktion, also die Entscheidungsvariablen, des zu lösenden Optimierungsproblems.

Des Weiteren werden Nebenbedingungen für die Schlupfvariablen angegeben. Diese lassen sich in zwei Gruppen unterteilen. Zuerst die Vorzeichenbedingungen $s_i^{(k)} \geq 0$, die sich sehr leicht dadurch erklären lassen, dass die weitergegebene Burstiness negativ werden kann. Zusätzlich können noch funktionale Nebenbedingungen hergeleitet werden, die den Schlupf nach oben begrenzen. Auch hier kann abgeschätzt werden, wie viel Burstiness an einem Knoten ankommt, wie viel verbraucht wird und wieviel maximal hinzukommt. Dazu werden die Service Curve Garantien betrachtet. Für den Datenstrom 2 ist dies

$$\alpha_2(t_1 - t_0) - s_2^{(1)} = F_2^{(1)}(t_1) - F_2^{(0)}(t_0) \geq \beta_1(t_1 - t_0)$$

aus der sich dann durch Umformung

$$\begin{aligned}
s_2^{(1)} &\leq \alpha_2(t_1 - t_0) - \beta_1(t_1 - t_0) \\
&\leq \sup_{t_1 - t_0 \geq 0} \{\alpha_2(t_1 - t_0) - \beta_1(t_1 - t_0)\} =: B_2^{(1)}
\end{aligned}$$

ergibt. Für den Datenstrom 3 ergibt sich die komplexere Ungleichung

$$\begin{aligned}
\alpha_3(t_2 - t_1) - s_3^{(2)} &= F_3^{(2)}(t_2) - F_3^{(1)}(t_1) \\
&\geq \left[\beta_2(t_2 - t_1) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+ \\
&\geq \left[\beta_2(t_2 - t_1 - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+
\end{aligned}$$

so dass sich nach Umformung die folgende obere Schranke ergibt

$$\begin{aligned}
s_3^{(2)} &\leq \alpha_3(t_2 - t_1) - \left[\beta_2(t_2 - t_1) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+ \\
&\leq \sup_{t_2 - t_1 \geq 0} \left\{\alpha_3(t_2 - t_1) - \left[\beta_2(t_2 - t_1) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+\right\} =: B_3^{(2)}
\end{aligned}$$

Hierbei ist es wichtig, dem Datenstrom 2 eine Strict Priorität gegenüber Datenstrom 3 einzuräumen. Das ist auf die Beobachtung zurückzuführen, dass dies den Datenstrom 3 so bursty wie möglich macht. Für den betrachteten Datenstrom stellt dies den ungünstigsten Fall dar, denn er wird vom Datenstrom 3 noch bis zum Knoten 3 begleitet, während Datenstrom 2 schon am Knoten 2 abgeht. Diese Beobachtung lässt sich dahingehend verallgemeinern, dass beim Scheduling zwischen dem Querverkehr immer demjenigen Datenstrom die höhere Priorität einzuräumen ist, der den betrachteten Datenstrom am ehesten verlässt.

Alle bisherigen Formeln zusammengenommen, ergibt sich nun unter den beiden Bedingungen $\mathcal{C}_1 = (0 \leq t_0 \leq t_1 \leq t_2 \leq t_3)$ und $\mathcal{C}_2 = \left(0 \leq s_2^{(1)} \leq B_2^{(1)}, 0 \leq s_3^{(2)} \leq B_3^{(2)}\right)$ die folgende Ungleichung:

$$
\begin{aligned}
& F_1^{(3)}(t_3) - F_1^{(0)}(t_0) \geq \beta^1(t_3 - t_0) \qquad (3.3)\\
& = \inf_{\mathcal{C}_1,\mathcal{C}_2} \Bigg\{ \left[\beta_3(t_3 - t_2) - \left(\alpha_3(t_3 - t_1) - \alpha_3(t_2 - t_1) + s_3^{(2)}\right)\right]^+ \\
& \qquad + \left[\beta_2(t_2 - t_1) - \left(\alpha_3(t_2 - t_1) - s_3^{(2)}\right) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+ \\
& \qquad + \left[\beta_1(t_1 - t_0) - \left(\alpha_2(t_1 - t_0) - s_2^{(1)}\right)\right]^+ \Bigg\}
\end{aligned}
$$

Wie an der Formel 3.3 zu sehen ist, muss für die Berechnung der Left-Over Service Curve für Datenstrom 1 zunächst das Optimierungsproblem gelöst werden, um dann über Theorem 1 den Delay Bound zu bestimmen. Unter allgemeinen Annahmen kann das Optimierungsproblem nicht gelöst werden. Deshalb wird die Annahme getroffen, dass die Arrival Curves durch $\alpha_i = \gamma_{r_i,b_i}$ und die Service Curves durch $\beta_i = \beta_{R_i,T_i}$ beschränkt werden. Für die oberen Schranken der Schlupfvariablen ergeben sich:

$$
\begin{aligned}
B_2^{(1)} &= b_2 + r_2 T_1 \\
B_3^{(2)} &= b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2}\right)
\end{aligned}
$$

und für die Left-Over Service Curve des Datenstroms 1:

$$
\begin{aligned}
& \beta^1(t_3 - t_0) \\
& \qquad = \inf_{\mathcal{C}_1,\mathcal{C}_2} \Bigg\{ \left[R_3\left[t_3 - t_2 - T_3\right]^+ - \left(r_3(t_3 - t_2) + s_3^{(2)}\right)\right]^+ \\
& \qquad\quad + \left[R_2\left[t_2 - t_1 - T_2\right]^+ - \left(b_3 + r_3(t_2 - t_1) - s_3^{(2)}\right) - \left(r_2(t_2 - t_1) + s_2^{(1)}\right)\right]^+ \\
& \qquad\quad + \left[R_1\left[t_1 - t_0 - T_1\right]^+ - \left(b_2 + r_2(t_1 - t_0) - s_2^{(1)}\right)\right]^+ \Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
&\geq \inf_{\mathcal{C}_1,\mathcal{C}_2} \Bigg\{ (R_3 - r_3) \left[t_3 - t_2 - T_3 - \frac{r_3 T_3 + s_3^{(2)}}{R_3 - r_3} \right]^+ \\
&\quad + (R_2 - r_2 - r_3) \left[t_2 - t_1 - T_2 - \frac{b_3 + (r_2 + r_3) T_2 - s_3^{(2)} + s_2^{(1)}}{R_2 - r_2 - r_3} \right]^+ \\
&\quad + (R_1 - r_2) \left[t_1 - t_0 - T_1 - \frac{b_2 + r_2 T_1 - s_2^{(1)}}{R_1 - r_2} \right]^+ \Bigg\} \\
&= ((R_1 - r_2) \wedge (R_2 - r_2 - r_3) \wedge (R_3 - r_3)) \times \\
&\quad \inf_{\mathcal{C}_2} \Bigg\{ \Bigg[t_3 - t_0 - (T_1 + T_2 + T_3) - \frac{r_3 T_3 + s_3^{(2)}}{R_3 - r_3} \\
&\quad - \frac{b_3 + (r_2 + r_3) T_2 - s_3^{(2)} + s_2^{(1)}}{R_2 - r_2 - r_3} - \frac{b_2 + r_2 T_1 - s_2^{(1)}}{R_1 - r_2} \Bigg]^+ \Bigg\}
\end{aligned}
$$

Um eine geschlossene Formel für die Left-Over Service Curve zu finden, muss ein lineares Programm gelöst werden. Genauer gesagt, die Zielfunktion muss minimiert werden. Die Koeffizienten der Zielfunktion des linearen Programms lassen sich aus obiger Formel herauslesen. Die funktionellen und Vorzeichenbedingungen wurden schon weiter oben angegeben.

$$
\begin{aligned}
&\text{min.} \left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3} \right) s_2^{(1)} + \left(\frac{1}{R_2 - r_2 - r_3} - \frac{1}{R_3 - r_3} \right) s_3^{(2)} \\
&\text{s.t. } 0 \leq s_2^{(1)} \leq b_2 + r_2 T_1 \\
&\qquad 0 \leq s_3^{(2)} \leq b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2} \right)
\end{aligned}
$$

Dieses einfache lineare Programm lässt sich auch graphisch darstellen. Abbildung 3.3 zeigt den Bereich mit der Menge der zulässigen Lösungen, d.h. die Menge der Punkte, die alle Nebenbedingungen erfüllen. Interessant ist hier, dass es immer eine optimale Lösung gibt, die ein Eckpunkt des zulässigen Bereichs ist [HK06].

Mit den gegebenen Parametern liefert die Zielfunktion unter den Nebenbedingungen eine konkrete Lösung. Eine allgemeine Lösung für dieses spezielle Problem beruht auf dem Verhältnis zwischen den verbleibenden Service Raten der Knoten. Mit den Koeffizienten $A := \frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3}$, $B := \frac{1}{R_2 - r_2 - r_3} - \frac{1}{R_3 - r_3}$ und $C := \frac{r_3}{R_2 - r_2}$ der Schlupfvariablen aus dem

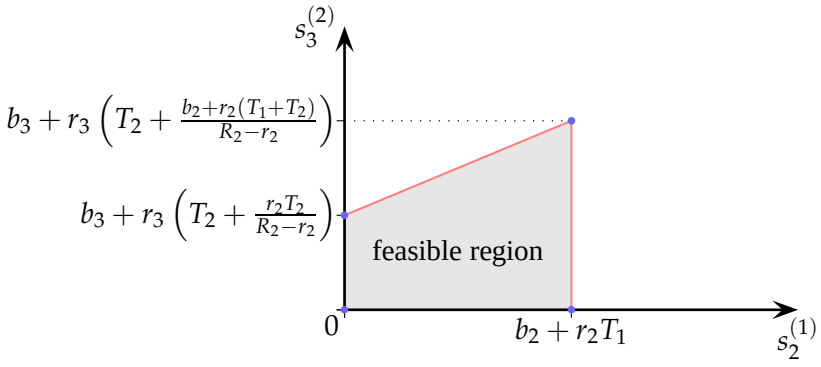


Abbildung 3.3: Feasible region for the overlapping interference scenario.

linearen Programm, erhält man für deren Lösungsvektor $\left(s_2^{(1)}, s_3^{(2)}\right)$ verschiedene Fälle

$$
\begin{aligned}
A \leq 0, B > 0 &: (b_2 + r_2T_2, 0) \\
A \leq 0, B \leq 0 &: \left(b_2 + r_2T_1, b_3 + r_3\left(T_2 + \frac{b_2 + r_2T_1 + r_2T_2}{R_2 - r_2}\right)\right) \\
A > 0, B > 0 &: (0, 0) \\
A > 0, B \leq 0, -CB \leq A &: \left(0, b_3 + r_3\left(T_2 + \frac{r_2T_2}{R_2 - r_2}\right)\right) \\
A > 0, B \leq 0, -CB > A &: \left(b_2 + r_2T_1, b_3 + r_3\left(T_2 + \frac{b_2 + r_2T_1 + r_2T_2}{R_2 - r_2}\right)\right)
\end{aligned}
$$

Mit konkreten Werten für die Service Raten können für dieses Szenario nun die Werte der Schlupfvariablen bestimmt und somit die Left-Over Service Curve für Datenstrom 1 berechet werden. Hier wird es für den letzten Fall $(A > 0, B \leq 0, -CB > A)$ exemplarisch durchgeführt.

$$\beta^1 = \beta_{R^{\text{l.o.}}, T^{\text{l.o.}}}$$

mit

$$
\begin{aligned}
R^{\text{l.o.}} =& (R_1 - r_2) \wedge (R_2 - r_2 - r_3) \wedge (R_3 - r_3) \\
T^{\text{l.o.}} =& T_1 + T_2 + T_3 + \frac{b_3 + r_3(T_2 + T_3)}{R_3 - r_3} + \frac{b_2 + r_2(T_1 + T_2)}{R_2 - r_2 - r_3} \\
&+ \frac{(R_2 - r_2 - R_3)r_3\frac{b_2 + r_2(T_1+T_2)}{R_2 - r_2}}{(R_2 - r_2 - r_3)(R_3 - r_3)}
\end{aligned}
$$

Dieser Fall entspricht der Situation, wenn $R_1 - r_2 < R_2 - r_2 - r_3$ $(A > 0)$, $R_2 - r_2 - r_3 \geq R_3 - r_3$ $(B \leq 0)$ und $r_2 > \frac{(R_2 - R_1)R_3}{R_2 - r_2}$ $(-CB > A)$. Dieser Fall ist interessant, da der Burst von Datenstrom 2 nicht an Knoten 1 gezahlt wird, obwohl dessen restlicher Service langsamer als der von Knoten 2 ist. Stattdessen wird er an Knoten 2 gezahlt, weil der Effekt

die Burstiness von Datenstrom 3 zu erhöhen, die dann wiederum an Knoten 3 gezahlt wird, stärker ist, als wenn er an Knoten 1 gezahlt worden wäre. Hier zeigt sich, dass die Burstiness auf den gewählten Parametern des Szenarios beruht. Folglich hängen auch die Delay Bounds von diesen Bedingungen ab.

Aus diesem Szenario kann das einfache Netzwerk mit zwei Knoten aus Abbildung 3.1 hergeleitet werden, wenn $F_3 = 0$ und $\beta_3 = +\infty$ gesetzt werden. Dann vereinfachen sich die Left-Over Service Curve für Datenstrom 1 und dem zugehörigen Delay Bound zu

$$\beta^1 = \beta_{(R_1 \wedge R_2) - r_2, T_1 + T_2 + \frac{b_2 + r_2 T_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{R_2 - r_2}}$$

und

$$\begin{aligned} d^{\texttt{OPT}} &= h(\alpha_1, \beta_1) \\ &= T_1 + T_2 + \frac{b_1 + b_2 + r_2 T_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{R_2 - r_2} \\ d^{\texttt{SFA}} &= T_2 + \frac{b_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{R_2 - r_2} \\ d^{\texttt{PMOO}} &= T_2 + \frac{b_1 + r_2 T_2}{(R_1 \wedge R_2) - r_2} \end{aligned}$$

Hier zeigt sich, dass die Burstiness des Datenstroms 2 nun korrekt an Knoten 2 und nicht an Knoten 1 verrechnet wird, wie in der PMOO-SFA. Zusätzlich werden die Bursts nur einmal berechnet, folglich wird der *Pay-Multiplexing-Only-Once*-Vorteil weiterhin beibehalten.

3.3 Allgemeine Feed-Forward-Netzwerke

In diesem Kapitel wird nun auf die Verallgemeinerung der Optimisation-Based Bounding Methode auf allgemeine Feed-Forward-Netzwerke eingegangen. Die Vorgehensweise ist analog zu Kapitel 3.2.

Als formale Repräsentation eines allgemeinen Netzwerks wird das Szenario in Abbildung 3.4 gewählt, das das Netzwerk aus Sicht des betrachteten Datenstroms darstellt. Für das Modell wird dabei nur vom Pfad des Datenstroms ausgegangen, bei dem die Knoten in aufsteigender Reihenfolge nummeriert sind. Insbesondere ist für das Konzept des Querverkehrs nur der Eintritts- und Austrittsknoten der jeweiligen Datenströme ausschlaggebend. Alle Datenströme, die den gleichen Eintrittsknoten i wie Austrittsknoten j besitzen, werden zu einem Datenstrom $F_{i,j}^{(k)}$ aufsummiert, wobei der Index $k = i-1, \ldots j$ den Datenstrom in Segmente unterteilt und Ursprungknoten des jeweiligen Segments angibt. Um dieses Konzept allgemein halten zu können, wird bei Eintrittsknoten i der Ursprungsknoten auf $i-1$ gesetzt. Dies führt zu einer maximalen Anzahl von $\frac{n}{2}(n+1)$ Datenströme des Querverkehrs. Falls einer dieser Datenströme nicht existiert, wird er auf 0 gesetzt. Im Folgenden wird jedoch von der maximalen Anzahl an Querverkehrs ausgegangen. Der betrachtete Datenstrom wird mit $F_{\texttt{int}}^{(k)}$,

$k = 0, \ldots n$ bezeichnet. Des Weiteren wird $t_0 \leq t_1 \leq t_2 \leq \ldots \leq t_n$ angenommen, wobei t_{i-1} der Start der letzten Backlogged-Period vor t_i an Knoten i ist.

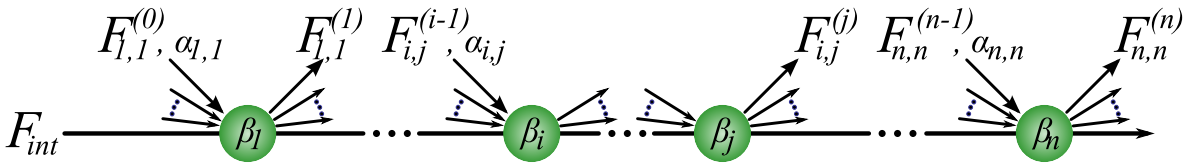


Abbildung 3.4: General feed-forward network scenario.

Über die geschaffenen Voraussetzungen, nämlich die Eigenschaft der Strict Service Curves, die Monotonie der Ein- und Ausgabefunktionen, sowie die Wahl des t_i lässt sich nun für den betrachteten Datenstrom schreiben:

$$F_{\mathtt{int}}^{(n)}(t_n) - F_{\mathtt{int}}^{(0)}(t_0) \geq \sum_{k=1}^{n} \left[\beta_k(t_k - t_{k-1}) - \sum_{l,m:l \leq k \leq m} \left(F_{l,m}^{(k)}(t_k) - F_{l,m}^{(k-1)}(t_{k-1}) \right) \right]^+$$

Die Kausalität des Systems und die Ankunftsbedingungen der Datenströme des Querverkehrs $F_{i,j}^{(k)}$ führen für alle $k : i \leq k \leq j$ zu dem Zusammenhang:

$$F_{i,j}^{(k)}(t_k) - F_{i,j}^{(i-1)}(t_{i-1}) \leq F_{i,j}^{(i-1)}(t_k) - F_{i,j}^{(i-1)}(t_{i-1}) \leq \alpha_{i,j}(t_k - t_{i-1})$$

Für jeden Datenstrom des Querverkehrs und jedem von ihm durchlaufenen Knoten erfolgt nun die Einführung der Schlupfvariablen $s_{i,j}^{(k)} \geq 0$ für $i, j : 1 \leq i \leq j \leq n$, $k : i - 1 \leq k \leq j$. Die Ankunftsbedingungen stellen sich wie folgt dar:

$$F_{i,j}^{(k)}(t_k) - F_{i,j}^{(k-1)}(t_{k-1}) = \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)} \right) - s_{i,j}^{(k)}, \text{ für } k \neq j \tag{3.4}$$

$$F_{i,j}^{(k)}(t_k) - F_{i,j}^{(k-1)}(t_{k-1}) \geq \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)} \right) - s_{i,j}^{(k)}, \text{ für } k = j$$

Dabei werden zwei Sonderfälle unterschieden, die gerade die Burstiness am Eintrittsknoten $s_{i,j}^{(i-1)} := \alpha_{i,j}(0)$ und nach dem Austrittsknoten $s_{i,j}^{(j)} := 0$ direkt bestimmen. Für den Fall $k = j$ wandelt sich die Gleichheit in eine Ungleichung, da hier der Querverkehr den Pfad des betrachteten Datenstroms verlässt und somit die Burstiness nicht mehr weitergegeben wird.

Als nächstes wird die obere Schranke der Schlupfvariablen hergeleitet:

$$\begin{aligned} \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)} \right) - s_{i,j}^{(k)} &= F_{i,j}^{(k)}(t_k) - F_{i,j}^{(k-1)}(t_{k-1}) \\ &\geq \left[\beta_k(t_k - t_{k-1}) - \sum_{\substack{l,m:l \leq k \leq m, \\ (m<j) \vee (m=j \wedge i>l)}} \left(F_{l,m}^{(k)}(t_k) - F_{l,m}^{(k-1)}(t_{k-1}) \right) \right]^+ \end{aligned}$$

An den Grenzen des Summationszeichens kann beobachtet werden, dass gerade den Datenströmen, die den betrachteten Datenstrom am ehesten verlassen, eine höhere Priorität zugewiesen wird. Bei Gleichheit wird zugunsten des Datenstroms entschieden, der am längsten den betrachteten Datenstrom begleitet, also anhand des Eintrittsknotens. Mit Formel 3.4 kann die obige Ungleichung nach der Schlupfvariablen aufgelöst werden:

$$s_{i,j}^{(k)} \leq \sup_{t_k - t_{k-1} \geq 0} \left\{ \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)}\right) - \left[\beta_k(t_k - t_{k-1}) \right.\right.$$
$$\left.\left. - \sum_{\substack{l,m: l \leq k \leq m, \\ (m<j) \vee (m=j \wedge i>l)}} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)}\right) - s_{l,m}^{(k)}\right) \right]^+ \right\} =: B_{i,j}^{(k)}$$

Zusätzlich zu den Vorzeichenbedingungen werden nun noch die oberen Schranken der Schlupfvariablen bestimmt. Dazu werden die bislang vorhandenen Formeln zuerst einmal zusammengefasst und umgestellt, so dass mit den Bedingungen $\mathcal{C}_1 = (1 \leq i \leq n) \wedge (t_i \geq t_{i-1})$ und $\mathcal{C}_2 = (1 \leq i \leq k \leq j \leq n) \wedge \left(0 \leq s_{i,j}^{(k)} \leq B_{i,j}^{(k)}\right)$ die allgemeine Formel für die Left-Over Service Curve wie folgt aufgestellt werden kann

$$F_{\mathtt{int}}^{(n)}(t_n) - F_{\mathtt{int}}^{(0)}(t_0) \geq \inf_{\mathcal{C}_1, \mathcal{C}_2} \left\{ \sum_{k=1}^{n} \left[\beta_k(t_k - t_{k-1}) \right.\right.$$
$$\left.\left. - \sum_{l,m: l \leq k \leq m} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)}\right) - s_{l,m}^{(k)}\right) \right]^+ \right\} \quad (3.5)$$

Auch hier muss ein Optimierungsproblem gelöst werden. Dessen Lösung hängt von der Wahl der Kurven ab. Sofern die Datenströme durch Token-Buckets und die Server durch Rate-Latencies beschränkt werden, handelt es sich wieder um ein lineares Optimierungsproblem. Die Koeffizienten der Zielfunktion, sowie die Nebenbedingungen ergeben sich aus der Lösung von Gleichung 3.5. Die Anzahl der Entscheidungsvariablen, also der Schlupfvariablen, des zugehörigen linearen Programms berechnet sich aus der Anzahl der Ströme des Querverkehrs multipliziert mit der Anzahl an Knoten, über die sie den betrachteten Datenstrom begleiten. Daher ist die maximale Anzahl der Entscheidungsvariablen bestimmt durch:

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1) = \frac{n(n+1)(n+2)}{6} = \mathcal{O}\left(n^3\right)$$

Jede Entscheidungsvariable wird durch zwei Nebenbedingungen nach unten und nach oben beschränkt – einmal die Vorzeichenbedingung und die funktionale Nebenbedingung, weshalb die Anzahl der Nebenbedingungen doppelt so hoch ist wie die der Entscheidungsvariablen.

Im Allgemeinen ist es sehr schwierig, aus der Formel 3.5 eine geschlossene Form herzuleiten, da eine sehr hohe Anzahl an Fällen unterschieden werden muss, es sei denn, das Optimierungsproblem hat eine bestimmte Form.

Ein Spezialfall dieses allgemeinen Modells ist der Sink-Tree, auf den hier aber nicht genauer eingegangen wird. Hier kann man den Delay Bound explizit berechnen, da die Struktur des Problems es für stückweise lineare konkave Arrival und konvexe Service Curves möglich macht, die Lösung explizit als geschlossene Formel darzustellen [SZM06].

3.4 Verallgemeinerung auf stückweise lineare Kurven

Das bisherige Vorgehen hat die Verwendung von Token-Bucket begrenzten Querverkehres und Rate-Latency begrenzte Server vorausgesetzt. Für die weitaus häufigeren Fälle muss diese Beschränkung jedoch aufgehoben und auf stückweise lineare konkave Arrival und konvexe Service Curves verallgemeinern werden.

Hierbei ist Proposition 1 wichtig, mit der man diese allgemeinere Darstellung der Kurven durch Zerlegung auf einzelne Token-Buckets und Rate-Latencies anwenden kann.

Proposition 1. *Gegeben sei für jeden Datenstrom $i = 1, \ldots, n$ des Querverkehrs eine stückweise lineare konkave Arrival Curve $\alpha_i = \bigwedge_{k_i=1}^{n_i} \gamma_{r_{k_i}, b_{k_i}}$, sowie für jeden Knoten $j = 1, \ldots, m$ des Pfades des betrachteten Datenstroms eine stückweise lineare konvexe Strict Service Curve $\beta_j = \bigvee_{l_j=1}^{m_j} \beta_{R_{l_j}, T_{l_j}}$. Die Service Curve für den betrateten Datenstrom unter der Annahme von Arbitrary Multiplexing ist beschränkt durch:*

$$\beta^{\text{l.o.}} \geq \bigvee_{i=1}^{n} \bigvee_{j=1}^{m} \bigvee_{k_i=1}^{n_i} \bigvee_{l_j=1}^{m_j} \beta^{\text{l.o.}}_{\{k_i\},\{l_i\}}$$

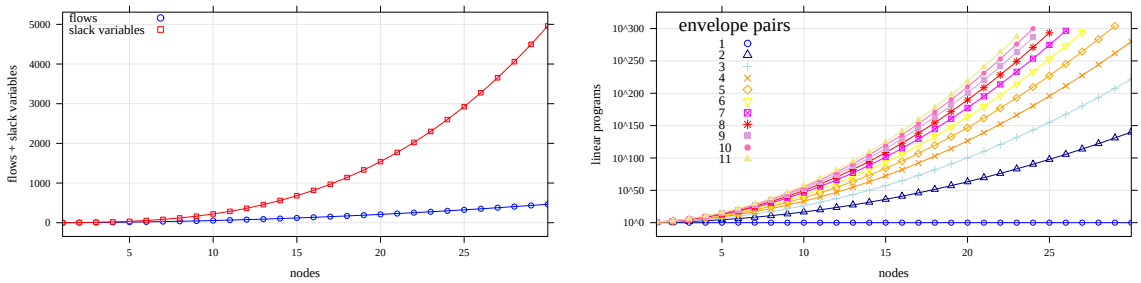
wobei $\beta^{\text{l.o.}}_{\{k_i\},\{l_i\}}$ die End-to-End Left-Over Service Curves unter Arbitrary Multiplexing, für eine bestimmte Kombination ist. Für eine Kombination wird ein einzelner Token-Buckets für jeden Fluss des Querverkehrs und eine einzelnen Rate-Latency pro Knoten verwendet.

Auf den Beweis durch vollständige Induktion wird hier mit Verweis auf [SZF07] verzichtet.

Die entscheidende Konsequenz aus der Zerlegung von Arrival und Service Curves in jeweils Token-Buckets und Rate-Latencies ist, dass zahlreiche lineare Programme gelöst werden müssen. Die end-to-end Left-Over Service Curve wird dann über das punktweise Maximum bestimmt. Gerade diese Verallgemeinerung der Curves und die damit verbundene Zerlegung schafft ein beträchtliches Problem, weil die Anzahl der zu berechnenden Left-Over Service Curves, und damit die linearen Programme, mit der Anzahl der linearen Segmente massiv steigt. Die Kombinationsmöglichkeiten können durch $\prod_{i=1}^{n} \prod_{j=1}^{m} n_i m_j$ berechnet werden.

Eine graphische Veranschaulichung dieser Größenordnung zeigt Abbildung 3.5b, wobei hier der Einfachheit halber eine Rate-Latency Service Curve und sämtliche Datenströme die gleiche Arrival Curve besitzen. Eine Arrival Curve wird durch einen Envelope Prozess $\{\vec{\sigma}, \vec{\rho}\}$ mit mehreren $\{\sigma, \rho\}$ Paaren beschränkt. Ein $\{\sigma, \rho\}$ Envelope kann einfach durch eine Token-Bucket-Funktion $\gamma_{\rho,\sigma}$ beschrieben werden [MP06]. Der maximale darstellbare Wert einer

Fließkommazahl doppelter Genauigkeit liegt bei etwa $1.8 * 10^{+308}$, welcher aber nicht immer ausreicht, wie in manchen der Kombinationen in Abbildung 3.5b zu sehen ist. Dies alleine weckt schon ein gewisses Verständnis für die Dimensionen, die hier zu bewältigen sind. Als Vergleich: Ein Jahr hat $365 * 24 * 60 * 60 = 31.536.000$ Sekunden, so dass schon bei grober Abschätzung eine vollständige Berechnung sämtlicher Kombinationen für die meisten Problemgrößen in absehbarer Zeit nicht möglich ist, da diese massiv ansteigen.

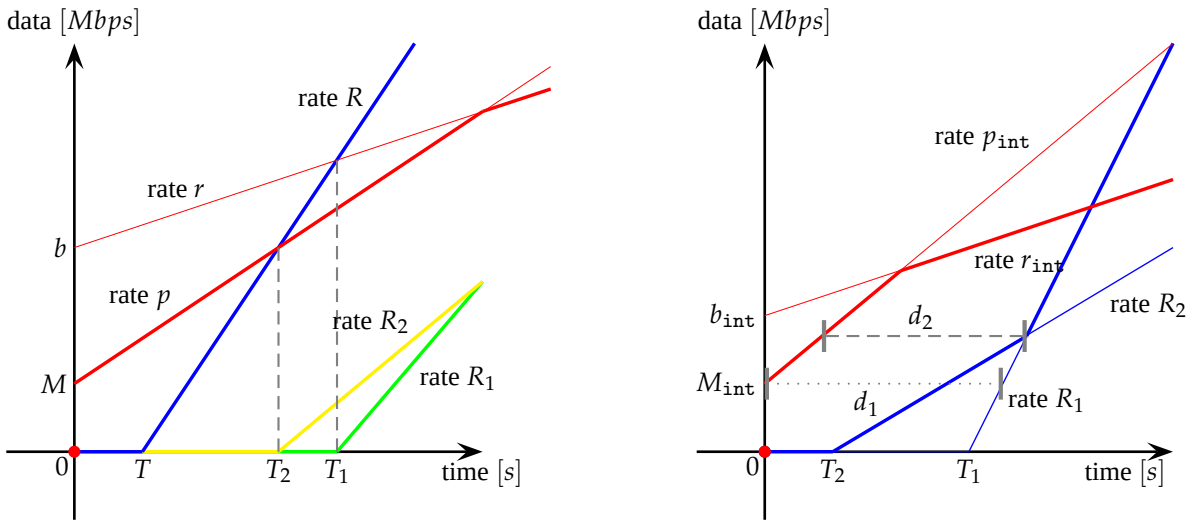


(a) Flows and slack variables

(b) Linear programs

Abbildung 3.5: Maximum count of various aspects.

Ein wesentlicher Nachteil der Zerlegung ist, dass die Bounds nicht mehr Tight sind, wie das Gegenbeispiel in Abbildung 3.6 verdeutlicht.



(a) Multiplexing of the interfering flow.

(b) Left-over service for the flow of interest.

Abbildung 3.6: Example to show why the bounds are not tight.

Angenommen wird ein einzelner Server, dessen Service durch eine Rate-Latency-Funktion $\beta_{R,T}$ garantiert wird. Für diesen Server kann jetzt die Left-Over Service Curve berechnet

werden, wenn ein T-SPEC (M, p, r, b) bedient wird. Durch visuelle Berechnung mit Abbildung 3.6a und den Voraussetzungen $R > p$ und $T_1 < \frac{b-M}{p-r}$, wobei $\frac{b-M}{p-r}$ gerade der Schnittpunkt $\gamma_{M,p}(t) = \gamma_{b,r}(t)$, $t > 0$ ist, ergibt sich für die Left-Over Service Curve:

$$\begin{aligned}\beta^{\text{l.o.}}_{R_1,T_1} &= [\beta_{R,T} - \gamma_{r,b}]^+ \\ &= \beta_{R-r,\frac{b+RT}{R-r}}\end{aligned}$$

Wenn die Proposition 1 zur Zerlegung angewandt wird, liefert dies aber zwei Left-Over Service Curves:

$$\begin{aligned}\beta^{\text{l.o.}}_{R_1,T_1} &= \beta_{R-r,\frac{b+RT}{R-r}} \\ \beta^{\text{l.o.}}_{R_2,T_2} &= \beta_{R-p,\frac{M+RT}{R-p}}\end{aligned}$$

die aufgrund einer geschickten Parameterwahl gerade

$$\beta^{\text{l.o.}}_{R',T'} = \beta^{\text{l.o.}}_{R_1,T_1} \vee \beta^{\text{l.o.}}_{R_2,T_2}$$

liefern. Mit diesem noch zur Verfügung stehenden Service $\beta^{\text{l.o.}}_{R',T'}$ kann nun der Delay für einen weiteren Datenstrom betrachet werden. Dieser Datenstrom wird wieder als T-SPEC $(M_{\texttt{int}}, p_{\texttt{int}}, r_{\texttt{int}}, b_{\texttt{int}})$ definiert, so dass sich das Szenario in Abbildung 3.6b einstellt. Das führt nun zu unterschiedlichen Delay Bounds. Zum einen wäre da der gepunktete Delay Bound $d_1 = h(\gamma_{p,M}, \beta_{R_1,T_1})$, zum anderen die gestrichelte Linie für den Delay $d_2 = h(\gamma_{p,M}, \beta_{R_1,T_1} \vee \beta_{R_2,T_2})$. Ein rein visueller Vergleich ergibt schon mit der Voraussetzung $R_2 > p_{\texttt{int}} > R_1$, dass $d_2 < d_1$ ist.

4 Implementierung

Aus dem vorherigen Kapitel wurde deutlich, dass in Szenarien mit Token-Bucket beschränkten Datenströmen und Rate-Latency Garantien Tight Bounds berechnet werden können. Die Generalisierung auf stückweise lineare konkave und konvexe Kurven stößt aber ein Tor in eine Komplexitätsklasse auf, in der es unumgänglich ist, diese auf ein akzeptables Ausmaß zu reduzieren. Auf analytischem Wege scheint dies aussichtslos. In dieser Diplomarbeit wird deshalb der Ansatz verfolgt, nur einen geringen Teil der möglichen Kombinationen zu berechnen und einen Näherungswert zu erhalten.

Das vorangegangene Kapitel hat sich mit den Formeln und mathematischen Grundlagen beschäftigt, die nun implementiert werden sollen. Eine wichtige Basis bildet hierbei der DISCO Network Calculator [DIS08]. Er enthält schon alle benötigten Grundelemente und wird erweitert mit der im vorangegangenen Kapitel beschriebenen OBA.

Das erste wichtige Ziel war, den gewachsenen Code so zu refactorn und Abhängigkeiten aufzubrechen, dass neue Implementierungen nahtlos eingebettet werden können.

Die weiteren Unterkapitel beschäftigen sich mit der Umsetzung, wobei zuerst einmal die neue Klassenstruktur und Vererbungshierarchie vorgestellt wird. Dann folgen konkrete Beschreibungen der neuen Programmteile. Diese beinhalten vor allem die Algorithmen für die Formeln der in Kapitel 3 vorgestellten Methode für die Berechnung der optimierten Bounds. Danach wird kurz die Anbindung eines linearen Optimierers vorgestellt. Zum Schluss werden die umgesetzten Methoden zur Komplexitätsreduzierung beschrieben.

4.1 Architektur

Wie schon erwähnt, handelt es sich um gewachsenen Code, der hier erweitert wird. Die bisherige Implementierung besaß als Kern die Klasse *NetworkAnalyser*, die sich um sämtliche Aspekte der Analyse eines Szenarios kümmerte. Die Zuständigkeiten der Klasse erstreckten sich über viele Gebiete, u.a. die Verwaltung von Netzwerkgraph, Datenströmen und Parametern des Network Calculus. Also der Parametrisierung der Beschränkungen für Datenströme und Server, sowie den kompletten Quellcode der Analysemethoden.

Monolithische Klassen haben bei Erweiterungen des Codes den klaren Nachteil, dass sie auf Dauer durch die ständig wachsende Komplexität unübersichtlich und unwartbar werden. Damit kann es zu unbeabsichtigten Nebeneffekten kommen. Um die Wartbarkeit der Software zu

erhalten, sind Refactorings notwendig. Daher ist es sinnvoll, die Anwendung zu strukturieren. Dieses Prinzip heißt *Separation of Concern*.

Dem *NetworkAnalyser*werden viele Aufgaben entzogen und nur noch die notwendigsten überlassen. Der neue Aufgabenbereich beinhaltet nur noch die Verwaltung des Szenarios. Er ist zuständig für die Verwaltung des Netzwerks und der darüber laufenden Datenströme. Ebenfalls assoziiert er die dazugehörigen Network Calculus Elemente, also Arrival und Service Curves. Die Methodenschnittstelle bietet Zugriffsmöglichkeiten auf diese Daten. Ein Diagramm der neuen Struktur ist in Abbildung 4.1 zu sehen.

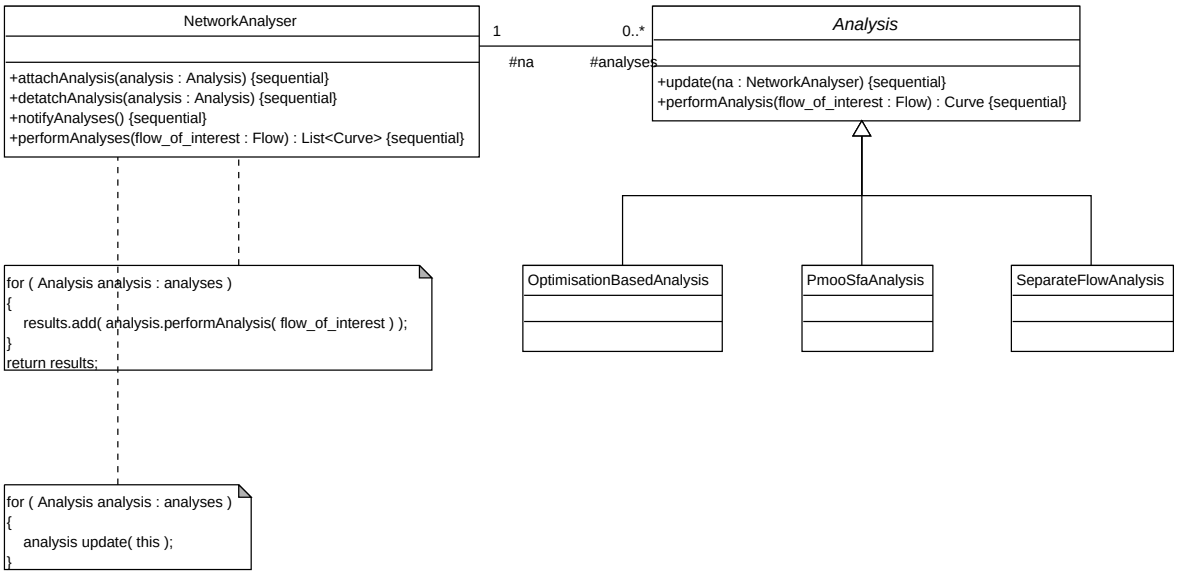


Abbildung 4.1: Dependency and inheritance diagram for the new *NetworkAnalyser*.

Dagegen wurden die Analysemethoden aus dem *NetworkAnalyser*extrahiert und in eine Vererbungshierarchie verlagert. Als Vorlage diente das Beobachter Entwurfsmuster [GHJV95]. Ein Beobachter wird benachrichtigt, wenn sich der Zustand des Subjekts, hier also Eigenschaften des *NetworkAnalyser*, ändert. Die Beobachter sind in diesem Falle die verschiedenen Analysemethoden. Eine Veränderung in der verwalteten Netzwerktopologie oder den Datenströmen könnte dies auslösen. Ebensogut kann auf eine veränderte Service Curve reagiert werden. Umgekehrt kann ein Beobachter aber auch den aktuellen Zustand abfragen, also auf dem aktuellen Netzwerk die Berechnung ausführen, wenn sich z.B. ein Parameter der Analyse ändert oder vom Benutzer eine Aktualisierung aktiv gefordert wird.

Diese Funktionalität wäre im Hinblick auf die Erweiterung des DISCO Network Calculators [DIS08] mit einer graphischen Benutzeroberfläche hilfreich.

Es ist möglich, dem *NetworkAnalyser*nur bestimmte Analysen zu übergeben. Jede Instanz der Unterklassen von *Analysis*verwaltet ihre eigenen Einstellungen, wie zum Beispiel die Verwendung einer Maximum Service Curve. Daher ist es auch möglich, die gleiche Analyse unter Verwendung verschiedener Parameter zu vergleichen. Die Berechnung der Bounds kann entweder

wie gehabt über *NetworkAnalyser*geschehen, oder aber auch über die jeweilige Analyse. Eine entsprechende Methode *performAnalysis()*liefert für eine Analyse direkt die Left-Over Service Curve. Beim *NetworkAnalyser*dagegen werden sämtliche Kurven in der Reihenfolge der eingefügten Analysen als Liste geliefert.

Die in Kapitel 3 vorgestellte Optimisation-Based Bounding Methode findet nun als Spezialisierung von *Analysis*ihren Platz in der Hierarchie. Für die schon existierenden Analysen reichte bislang eine einzige Klasse aus. Hier werden aber für das komplexere Verhalten zusätzliche Strukturen benötigt. Die in Abbildung 4.2 dargestellten Klassen *OptimisationBasedComputation*und *Decompositor*bilden den wichtigsten Teil. Das sind zwar nicht die einzigen Klassen, die im Quellcode verwendet werden. Die nicht dargestellten sind jedoch nicht essentiell wichtig für das Zusammenspiel. Sie vereinfachen nur den Quellcode und kapseln einige Daten.

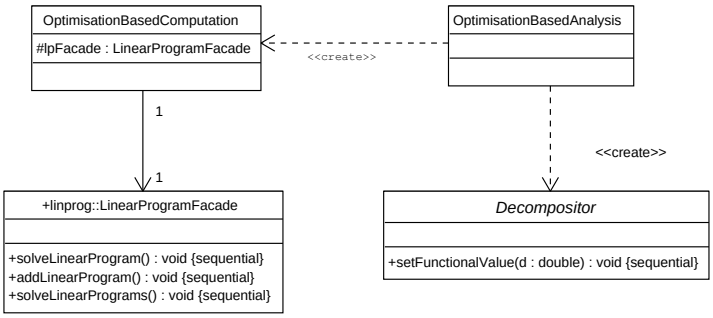


Abbildung 4.2: Dependency and inheritance diagram for the Optimisation-Base Bounding Method

Essentiell dagegen ist die Realisierung der Optimisation-Based Bounding Methode. Wie schon in Kapitel 3 angeführt, besteht diese Methode aus zwei wesentlichen Teilen: zum einen die Berechnung von Tight Bounds für Token-Bucket beschränkte Datenströme und Servern mit Rate-Latency Garantien in allgemeinen Feed-Forward-Netzwerken in Kapitel 3.3. Zum anderen die Verallgemeinerung auf stückweise lineare konkave Arrival und konvexe Service Curves in Kapitel 4.2, aber einhergehend mit dem Verlust der Tightness.

Die abstrakte Klasse *Decompositor*fasst die Zerlegung der Kurven zusammen. Die Konsequenzen der Zerlegung wurden schon in Kapitel 3.4 und der Propositon 1 beschrieben. Die genaue Ableitungshierarchie mit verschiedenen Methoden und deren Funktionsweise werden später ausführlich in Kapitel 4.4 betrachtet. Dort werden in einzelnen Unterklassen der komplette Durchlauf und zwei Suchverfahren spezialisiert.

Resultierend aus der Zerlegung ergeben sich einzelne Rate-Latencies und Token-Buckets. Mit diesen wird die Vorgehensweise zur Berechnung für allgemeine Feed-Forward-Netzwerke angewendet. Der Kern der Berechnung ist das Lösen eines linearen Programms. Damit steht auch die Funktionsweise der Klasse *OptimisationBasedComputation*fest: Sie beinhaltet die Algorithmen zum Berechnen der geschlossenen Formeln, sowie die Anbindung an einen linearen Optimierer. Beides wird in den folgenden Unterkapiteln näher betrachtet.

4.2 Algorithmen

Im Kapitel 3.2 wurden die notwendigen Formeln schon für ein Beispiel vorgestellt und in Kapitel 3.3 für Feed-Forward Netze verallgemeinert. Jetzt müssen die Formeln auch für die Java-Implementierung aufbereitet werden. In den folgenden Unterkapiteln werden die Formeln erklärt und daraus der Pseudocode abgeleitet.

Die Voraussetzung, sich auf den speziellen Fall der Token-Buckets und Rate-Latencies zu konzentrieren, führt zu wesentlichen Vereinfachungen in den einzelnen Umsetzungen. Für diesen Fall können nämlich geschlossene Formeln angegeben werden.

Das allgemeine Zusammenspiel der Formeln – ausgehend von der Parametrisierung des Szenarios, bis hin zur Präsentation der Zahlenwerte des Output Bounds – wird hier nicht mehr im Detail erklärt. Vielmehr werden die Pseudocodefragmente einzeln betrachtet. Verweise zwischen den Fragmenten sind durch Prozeduraufrufe gekennzeichnet. Die Reihenfolge der beschriebenen Formeln richtet sich nach deren Auftreten im Methodenablauf, also dem Kontrollfluss folgend.

4.2.1 Allgemeinere Arrival und Service Curves

Um den allgemeinen Fall der stückweise linearen konkaven Arrival und konvexen Service Curves behandeln zu können, muss ein Szenario auf den speziellen Fall der Token-Buckets und Rate-Latencies herunter skaliert werden. Daher wird zuerst die Zerlegung der Kurven vorgenommen.

Der Envelope jeder Arrival Curve eines allgemeinen Feed-Forward Netzwerkes mit n Knoten, sowie die Service Curves, werden mit

$$\alpha_{i,j} = \bigwedge_{k_{i,j}=1}^{m_{i,j}} \gamma_{r_{k_{i,j}},b_{k_{i,j}}} \qquad i = 1,\ldots,n;\ j = i,\ldots,n$$

$$\beta_i = \bigvee_{l_i=1}^{n_i} \beta_{R_{l_i},T_{l_i}} \qquad i = 1,\ldots,n$$

dargestellt. Das i gibt den Eintritts- und das j den Austrittsknoten des jeweiligen Datenstroms an. Über die Zerlegung aus Proposition 1

$$\beta^{\text{l.o.}} \geq \bigvee_{i=1}^{n} \bigvee_{j=1}^{m} \bigvee_{k_{i,j}=1}^{m_{i,j}} \bigvee_{l_j=1}^{n_j} \beta^{\text{l.o.}}_{\{k_{i,j}\},\{l_i\}}$$

$$\{k_{i,j}\} = \{k_{1,1},\ldots,k_{1,n},k_{2,2},\ldots,k_{n,n}\},$$
$$\{l_i\} = \{l_1,\ldots,l_n\}$$

kann schon sehr zielgerichtet der Pseudocode abgeleitet werden. Wie schon erwähnt, ist diese Zerlegung wegen den sich ergebenden, kombinatorischen Möglichkeiten sehr problematisch.

Deshalb ist dieser Algorithmus der wesentliche Schwerpunkt, den es zu optimieren gilt. Das folgende Pseudocodefragment ist unter diesem Aspekt als „Brute Force“ Methode anzusehen, die sämtliche Kombinationen durchläuft.

Algorithmus 1 : Optimisation-Based Bounding Computation

Data :
Curves β_i
Curves $\alpha_{i,j}$
Result :
Curve $\beta^{\text{l.o.}}$
begin
$\beta^{\text{l.o.}} \leftarrow 0$
for $1 \leq i \leq n$ **do**
for $1 \leq l_i \leq n_i$ **do**
for $i \leq j \leq n$ **do**
for $1 \leq k_{i,j} \leq m_{i,j}$ **do**
$\beta^{\text{l.o.}}_{\{k_{i,j}\},\{l_i\}} \leftarrow \texttt{LeftOverServiceCurve}(\beta_{R_{l_i},T_{l_i}}, \gamma_{r_{k_{i,j}},b_{k_{i,j}}})$
$\beta^{\text{l.o.}} \leftarrow \max\left(\beta^{\text{l.o.}}, \beta^{\text{l.o.}}_{\{k_{i,j}\},\{l_i\}}\right)$
end

4.2.2 Left-Over Service Curve

Die Berechnung der Left-Over Service Curve für eine einzelne Kombination aus Token-Buckets und Rate-Latencies erfordert die Lösung der Formel (3.5):

$$F^{(n)}_{\texttt{int}}(t_n) - F^{(0)}_{\texttt{int}}(t_0) \geq \inf_{\mathcal{C}_1,\mathcal{C}_2} \left\{ \sum_{k=1}^{n} \left[\beta_k(t_k - t_{k-1}) - \sum_{l,m:l\leq k\leq m} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s^{(k-1)}_{l,m} \right) - s^{(k)}_{l,m} \right) \right]^{+} \right\}$$

In dieser Formel fließt die Lösung des linearen Programms ein. In der äußeren Summe wird der jeweils noch zur Verfügung stehende Service eines Knotens unter Blind-Multiplexing gebildet. Mit den gegebenen Service Curves ist dies recht unkompliziert. Interessanter ist aber die Bildung der inneren Summe, in der sämtlicher anfallender Querverkehr zusammengefasst wird. Ohne die Voraussetzung von Token-Buckets, würde deren Lösung in einer Menge Fallunterscheidung enden. So aber lässt sich, durch Einsetzen, die Formel zu einer geschlossenen Formel führen.

$$\gamma_{r_k^*,b_k^*}(t_k - t_{k-1}) + s_k^* = \sum_{l,m:l \le k \le m} \left(\gamma_{l,m}(t_k - t_{l-1}) - \left(\gamma_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)} \right) - s_{l,m}^{(k)} \right)$$

Über Fallunterscheidungen werden alle Token-Buckets und Schlupfvariablen jeweils zusammengefasst. Hierbei kann über einen Koeffizientenvergleich bestimmt werden, welche Bursts, Raten oder Schlupfvariablen in b_k^*, r_k^* und s_k^* einfließen.

Algorithmus 2 : Computation of the inner sum

Data :
- Index k
- Curves $\gamma_{r_{l,m},b_{l,m}}$
- Values $s_{l,m}^{(k-1)}, s_{l,m}^{(k)}$

Result :
- Curve $\gamma_{r_k^*,b_k^*}$
- List s_k^*

begin
- $b_k^* \leftarrow 0$
- $r_k^* \leftarrow 0$
- **for** $1 \le l \le k$ **do**
  - **for** $k \le m \le n$ **do**
    - **if** $k = l$ **then**
      - $b_k^* \leftarrow b_k^* + b_{l,m}$
      - $r_k^* \leftarrow r_k^* + r_{l,m}$
      - $s_k^* \leftarrow -s_{l,m}^{(k)}$
    - **else if** $k = m$ **then**
      - $r_k^* \leftarrow r_k^* + r_{l,m}$
      - $s_k^* \leftarrow s_{l,m}^{(k-1)}$
    - **else if** $l < k$ **then**
      - $r_k^* \leftarrow r_k^* + r_{l,m}$
      - $s_k^* \leftarrow s_{l,m}^{(k-1)}$
      - $s_k^* \leftarrow -s_{l,m}^{(k)}$

end

Mit der so berechneten inneren Summe sind nun alle Werte vorhanden, um die Left-Over Service Curve zu bestimmen – die Berechnung der Schlupfvariablen $s_{l,m}^{(k)}$ durch Lösen eines linearen Programms vorausgesetzt. Deren Rate und Latency lassen sich aus der Ankunftsbedingung durch vorsichtige Abschätzungen an den richtigen Stellen zu folgenden Gleichungen umformen.

$$
\begin{aligned}
R^{\text{l.o.}} &= \min_{k=1}^{n} \left(R_k - r_k^*\right) \\
T^{\text{l.o.}} &= \sum_{k=1}^{n} \left(T_k + \frac{b_k + r_k^* T_k}{R_k - r_k^*} + \frac{s_k^*}{R_k - r_k^*}\right) \\
&= \sum_{k=1}^{n} \left(T_k + \frac{b_k + r_k^* T_k}{A_k} + \frac{s_k^*}{A_k}\right)
\end{aligned}
$$

Algorithmus 3 : Left-over service curve computation

Data :
Curves $\beta_{R_{l_i}, T_{l_i}}$
Curves $\gamma_{r_{k_{i,j}}, b_{k_{i,j}}}$

Result :
Curve $\beta^{\text{l.o.}}_{\{k_{i,j}\},\{l_i\}}$

begin
- **for** $1 \leq k \leq n$ **do**
  - $\left(\gamma_{r_k^*, b_k^*}, s_k^*\right) \leftarrow$ `CalculateInnerSum(`$\gamma_{r_{l,m}, b_{l,m}}, s_{l,m}^{(k-1)}, s_{l,m}^{(k)}$`)`
  - $A_k \leftarrow \left(R_k - r_k^*\right)$
- $R^{\text{l.o.}} \leftarrow \infty$
- **for** $1 \leq k \leq n$ **do**
  - $R^{\text{l.o.}} \leftarrow \min\left(R^{\text{l.o.}}, R_k - r_k^*\right)$
- $T^{\text{l.o.}} \leftarrow 0$
- **for** $1 \leq k \leq n$ **do**
  - $T^{\text{l.o.}} \leftarrow T^{\text{l.o.}} + \left(T_k + \frac{b_k + r_k^* T_k}{R_k - r_k^*} + \frac{s_k^*}{R_k - r_k^*}\right)$

end

4.2.3 Erstellung des linearen Programms

In diesem Codefragment werden sämtliche Werte für das lineare Programm aufbereitet. Die hier behandelten linearen Programme haben die Form [HK06]

$$
\begin{aligned}
\text{minimise.} \quad & c_1 x_1 + \ldots + c_n x_n & & \\
\text{subject to (s.t.) the constraints} \quad & a_{i1} x_1 + \ldots + a_{in} x_n & \leq b_i & \quad \forall i = 1, \ldots, n \\
& x_i & \geq 0 & \quad \forall i = 1, \ldots, n
\end{aligned}
$$

Die Parameter $\vec{x}$ der Zielfunktion $\vec{c}\vec{x}$ sind die Schlupfvariablen $s_{l,m}^{(k)}$. $\vec{c}$ bildet sich aus den Koeffizienten der Schlupfvariablen in der Ankunftsbedingung. Die funktionalen Bedingungen

$A\vec{x}$ mit der $n \times n$ Matrix der Koeffizienten A bestimmen sich durch die Abhängigkeit der oberen Schranken $B_{l,m}^{(k)}$ von den Schlupfvariablen.

Algorithmus 4 : Solving the linear program

Data :
- Index k
- Matrix $B_{i,j}^{(k)}$
- Matrix $s_{i,j}^{(k)}$
- Lists s_k^*
- Array A_k

Result :
- Matrix $s_{i,j}^{(k)}$

begin
- **foreach** $s_{i,j}^{(k)}$ **do**
  - **foreach** s_k^* **do**
    - **if** $s_{i,j}^{(k)}$ **in** s_k^* **then**
      - add A_k to the factor
  - print objective function with corresponding factors
- **foreach** $s_{i,j}^{(k)}$ **do**
  - print non-negative constraints $s_{i,j}^{(k)} \geq 0$
  - print functional constraints $s_{i,j}^{(k)} \leq B_{i,j}^{(k)}$

end

4.2.4 Obere Schranke der Schlupfvariablen

Der letzte Algorithmus ist die Ausarbeitung der geschlossenen Formel für die obere Schranke der Schlupfvariablen. Bei der Beschreibung der Optimisation-Based Bounding Methode anhand eines Beispiels in Kapitel 3.2 kann beobachtet werden, dass die obere Schranke auch von vorherigen Schlupfvariablen abhängt. Dies ist einleuchtend, denn die Burstiness eines Datenstroms wirkt sich auch auf andere aus. Deswegen kann die Abhängigkeit zwischen den Schlupfvariablen besser durch

$$B_{i,j}^{(k)}\left(s_{l,m}^{(n)}\right) = B_{i,j}^{(k)^*} + \sum a_{l,m}^{n} s_{l,m}^{(n)}$$

dargestellt werden. Die Schranke wird also aus einem festen Anteil $B_{i,j}^{(k)^*}$ und einer Summe zusammengesetzt. Die vollständige Formel lautet:

$$s_{i,j}^{(k)} \leq \sup_{t_k - t_{k-1} \geq 0} \left\{ \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)} \right) - \left[\beta_k(t_k - t_{k-1}) \right.\right.$$

$$\left.\left. - \sum_{\substack{l,m: l \leq k \leq m, \\ (m<j) \vee (m=j \wedge i>l)}} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)} \right) - s_{l,m}^{(k)} \right) \right]^+ \right\} =: B_{i,j}^{(k)}$$

Eine genauere Betrachtung der inneren Summe führt zu der Beobachtung, dass sich dieser Algorithmus 5 von dem weiter oben beschriebenen nur durch den Laufindex unterscheidet. Das liegt an der unterschiedlichen Priorisierung der Datenströme in den einzelnen Fällen. Für die Berechnung der Left-Over Service Curve in Algorithmus 3 wird sämtlicher Querverkehr betrachtet. Hier werden die Datenströme priorisiert, die den Pfad des betrachteten Datenstroms am ehesten verlassen oder am längsten begleiten, wie in Kapitel 3.3 beschrieben. Die Herleitung des folgenden Algorithmus entspricht also der des Algorithmus 2.

Algorithmus 5 : Computation of the inner sum

Data :
Curves $\alpha_{l,m}(t)$
Result :
Curve $\gamma_{r_k,b_k}(t)$
List s_k
begin
$b_k \leftarrow 0$
$r_k \leftarrow 0$
for $1 \leq l \leq k$ **do**
for $k \leq m \leq n$ **do**
if $\neg((m < j)$ **or** $(m = j$ **and** $l < i))$ **then**
continue
if $k = l$ **then**
$b_k \leftarrow b_k + b_{l,m}$
$r_k \leftarrow r_k + r_{l,m}$
$s_k \leftarrow -s_{l,m}^{(k)}$
else if $k = m$ **then**
$r_k \leftarrow r_k + r_{l,m}$
$s_k \leftarrow s_{l,m}^{(k-1)}$
else if $l < k$ **then**
$r_k \leftarrow r_k + r_{l,m}$
$s_k \leftarrow s_{l,m}^{(k-1)}$
$s_k \leftarrow -s_{l,m}^{(k)}$
end

Über die Fallunterscheidungen werden alle Token-Buckets und Schlupfvariablen jeweils zu b_k^*, r_k^* und s_k^* zusammengefasst. Die Bestimmung der oberen Schranke für die Schlupfvariablen wird in Algorithmus 6 dargestellt.

Algorithmus 6 : Upper bounds on slack variables

Data :
Curves $\alpha_{i,j}(t)$
Curve $\gamma_{r_k,b_k}(t)$
List s_k
Matrix $s_{i,j}^{(k)}$
Result :
Matrix $B_{i,j}^{(k)}$
begin
for $1 \leq i \leq n$ **do**
for $i \leq j \leq n$ **do**
if $\neg\exists F_{i,j}$ **then**
continue
for $i-1 \leq k \leq j$ **do**
if $k = j$ **then**
$B_{i,j}^{(j)} \leftarrow 0$
else if $k = i-1$ **then**
$B_{i,j}^{(i-1)} \leftarrow \alpha_{i,j}(0)$
else if $k = i$ **then**
$B_{i,j}^{(i)} \leftarrow b_{i,j} + r_{i,j}\left(T_k + \frac{b_k + r_k T_k}{R_k - r_k} + \frac{s_k}{R_k - r_k}\right)$
else if $i < k < j$ **then**
$B_{i,j}^{(k)} \leftarrow r_{i,j}\left(T_k + \frac{b_k + r_k T_k}{R_k - r_k} + \frac{s_k}{R_k - r_k}\right) + s_{i,j}^{(k-1)}$
end

4.3 Linear Program Solver

Da für die Berechnung der OBA auf das Lösen linearer Programme (LP) zurückgegriffen wird, ist es sinnvoll, schon bestehende Programme oder Frameworks zu benutzen. Es gibt dafür eine große Anzahl an Software-Lösungen[1]. Für die aktuelle Realisierung wurde lp_solve 5.5 [lps09] verwendet. Wenn eine Java API existiert, so kann die bestehende Implementierung ersetzt oder erweitert werden.

[1] `http://en.wikipedia.org/wiki/Linear_programming`

Um unabhängiger von einem bestimmten Programm zu sein, wurde die Erstellung und der Aufbau des linearen Programms in ein Subsystem ausgelagert und hinter einer Fassade [GHJV95] versteckt, wie in Abbildung 4.3 dargestellt.

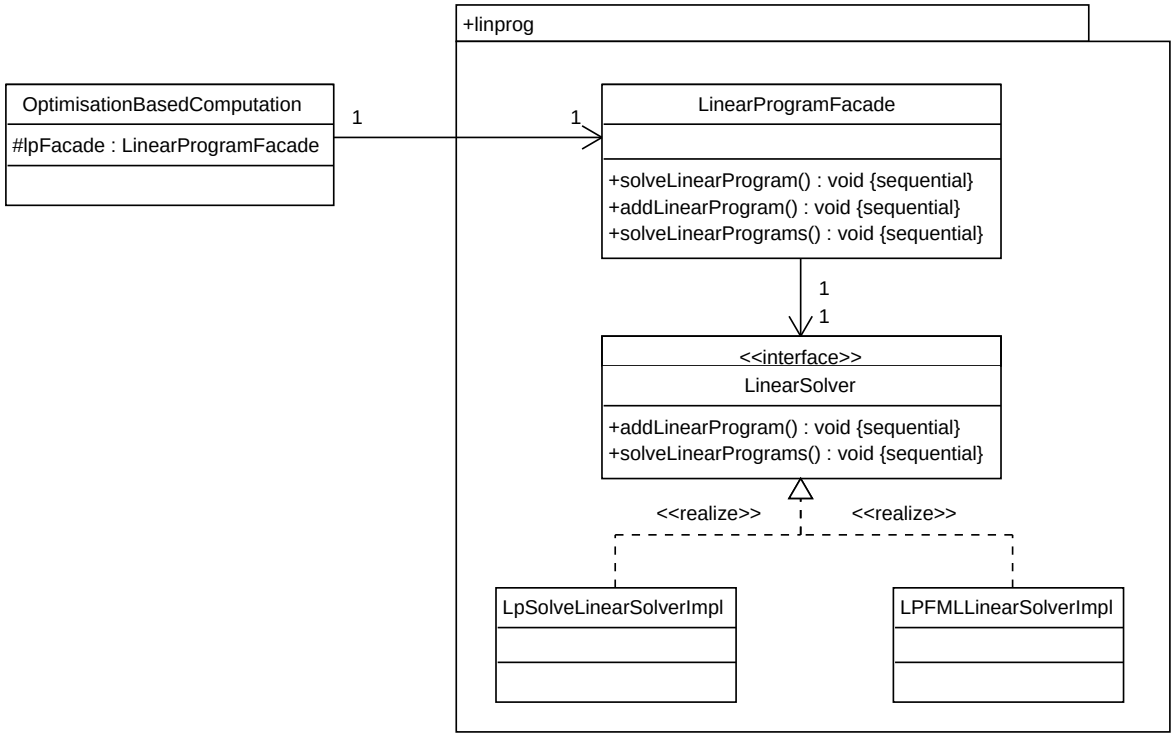


Abbildung 4.3: Class diagram for the linear program package.

Solange an der Schnittstelle der Klasse *LinearProgramFacade*nichts verändert wird, kann die Implementierung beliebig verändert oder erweitert werden. Wichtig für die aufrufende Klasse sind drei Methoden zur Berechnung der linearen Programme. Die Daten dafür können an *solveLinearProgram()*übergeben werden und die Lösung wird bei Beendigung des Methodenaufrufs zurückgeliefert. Alternativ können die Daten mehrerer linearer Programme nacheinander mit *addLinearProgram()*übermittelt werden und erst die Methode *solveLinearPrograms()*liefert die Lösungen. Dabei sollte beachtet werden, dass die Speicherung der Programme sehr speicherintensiv ist.

Um die Funktionalität zu erweitern oder zu ersetzen, muss nur das Paket *linprog*verändert werden. Die aufrufenden Klassen sehen immer nur *LinearProgramFacade*und brauchen nichts über die eigentliche Umsetzung zu wissen. In Bezug auf Sichtbarkeit und Zugriff bietet die Fassade dem Subsystem ein Äquivalent, wie das Interface einer Klasse.

Die beiden Realisierungen der Schnittstelle *LinearSolver*bieten verschiedene Möglichkeiten, ein lineares Programm zu lösen. *LpSolveLinearSolverImpl*verwendet die Java API von lp_solve[2],

[2]`http://lpsolve.sourceforge.net/5.5/Java/README.html`

die über eine Bibliothek zur Verfügung steht. Damit ist es möglich, lp_solve aus dem eigenen Code aufzurufen und sämtliche Funktionen zu steuern.

Zusätzlich besteht mit der Klasse *LPFMLLinearSolverImpl*die Möglichkeit, das lineare Programm als Datei zu speichern und separat zu lösen. Das hat als einzigen Vorteil, dass die Installation eines Optimierers nicht vorausgesetzt werden muss. Dafür sah das Format LPFML[3] vielversprechen aus. Dies ist ein XML Schema, das sowohl die Formulierung des LP, als auch dessen Lösung in einer Datei speichern kann.

Laut Dokumentation unterstützt lp_solve das Lesen und Schreiben des Formats. Allerdings wurde die Voraussetzung, dass dieses Dateiformat auch korrekt geschrieben werden kann, nicht erfüllt. Ein Beispiel, das falsche Daten in der LPFML-Datei erzeugt, ist folgendes LP:

$$\begin{aligned} \texttt{min} - 2a \quad & -3b + 2c - 3d - 3e \\ \texttt{s.t.} + a \quad & +2b + c \ +4d + 2e = 8; \\ & + 0.5a + 4b + 3c \ + d + 2e \geq 3; \end{aligned}$$

Es wird zwar korrekt gelesen und die Lösung

$$\begin{array}{ccccc} a & b & c & d & e \\ 8 & 0 & 0 & 0 & 0 \end{array}$$

auch korrekt auf die Kommandozeile ausgegeben. Allerdings schreibt lp_solve 5.5.0.13 ein falsche Lösung

$$\begin{array}{ccccc} a & b & c & d & e \\ 8 & 4 & 8 & 0 & 0 \end{array}$$

in die Datei zurück. Hier bestünde also die Möglichkeit falsche Lösungen aus der LPFML-Datei zu lesen.

Beim Testen der Anbindung ist ein interessantes Problem aufgetreten. Dazu ein Rückblick auf das Beispiel der überlappenden Interferenz aus Kapitel 3.2 und dem dazugehörigen linearen Programm. Dessen allgemeine Form ist:

$$\begin{aligned} \text{min.} \quad & \left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3} \right) s_2^{(1)} + \left(\frac{1}{R_2 - r_2 - r_3} - \frac{1}{R_3 - r_3} \right) s_3^{(2)} \\ \text{s.t.} \quad & 0 \leq s_2^{(1)} \leq b_2 + r_2 T_1 \\ & 0 \leq s_3^{(2)} \leq b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2} \right) \end{aligned}$$

mit der graphischen Darstellung in Abbildung 4.4.

Zur Veranschaulichung wird das Programm graphisch gelöst [HK06]. Wenn die Werte der Koeffizienten der Zielfunktion vorliegen, kann diese als Gerade in die Darstellung eingezeichnet

[3] `http://gsbkip.chicagogsb.edu/fml/fml.html`

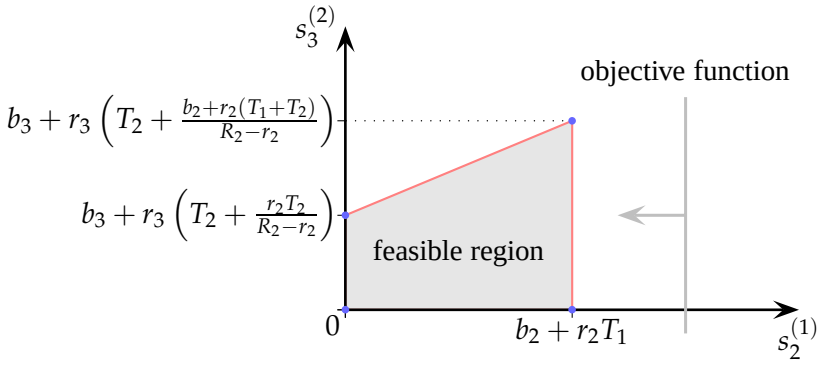


Abbildung 4.4: Solving a linear program the graphical way.

werden. Über eine Parallelverschiebung werden die Punkte aus dem Bereich der zulässigen Lösungen bestimmt, die sich mit der Zielfunktion schneiden. Die Richtung, aus der die Zielfunktion zu dem konvexen Polyeder hin verschoben wird, bestimmt sich aus den jeweiligen Vorzeichen der Koeffizienten und der Art des Optimierungsproblems, hier also durch die Minimierung. Der Hauptsatz der linearen Optimierung besagt anschaulich: wenn ein lineares Programm optimale Lösungen besitzt, gibt es darunter auch Eckpunkte.

Das Problem tritt auf, wenn einer der beiden Koeffizienten der Schlupfvariablen gleich Null wird. In Abbildung 4.4 ist der Fall für $\frac{1}{R_2 - r_2 - r_3} = \frac{1}{R_3 - r_3}$ und dem daraus resultierenden linearen Programm $\left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3}\right) s_2^{(1)}$ eingezeichnet. Für die Richtung der Parallelverschiebung ist $\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3} < 0$ angenommen worden. Um die Zielfunktion zu minimieren, muss also $s_2^{(1)}$ maximiert werden. Für diesen Fall lassen sich nun unendlich viele zulässigen Lösungen im Intervall $0 \leq s_3^{(2)} \leq B_3^{(2)}$ finden, wie das zugehörige lineare Programm zeigt:

$$\begin{aligned} \text{min.} \quad & \left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3}\right) s_2^{(1)} \\ \text{s.t.} \quad & 0 \leq s_2^{(1)} \leq b_2 + r_2 T_1 \\ & 0 \leq s_3^{(2)} \leq b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2}\right) \end{aligned}$$

Bislang ist das noch korrekt. Nun tritt aber ein Effekt auf, der bei lp_solve zu dem Problem führt. Es hat Schwierigkeiten mit unendlichen Möglichkeiten umzugehen und liefert nur den minimalen Wert für $s_3^{(2)} = 0$ zurück. $s_2^{(1)}$ wird korrekt auf $b_2 + r_2 T_1$ gesetzt.

Physikalisch interpretiert würde die Situation bedeuten, dass es einen Knoten gibt, an dem die Möglichkeit besteht, dass eine Burstiness zwischen 0 und $B_3^{(2)}$ frei wählen kann. Hier keine Burstiness weiterzugeben würde aber dem Worst-Case-Szenario widersprechen. Ein Datenstrom würde theoretisch sofort und unendlich schnell bearbeitet werden. Diese Wahl wirkt

sich aufgrund der Abhängigkeit der funktionalen Nebenbedingungen auf das ganze Szenario aus. Zu den in Kapitel 3.2 formulierten Fallunterscheidungen für die Wahl der Schlupfvariablen ergeben sich auch gravierende Widersprüche. Daher muss zwangsläufig eine beliebig wählbare Burstiness, also mit Koeffizienten Null, auf den maximalen Wert gesetzt werden, um eine korrekte Berechnung zu gewährleisten.

Eine Lösung des Problems ist, die funktionale Abhängigkeit der Nebenbedingung zu ändern. Statt $0 \leq s_{i,j}^{(k)} \leq B_{i,j}^{(k)}$ muss nur $s_{i,j}^{(k)} = B_{i,j}^{(k)}$ gesetzt werden, womit die korrekte Berechnung gewährleistet wird. Eine korrekte Formulierung des linearen Programms wäre also:

$$\begin{aligned} \min. \; & \left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2 - r_3} \right) s_2^{(1)} \\ \text{s.t. } & s_2^{(1)} \leq b_2 + r_2 T_1 \\ & s_3^{(2)} = b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2} \right) \\ & s_2^{(1)} \geq 0 \end{aligned}$$

4.4 Optimierungsmethoden

Wie schon in Abbildung 3.5 dargestellt, ist die Berechnung bei mehreren Knoten und Kurvensegmenten nur noch mit maschineller Unterstützung praktikabel. Zwar wäre es möglich, die Parameter des Modells so zu dimensionieren, dass diese Problemgröße nicht mehr erreicht wird. Dies wäre aber nur eine Umgehung des Problems. Durch bloße Vermeidung der Komplexität würden sich durch diese Abschätzung auch Fehler oder zumindest Differenzen zu den eigentlichen Werten ergeben. Zwar könnten damit Tight Bounds berechnet werden. Aber inwieweit diese aufgrund der aufsummierten Fehler aussagekräftig wären, müsste überprüft werden.

In dieser Diplomarbeit liegt der Fokus darauf, mit einem unveränderlichem Szenario zu rechnen, aber die Dimension des Problems stark einzuschränken. Hierzu gilt es, verschiedene Suchalgorithmen zu implementieren und zu vergleichen.

Bislang wurden viele Algorithmen und Ansätze entwickelt, die es möglich machen, einen großen Suchraum nach Objekten zu durchsuchen, so dass sich ein Verfahren um Größenordnungen optimieren lässt.

Mehrere Suchverfahren wurden in Betracht gezogen und deren Anwendbarkeit auf das spezifische Problem untersucht. Der Vollständigkeit halber werden hier zwei Algorithmen umrissen, die in der näheren Auswahl waren, aber nicht realisiert wurden.

Der erste ist ein *genetischer Algorithmus*[4], dessen Idee auf der Evolutionstheorie beruht. Dafür wird eine Populationsmenge aus Individuen bestimmt, mit denen der Algorithmus initialisiert wird. Aus dieser Menge werden geeignete Kandidaten selektiert, die optimale Lösungen versprechen. Deren Chromosome werden leicht mutiert oder gekreuzt und somit zur neuen Generation. Dieses Verfahren wird solange auf die aktuelle Generation angewandt, bis eine optimale Lösung oder ein Abbruchkriterium erreicht ist. Für die Wahl der Chromosomen oder des Verfahrens zur Reproduktion wird aber eine sehr genaue Kenntnis des Suchraums vorausgesetzt und nur kleine Veränderungen an diesen Parametern können starke Auswirkungen auf den gesamten Algorithmus haben. Auch die Größe der Anfangspopulation ist ein wichtiger Faktor.

Auch der nächste Algorithmus, *Simulated Annealing (Simulierte Abkühlung)*[5], nimmt ein physikalisches Phänomen zum Vorbild. Über die Abkühlungsgeschwindigkeit von Flüssigkeiten kann auf deren Kristallstruktur Einfluss genommen werden. Durch sehr rasche Abühlung ergeben sich chaotische Systeme. Langsames Abkühlen führt dagegen zu gleichmäßig geordneten Strukturen, da die Moleküle energetisch minimale Strukturen bilden können. Übertragen in einen Algorithmus bedeutet dies, dass aktuelle Lösungen eines Suchraums durch zufällige Lösungen aus deren näherer Umgebung ersetzt werden. Diese nähere Umgebung wird durch eine Wahrscheinlichkeit gewählt, die wiederum mit den beiden zugehörigen Funktionswerten und einem globalen Parameter T parametrisiert ist. T gibt die Temperatur an, die aktuell in dem System herrscht, und wird schrittweise verringert. Die Willkürlichkeit, mit der sich die Lösungen ändern, sinkt mit abnehmender Temperatur immer mehr. Die Güte des Algorithmus hängt ab von der Wahl der Anfangstemperatur, der Abkühlungsgeschwindigkeit und wie viele Ersetzungen pro Temperatur gemacht werden.

In Anbetracht der Tatsache, dass hier bei den meisten Szenarien nur ein minimaler Teil durchlaufen werden kann, wird oft die Überlegung vorherrschen, wie lange ein Durchlauf dauern darf. Über die Laufzeit für eine Berechnung der Optimisation-Based Bounding Methode kann die maximale Anzahl an zu überprüfenden Punkten approximiert werden. Solch harte Abbruchbedingungen könnten den beiden Algorithmen auch schaden, obwohl sehr gute Ergebnisse möglich wären, da die lokalen Minima gefunden und auch wieder verlassen werden können. Deren Beherrschbarkeit durch korrekte Parametrisierung der Heuristiken kann aber schwierig sein. Das führte zum Ausschluss, bis mehr Wissen über die Struktur des Lösungsraum vorhanden ist, die es dann ermöglicht, bessere Annahmen zu machen.

In Betracht wäre eventuell auch eine Optimierungsmethode gekommen, die einem Spielbaum nachempfunden ist. Aber um die Größe des Baums zu beschränken, müsste eine Heuristik entwickelt werden, die bestimmte Kombinationen von vornherein ausschließt und somit eine sehr genaue Kenntnis des Lösungsraums voraussetzt. Die vorrangige Frage wäre hier gewesen, inwieweit sich überhaupt sinnvolle Bedingungen aufstellen lassen.

Für diese Arbeit scheint es zielführender, die Anzahl an Startwerten sehr niedrig zu halten und eine sequenziell geartete Suche durchzuführen. Die naheliegende Möglichkeit ist, die jeweiligen Optimierungsmethoden als Iterator zu realisieren. Es wird also nur gefragt, ob es

[4] `http://en.wikipedia.org/wiki/Genetic_algorithm`
[5] `http://en.wikipedia.org/wiki/Simulated_annealing`

einen folgenden Punkt gibt, oder nicht. Zusätzlich besteht die Notwendigkeit, den aktuell berechneten Funktionswert der Methode zurückzuliefern, weil die meisten Suchalgorithmen den folgenden Punkt anhand der schon berechneten Werte bestimmen. Diese Ideen fließen nun in die Implementierung ein, wie in der Abbildung 4.5 gezeigt. Die abstrakte Klasse *Decompositor*implementiert einen Teil des *Iterator*Interface. Die restlichen Methoden werden von den konkreten Unterklassen *MonteCarloImpl*, *BruteForceImpl*und zu guter Letzt von *HookeJeevesImpl*implementiert. Jede der Spezialisierungen von *Decompositor*erhält den gesamten Satz

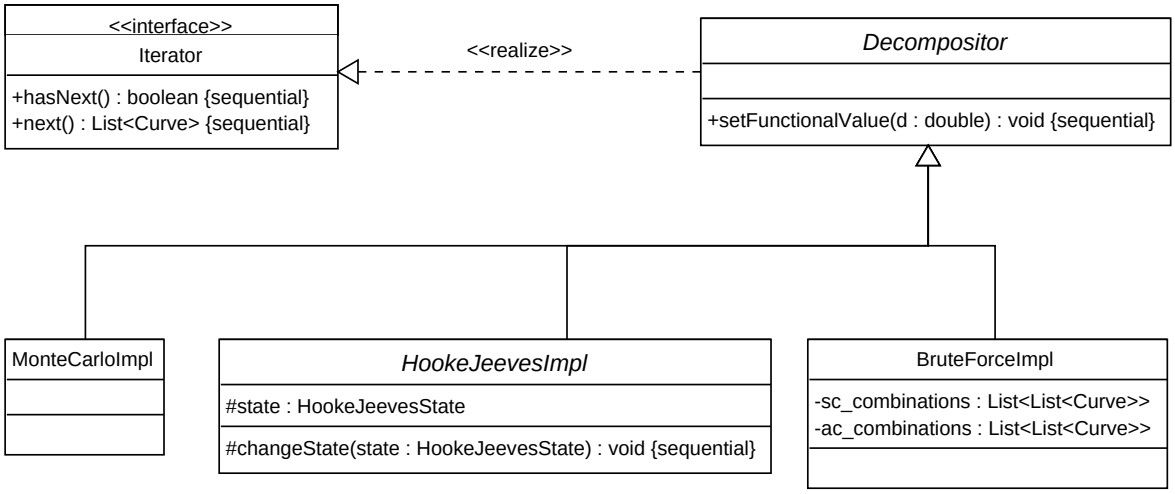


Abbildung 4.5: Dependency and inheritance diagram for search algorithms.

an stückweise linearen konkaven Arrival und konvexen Service Curves und kann diese für die eigenen Bedürfnisse aufbereiten.

Um die Anzahl der betrachteten Kombinationen begrenzt zu halten, wurde eine obere Schranke definiert. Die Formel dafür lautet

$$\lceil \text{interfering flows} * \log_1 0(\text{combinations}) \rceil$$

Um diese Funktion zu begrenzen, wurde eine Kappungsgrenze von 43200 verwendet. Da die „Brute Force“ Methode alle Kombinationen durchlaufen soll, hält sie sich nicht an eine obere Schranke.

In den folgenden Unterkapiteln werden die einzelnen umgesetzten Algorithmen vorgestellt. Kurz zuvor nochmals eine Rekapitulation der Zerlegung aus Proposition 1. Jede der n stückweise linearen konkaven Arrival Curves $\alpha_i = \bigwedge_{k_i=1}^{n_i} \gamma_{r_{k_i},b_{k_i}}$ wird dabei in n_i Token-Buckets, $i = 1,\ldots,n$ zerlegt werden. Die m stückweise linearen konvexen Strict Service Curves $\beta_j = \bigvee_{l_j=1}^{m_j} \beta_{R_{l_j},T_{l_j}}$ werden in m_j Rate-Latencies, $j = 1,\ldots m$ zerlegt. Ein Punkt aus dem Suchraum wird also mit $\left(\beta_{R_{l_1},T_{l_1}},\ldots,\beta_{R_{l_m},T_{l_m}},\gamma_{r_{k_1},b_{k_1}},\ldots,\gamma_{r_{k_n},b_{k_n}}\right)$ bestimmt. Für die Zuordnung der Datenströme zu den Arrival Curves wird im Weiteren $F_{1,1} \rightarrow \alpha_1, \ldots, F_{1,m} \rightarrow \alpha_m$, $\ldots, F_{2,2} \rightarrow \alpha_{m+1}, \ldots, F_{2,m} \rightarrow \alpha_{2m-1}, \ldots, F_{m,m} \rightarrow \alpha_n$ angenommen.

4.4.1 Brute Force

In der Klasse *BruteForceImpl*ist eine sequentielle Suche realisiert. Dabei handelt es sich zwar um ein Suchverfahren, aber nicht um ein Optimierungsverfahren. Hier werden sämtliche Kombinationen durchlaufen, was bei kleineren Suchraumgrößen aber sinnvoll ist, um so einen Anhaltspunkt für die Güte der anderen Algorithmen zu erhalten. Diese Klasse implementiert die Proposition 1.

In der Implementierung werden zuerst Listen für Arrival und Service Curves gebildet. Jede der n Arrival Curves wird in einzelne Token-Buckets zerlegt und eine Liste über sämtliche Kombinationen gebildet. Entsprechende Behandlung gilt für die Service Curves und deren Rate-Latencies. Ab einer bestimmten Problemgröße wird diese Methode aber zwangsläufig zu Speicherproblemen führen, da ein Teil der Kombinationen schon im Speicher vorliegt.

4.4.2 Monte Carlo

Die Monte-Carlo-Methode hat den großen Vorteil, dass sie auf extrem großen Mengen sehr schnell Ergebnisse liefern kann. Dies kann bei den hier behandelten Größenordnungen von Vorteil sein. Die Güte bleibt aber zu beobachten. Ebenfalls von Vorteil ist, dass der Rechenaufwand, also die Durchläufe oder Punkte des Suchraums, sowohl zeitlich als auch in der Anzahl begrenzt werden kann. Es wird also kein komplexer Algorithmus unter- oder abgebrochen werden müssen, um den Aufwand zu beschränken.

Der Nachteil des Monte-Carlo-Verfahrens ist jedoch, dass es absolut kein Gedächtnis hat. Abhängigkeiten zwischen Werten werden weder erkannt noch genutzt. Dies könnte sich aber auch zu einem Vorteil entwickeln, da er sich „ohne Vorurteil“ auf die Suche macht und nicht von möglichen Fehlinterpretationen durch falsche Heuristiken in die Irre führen lässt. Daher kann man Monte Carlo als Vergleichsmethode sehen, mit der sich höher entwickelte Algorithmen messen müssen.

Die Implementierung des Monte Carlo Algorithmus in der Klasse *MonteCarloImpl*lässt sich in einem Satz beschreiben: Um einen Punkt aus dem Suchraum zu wählen, werden sämtliche Arrival und Service Curves der Reihe nach durchlaufen und für jede wird zufällig ein Segment ausgewählt. Als Zufallszahlen-Generator wird eine Java-Implementierung des Mersenne-Twister-Algorithmus benutzt, der als *Seed* das aktuelle Datum und die Uhrzeit übergeben wird.

Um die Anzahl der betrachteten Kombinationen begrenzt zu halten, wurde eine obere Schranke definiert. Die Formel dafür lautet

$$\lceil \text{interfering flows} * \log_1 0(\text{combinations}) \rceil$$

Des weiteren wurde eine Kappungsgrenze von 43200 verwendet.

4.4.3 Hooke-and-Jeeves Direct Search

Der nächste Suchalgorithmus ist die „Direkte Suche“ von Hooke-and-Jeeves [HJ61]. Dessen Vorgehen besteht darin, das beste Ergebnis aus einer Sequenz von Testpunkten zu wählen. Die Strategie, einie Sequenz an Punkten aufzubauen, wir als Funktion aus den vorherigen Ergebnissen realisiert. Hierzu nun ein kleiner Überblick auf den Algorithmus und dessen Implementierung.

Ziel des Hooke-and-Jeeves Suchalgorithmus ist, eine Funktion $S(\phi)$ mit mehreren Variablen $\phi = (\phi_1, \phi_2, \ldots, \phi_K)$ zu minimieren. Die Variablen können als K-dimensionaler Raum aufgefasst werden, auf dem der Algorithmus arbeitet. Für die Strategie werden die Dimensionen so lange verändert, bis sich ein Optimum einstellt. Hier gilt es, den Delay $\beta^{1.\circ.}_{\{k_i\},\{l_i\}}$ zu minimieren.

Der Algorithmus gliedert sich in zwei wesentliche Phasen. Eine Phase betrifft die Erlangung einer Kenntnis über die zu optimierende Funktion. Hierzu wird eine Exploration in der Umgebung eines Punktes durchgeführt. Die so gewonnene Kenntnis fließt in ein Bewegungsmuster, das einen Anhalt für die Richtung bietet, in der ein Optimum erwartet werden kann. Die zweite Phase wendet dieses Muster auf den aktuellen Punkt an. Die Punkte, auf die das Muster angewendet werden, werden als Basispunkte bezeichnet. Nachdem ein Muster auf den aktuellen Basispunkt angewendet wird, erfolgt jeweils eine neue Exploration, so dass das momentane Muster ständig überarbeitet wird. Dadurch ergeben sich im Laufe des Algorithmus eine Reihe von Basispunkten, wobei der aktuelle Punkt B_r das bislang minimale Ergebnis erzielt hat.

Die Exploration läuft ab, wie in Abbildung 4.6 als Diagramm dargestellt. Der Routine wird ein Basispunkt $B_r = \phi$ und der dazugehörige funktionale Wert $S(B_r) = S$ übergeben. Eine einfache Strategie um neue Bewegungsmuster und Punkte zu entdecken verändert nur jeweils eine Variable oder Dimension ϕ_k, $k = 1, \ldots, K$, durch inkrementieren oder dekrementieren. Dadurch wird bestimmt, ob sich der Einfluss dieser Variablen in einem besseren Ergebnis $S(\phi)$ niederschlagen würde. Wenn dies der Fall ist, so wird diese Variablenänderung für alle folgenden Dimensionsänderungen übernommen. Am Ende der Explorationsphase werden der erruierte Punkte P_{r+1} und der Funktionswert S zurückgeliefert. Falls keine bessere Lösung gefunden wurde, entsprechen diese den zu Anfang der Exploration übergebenen Werten.

In Abbildung 4.7 ist nun der restliche Algorithmus mit der Anwendung des Bewegungsmusters abgebildet. Ausgehend von einem willkürlich gewählten initialen Basispunkt $\psi = B_0$ und dem berechneten Funktionswert $S(\psi)$ startet nun eine Exploration. Dadurch wird ein neuer Punkt $P_{r+1} = E(S_r, B_r)$ gewählt und $S(P_{r+1}) = S$ berechnet. Sofern ein besserer Wert gefunden wurde, wird dies der neue Basispunkt $B_{r+1} = P_{r+1}$ und der Funktionswert findet seinen Weg in das Gedächtnis des Algorithmus. Das Muster, das von der Exploration gefunden wurde, wird nun erneut auf den aktuellen Basispunkt angewendet.

Dies wird solange fortgesetzt, bis sich kein kleinerer Wert finden lässt. Einerseits könnte es daran liegen, dass das globale Minimum gefunden wurde. Oder es muss ein neues Bewegungsmuster angewendet werden. Eine Vorgehensweise und gleichzeitig die Abbruchbedingung ist, die Schrittweite zu verringern, bis sie einen Schwellenwert erreicht hat. Das bedeutet, dass die

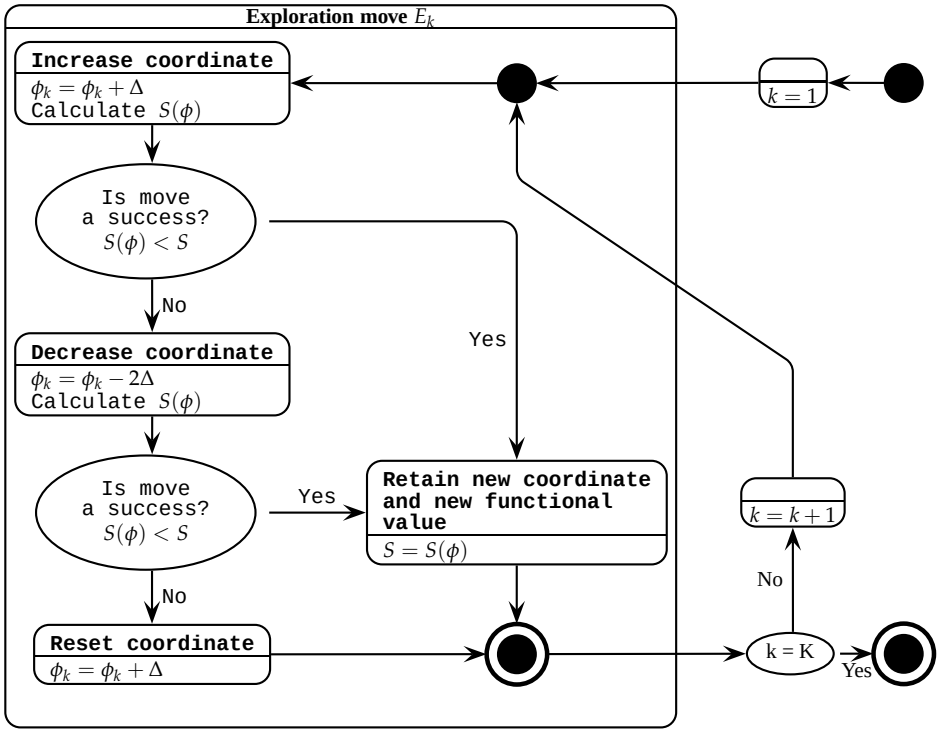


Abbildung 4.6: Detailed diagram for eploratory move [HJ61].

Explorations-Umgebung ständig verringert wird. Wenn die Anwendung des Bewegungsmusters nicht den gewünschten Erfolg zeigt, wird vom vorherigen Basispunkt eine Exploration ausgeführt. Dieser Punkt stammt aus dem Explorationsschritt, in dessen Umgebung aber nicht nach besseren möglichen Lösungen gesucht wurde. Das Beispiel in Abbildung 4.8 erklärt die Funktionsweise anschaulich.

Der beschriebene Algorithmus lässt sich allerdings nicht direkt umsetzen. Dies liegt daran, dass die Berechnung des Funktionswertes direkt aus dem Algorithmus heraus aufgerufen wird. Deswegen muss der Algorithmus so verändert werden, dass er an den Stellen unterbrochen wird, an denen die Berechnung stattfindet. Dies sind die Aufrufe von *Calculate* $S(.)$in den Abbildungen 4.7 und 4.6. Hier ist es sinnvoll, vor dem Aufruf den aktuellen Zustand des Algorithmus zu speichern und die Kontrolle an die Berechnung des Bounds abzugeben, um danach im gleichen Zustand fortzufahren. Eignen würde sich hierfür die Umsetzung als Zustand-Entwurfsmuster [GHJV95]. Eine knappe Darstellung der Abhängigkeiten und Vererbungshierarchie ist in Abbildung 4.9 zu sehen.

Während der Laufzeit muss es möglich sein, den Übergang zwischen verschiedenen Zuständen abhängig vom Verhalten zu machen. Die abstrakte Klasse *HookeJeevesImpl*definiert die allgemeine Schnittstelle um die Zugriffe auf den aktuellen Zustand zu kapseln. Die Methoden sind so definiert, dass die zustandsspezifischen Anfragen an die aktuelle Zustandsunterklasse delegiert werden.

Die abstrakte Klasse *HookeJeevesState*definiert die Schnittstelle zur Kapselung des eigentli-

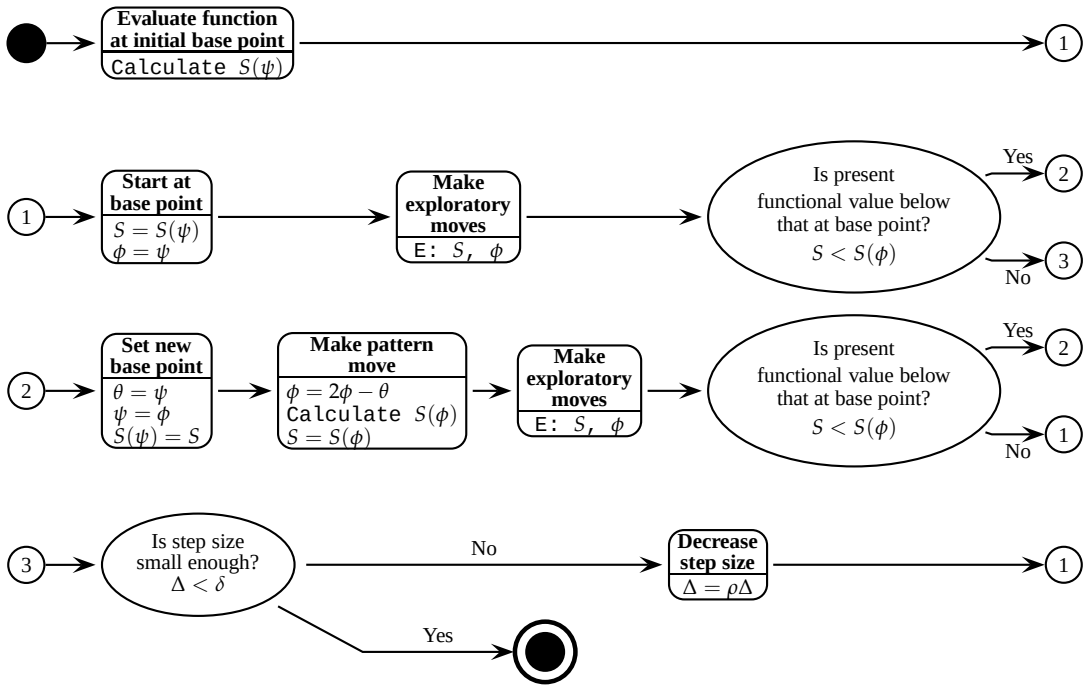


Abbildung 4.7: Detailed diagram for pattern move [HJ61].

θ	Der vorherige Basispunkt
ψ	Der aktuelle Basispunkt
ϕ	Der Basispunkt aus dem aktuellen Schritt
$S(\psi)$	Der Funktionswert am Basispunkt
$S(\phi)$	Der Funktionswert für diesen Schritt
S	Der Funktionswert vor diesem Schritt, normalerweise der kleinste Wert der Sondierungsschritte
Δ	Aktuelle Schrittweite
δ	Minimale Schrittweite
ρ	Reduktionsfaktor für die Schrittweite
ϕ_k	Eine der Koordinaten für ϕ, $k = 1, 2, \ldots, K$
K	Anzahl der Koordinaten der Punkte

Tabelle 4.1: Bedeutung der Variablen aus den Abbildungen 4.7 und 4.6 [HJ61]

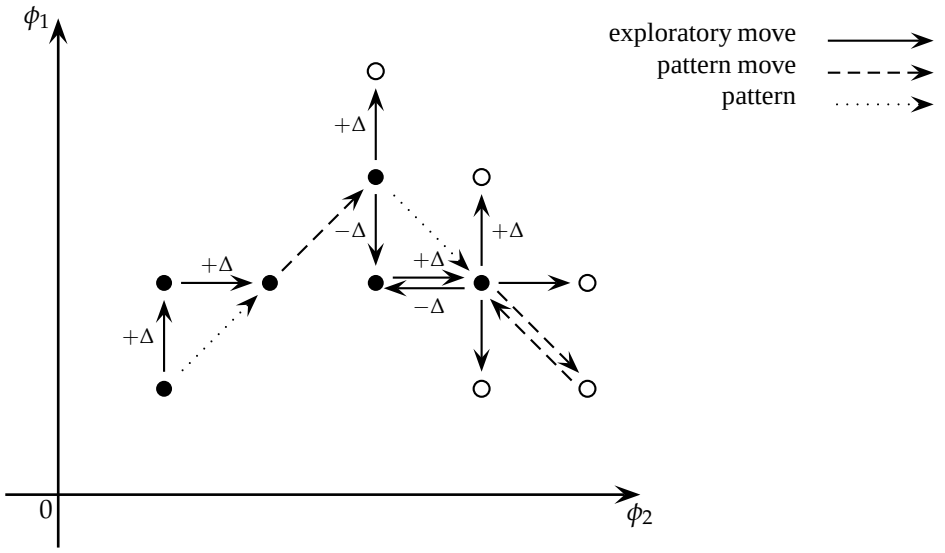


Abbildung 4.8: Example for a possible run.

chen Hooke-and-Jeeves-Algorithmus und beinhaltet deswegen die Speicherung der notwendigen Variablen. Die fünf Unterklassen implementieren das eigentliche Verhalten und sind an die Zustände des Algorithmus gekoppelt. Der Explorationsschritt aus Abbildung 4.6 wird komplett in der Klasse *HookeJeevesExploratoryMove*realisiert. Die etwas komplexere Umsetzung des in Abbildung 4.7 beschriebenen Verfahrens zur Bestimmung der Richtung des Bewegungsmusters ist in vier Zuständen realisiert. Dabei gilt folgende Zuordnung aus Tabelle 4.2.

●	*HookeJeevesInitialState*
(1)	*HookeJeevesInitialExploration*
(2)	*HookeJeevesPatternMove*
(3)	*HookeJeevesReduceStepSize*

Tabelle 4.2: Zuordnung zwischen dem Flussdiagramm und den Zustände.

Eine besondere Rolle kommt der Klasse *HookeJeevesDimension*zu. Diese dient dazu, jede einzelne der K Dimensionen von ϕ zu kapseln und darauf die Vorwärts- und Rückwärtsschritte zu definieren, sowie die Schrittweite Δ, die minimale Schrittweite δ und den Reduktionsfaktor ρ zu speichern.

Die einzelnen Dimensionswerte werden als Sequenz aufgefasst. Die sequentielle Definition und damit verbunden die Begrenzungen der Koordinaten einer Dimension führen dazu, dass die Schritte nur in einem begrenzen Raum ausgeführt werden können. So werden Fälle auftreten, bei denen manche Schritte nicht ausgeführt werden, da sie sich am Rande dieses Raumes

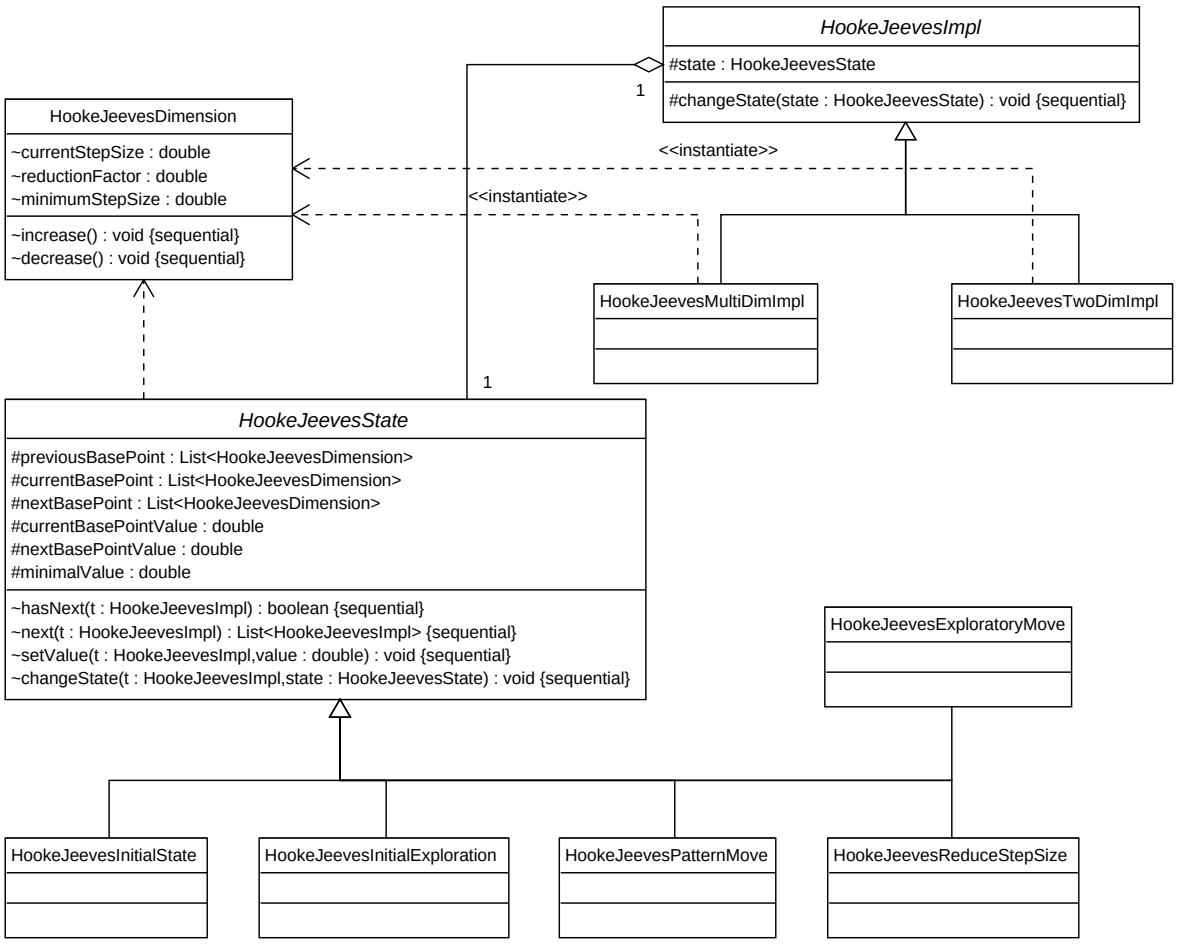


Abbildung 4.9: Dependency and inheritance diagram for Hooke and Jeeves algorithm.

bewegen. Eine zyklische Definition wäre auch möglich gewesen. Allerdings hätte hier das Problem bestanden, dass durch die Schrittweite sowohl der Vorwärts- als auch der Rückwärtsschritt die gleiche Koordinate ergeben hätten.

Es stellt sich die Frage, wie die Arrival und Service Curves auf die Dimensionen aufgeteilt werden können. Jede einzelne Kurve als Dimension zu verwenden, macht wenig Sinn, da die Anzahl der Segmente im Vergleich zu der Anzahl der Kurven verschwindend gering sein dürfte. Ein Nutzen wäre durch den Overhead des Algorithmus nicht gegeben. Es ist also eher sinnvoll, die Anzahl der Dimensionen möglichst gering zu halten. Hier gibt es verschiedene Ansätze:

Die einfachste Möglichkeit ist, nur zwei Dimensionen zu erstellen: jeweils eine für die Arrival und eine für die Service Curves. Die Kurven werden in der Reihenfolge der Server aufgelistet, also $(\beta_1, \ldots, \beta_m)$ für die Dimension der Service Curves und entsprechend $(\alpha_1, \ldots, \alpha_n)$ für die Arrival Curves. Diese Idee ist in *HookeJeevesTwoDimImpl*umgesetzt.

Die nächste Möglichkeit ist, die Service Curves in einer eigenen Dimension $(\beta_1, \ldots, \beta_m)$ zu belassen, die Arrival Curves aber in mehrere Dimensionen zu gliedern. Die Nachbarschaft der Kurven definiert sich über den Eintrittsknoten. Damit ergeben sich m Dimensionen $(\alpha_1, \ldots, \alpha_m)$, $(\alpha_{m+1}, \ldots, \alpha_{2m-1})$, …, (α_n). Da in jeder Dimension die Arrival Curves nach aufsteigender Länge sortiert sind, kommt den längeren Flüssen ein größeres Gewicht zu, denn sie weren bei den Schritten am häufigsten verändert. Die progammatische Umsetzung ist in *HookeJeevesMultiDimImpl*realisiert.

Die letzte Möglichkeit ist, die Arrival Curves nach den Servern zu gruppieren, über die die entsprechenden Datenströme laufen. Der Datenstrom $F_{1,m}$ mit α_m ist also in allen Dimensionen vorhanden. Dessen Änderung wirkt sich also auch auf die anderen Dimensionen aus. Eine schon veränderte Koordinate ändert sich erneut und der Explorationsschritt beschreibt dadurch eine Diagonalbewegung. Damit steht dieses Vorgehen im Gegensatz zu dem originalen Vorgehen, das davon ausgeht, dass sich nur eine Dimension auf einmal verändert. Diagonalbewegungen werden von dem Algorithmus aber schon unterstützt, weil die Dimensionen nacheinander variiert werden. Es besteht die Befürchtung, dass der Einfluss einer Dimension nicht mehr richtig erkannt wird, wenn mehrere Dimensionen zugleich verändert werden. Ein zusätzlicher Effekt ist, dass die Überprüfung bestimmter Punkte der Nachbarschaft nicht mehr stattfindet. Ein komplexeres Bewegungsmuster würde eine komplexere Parametereinstellung nach sich ziehen, weshalb auf die Implementierung untereinander abhängiger Dimensionen verzichtet wurde.

5 Ergebnisse

Bisher wurde die neue Optimisation-Based Bounding Methode nur aus formaler und Implementierungssicht betrachtet. Im nun folgenden Kapitel wird diese nun zu den bisherigen Methoden SFA und PMOO-SFA verglichen. Dafür werden verschiedene Szenarien berechnet, und die Ergebnisse verglichen.

Sämtliche Versuche wurden auf einem Cluster mit sechs Knoten durchgeführt. Jeder Knoten ist mit zwei Intel® Xeon® Quad Core E5420 CPUs mit 2.5GHz bestückt. Der Knoten, der die Serverfunktionalität innehat, ist mit 16GB Speicher und ca. 15GB Swap ausgerüstet und läuft mit Linux cluster 2.6.24-19-server. Die restlichen haben je 12GB Speicher und 12GB Swap und laufen mit Linux version 2.6.24-19-generic. Auf dem Cluster ist Java™ SE Runtime Environment (build 1.6.0 07-b06) installiert und der Bytecode des DISCO Network Calculators lief mit Java HotSpot™ 64-Bit Server VM (build 10.0-b23, mixed mode). Für die Ausführung wurde Java die Parameter *-Xms256m -Xmx3328m -XX:+UseConcMarkSweepGC -XX:-UseGCOverheadLimit* übergeben. Die linearen Programme wurden mit lp_solve 5.5.0.10 gelöst.

Sämtliche Berechnungen legen das Modell des allgemeinen Netzwerks aus Abbildung 3.4 zu Grunde. Eine allgemeine Topologie geht natürlich noch weiter und der Querverkehr des betrachteten Datenstroms muss ebenfalls in Betracht gezogen werden. Da es hier um Worst-Case Betrachtungen geht, wird ein voll besetztes Netzwerk betrachtet. Jeder Datenstrom $F_{i,j}$ wird also ungleich Null angenommen. Dies entspricht nicht unbedingt einem realen Hintergrund, dafür würde sich probabilistisch gewählte Datenströme eignen oder real erhobene Daten verwenden. Ein nicht vollbesetztes Netzwerk wäre jedoch ein wichtiger Faktor, der die Ergebnisse beeinflussen würde. Viele Versuche wären notwendig, den Einfluss dieses Faktors untersuchen zu können. Der hier betrachtete Aufbau hat vielmehr theoretischen, aber allgemeinen Charakter, reale Szenarien brächten den Vorwurf mit sich, dass die Ergebnisse zu spezialisiert sind.

In den einzelnen Versuchsreihen werden die Werte für Service und Arrival Curves verändert, aber im entsprechenden Kapitel geschildert. Es wird aber davon ausgegangen, dass die Server genügend Service zur Verfügung haben, so dass kein Paket in Gefahr läuft, verworfen zu werden.

5.1 Versuchsreihe 1

Die erste Versuchsreihe beschäftigt sich mit der Frage, wie stark sich die berechneten Output Bounds der bisherigen Methoden *PMOO-SFA* und *SFA* von den Tight Bounds unterscheiden.

Für diese Versuchsreihe wird ein allgemeines Netzwerk mit vollbesetztem Querverkehr verwendet. Der Workload dieses Systems wird durch feste Wahl der Arrival und Service Curves bestimmt. Für die Arrival Curves werden Token-Buckets mit Rate $r = 10\text{Mbps}$ und Burst $b = 1\text{Mb}$ gewählt. Für die Service Curves Rate-Latencies mit Latency $T = 0.0001\text{s} = 0.1\text{ms}$ und einer variablen Rate, so dass die Auslastung jedes Knoten pro Versuchsaufbau 20%, 50% oder 90% annimmt. Die Anzahl der Knoten wurde zwischen $1, \ldots, 30$ variiert.

Für die Output Bound Berechnungen ist es im DISCO Network Calculator möglich, für die SFA und PMOO-SFA zwischen iterativer oder rekursiver Prozedur zu wählen. Nach anfänglichen Testläufen kam heraus, dass die iterative Version nur beschränkt tauglich war. Dies gilt insbesondere für die SFA, weil sich deren Implementierung komplett auf die Output Bound Berechnung stützt. Die PMOO-SFA dagegen besitzt ihre eigene Prozedur und greift nur ab einer bestimmten Rekursionstiefe darauf zurück. Während den Testläufen war der Speicherbedarf für die SFA so groß, dass die Versuche nur von 1 bis 21 Knoten laufen konnten.

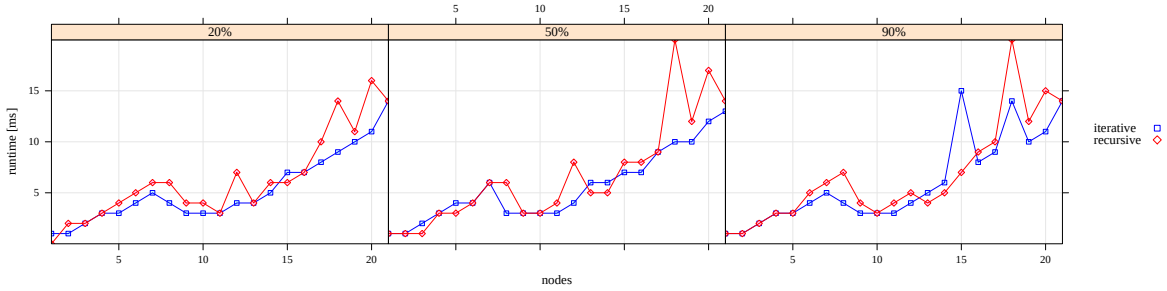
Deswegen wurde zuerst einmal eine kurze Gegenüberstellung der Iteration und Rekursion durchgeführt um die Laufzeitunterschiede zu verdeutlichen. Diese sind in Abbildung 5.1 gegeneinander aufgetragen. Bei der PMOO-SFA in 5.1a sind die Unterschiede vernachlässigbar, weil sie sich im Millisekundenbereich abspielen. Das unterstützt letztendlich auch die obige Erklärung. Die gelegentlichen Peaks sind durch Prozessorauslastung und Speicherzugriff zu erklären, da mehrere Prozesse gleichzeitig gestartet wurden. Vereinzelte Sanity-Checks konnten dies auch bestätigen.

Die Konsequenz aus der Laufzeitbetrachtung war, dass auf iterative Output Bound Berechnung verzichtet wird. Der Frage, ob sich diese Beobachtung auf komplexere Szenarien ausweiten lässt, wurde nicht weiter nachgegangen.

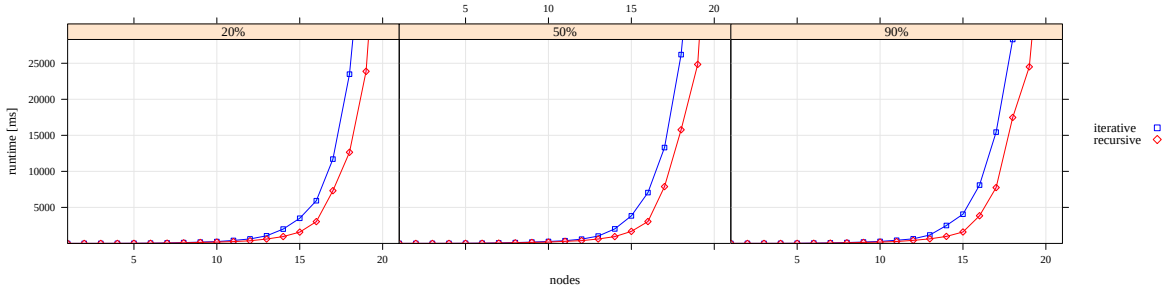
Als nächstes interessiert natürlich die Frage nach der quantitativen Verbesserung durch die Optimisation-Based Bounding Method. Aufgrund der Kurvenwahl handelt es sich um einen Tight Bound, so dass dies in den Abbildungen 5.2 und 5.3 auch durch die Bezeichnung *Tight* deutlich gemacht wurde.

Die Betrachtung des Delay Bounds zeigt keine besonderen Auffälligkeiten. Die Abbildung 5.2 zeigt den Delay Bound für jede Analysemethode, wobei die Auslastung der Server gegenübergestellt wurde. Was zu beobachten ist, ist der steigende Delay bei steigender Auslastung. Insbesondere sticht der generell stark ansteigende Delay für eine Auslastung von 90%.

Der Vergleich der Analysen untereinander ist in der Abbildung 5.3 zu finden. Hier zeigt sich ein generell besserer Delay für den Tight Bound. Die SFA liefert schon sehr schnell einen sehr schlechten Delay, wohingegen die PMOO-SFA und der Tight Bound für hohe und niedrige Auslastungen relativ nahe beieinander liegen. Interessant ist aber der Fall für 50% Auslastung. Die PMOO-SFA hat hier einen relativ geradlinigen Verlauf, der Anstieg des Tight Bound wird



(a) PMOO-SFA

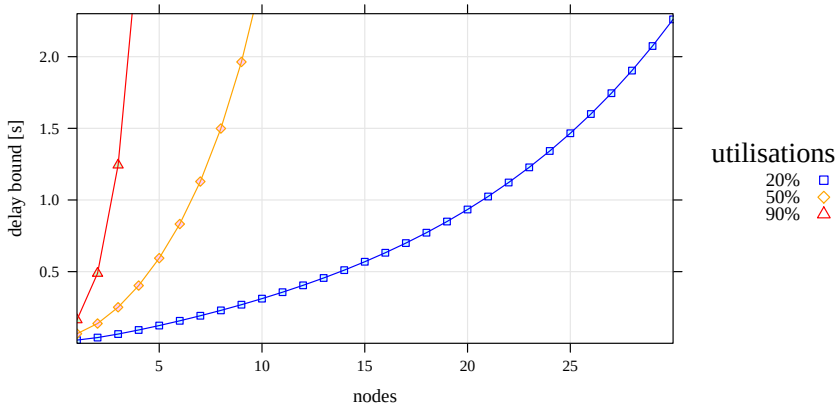


(b) SFA

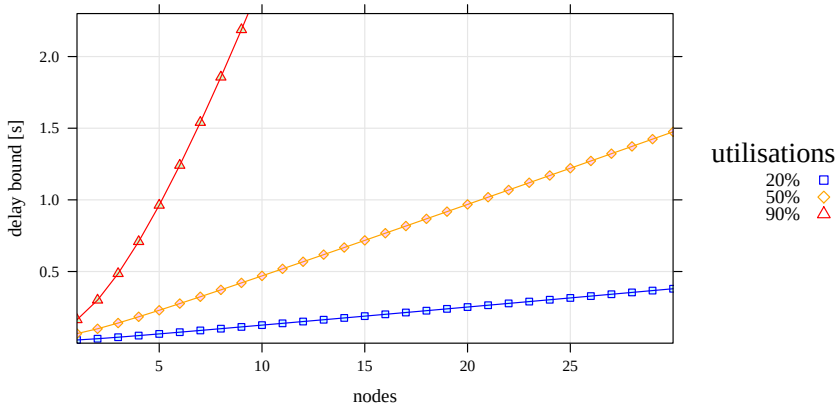
Abbildung 5.1: Runtime comparison between iterative and recursive output bound computation.

dagegen etwas schwächer. Aber auch für 20% Auslastung pro Knoten scheint der Anstieg des Tight Bound immer schwächer zu werden. Ob er sich einem Grenzwert annähert bleibt dahingestellt.

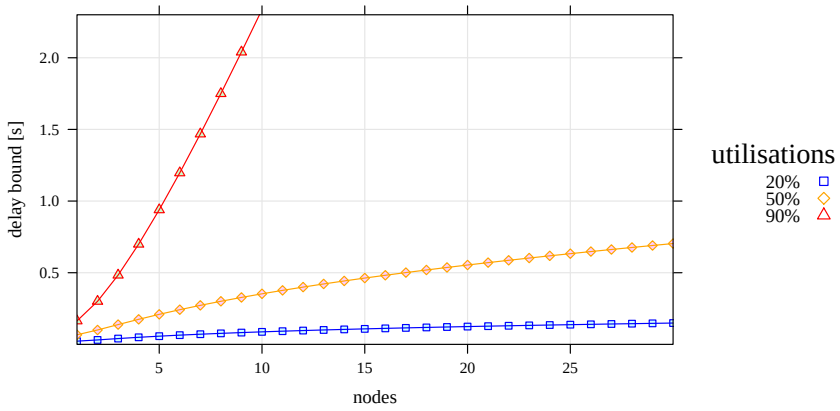
Eine genaue Laufzeitbetrachtung liefert zwei Erkenntnisse. Zum einen sind die Unterschiede zwischen den drei Auslastungswerten von 20%, 50% und 90% so gut wie nicht vorhanden, was auch nicht sonderlich überrascht. Die zugehörigen Kurven sind in Abbildung 5.4 aufgelistet. Die wesentliche Erkenntnis ist aber in Abbildung 5.5 zu sehen. Hier zeigt sich ein rapider Einbruch in der Laufzeit ab einer Pfadlänge von etwa 16 Knoten. Insbesondere die Unterschiede zwischen den einzelnen Analysemethoden werden immer stärker.



(a) SFA

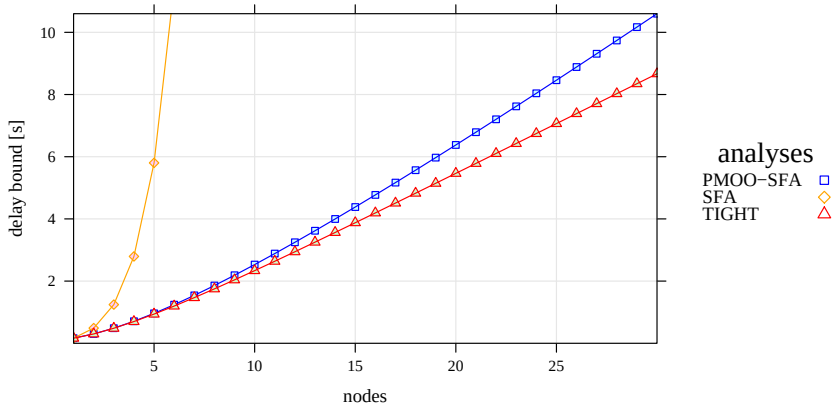


(b) PMOO-SFA

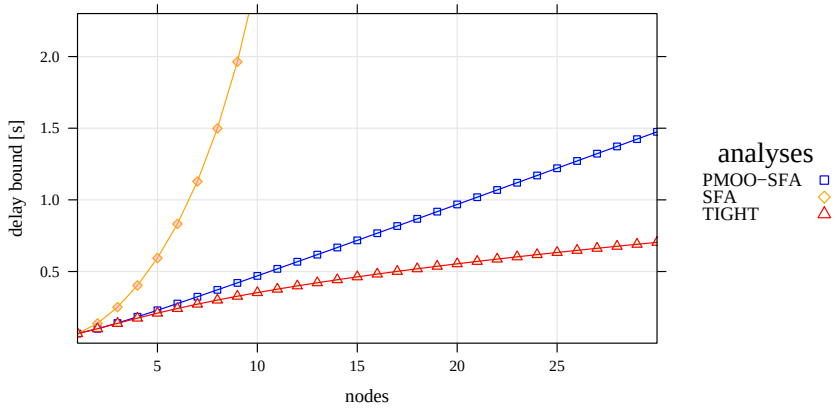


(c) TIGHT

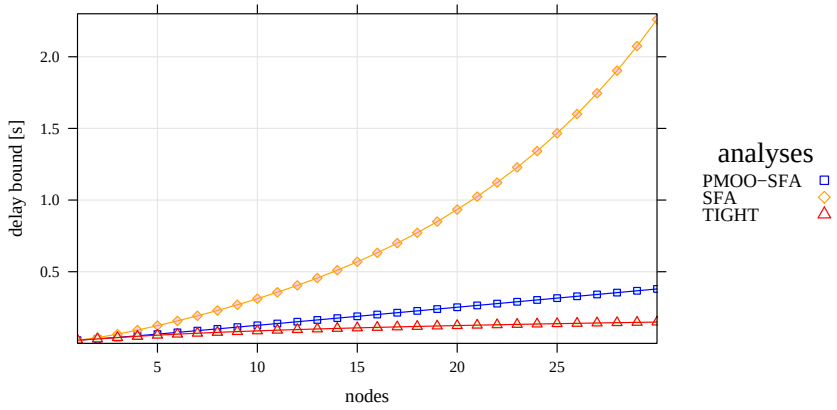
Abbildung 5.2: Delay bound comparison grouped by analysis method.



(a) 90% utilization

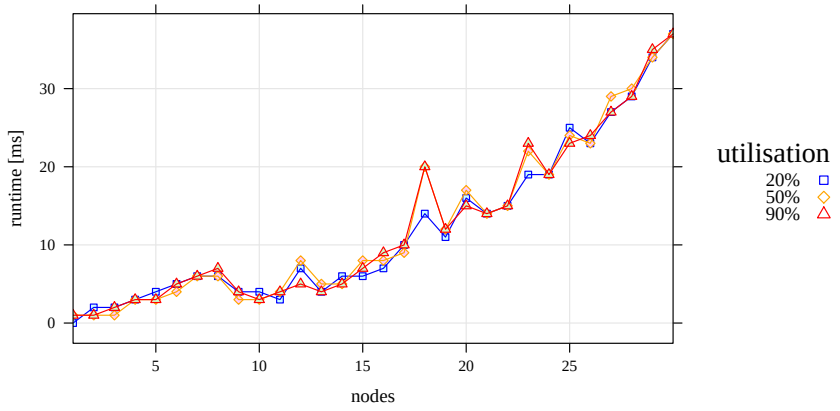


(b) 50% utilization

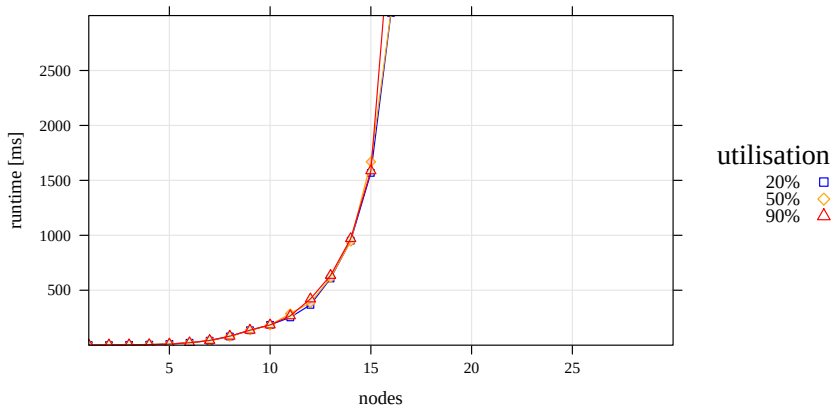


(c) 20% utilization

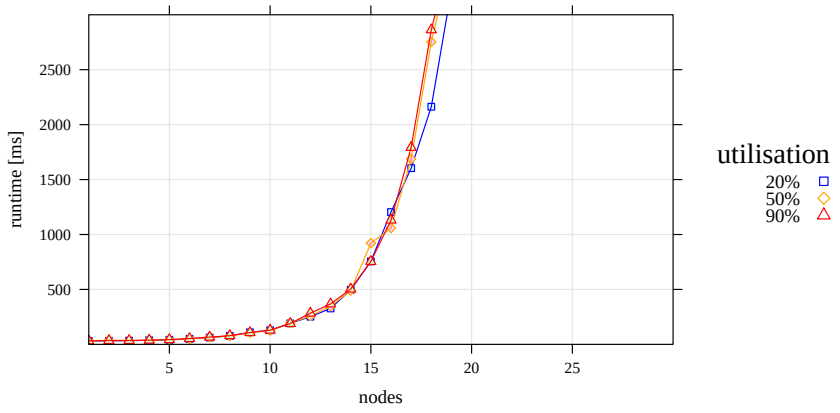
Abbildung 5.3: Delay bound comparison grouped by utilization.



(a) PMOO-SFA

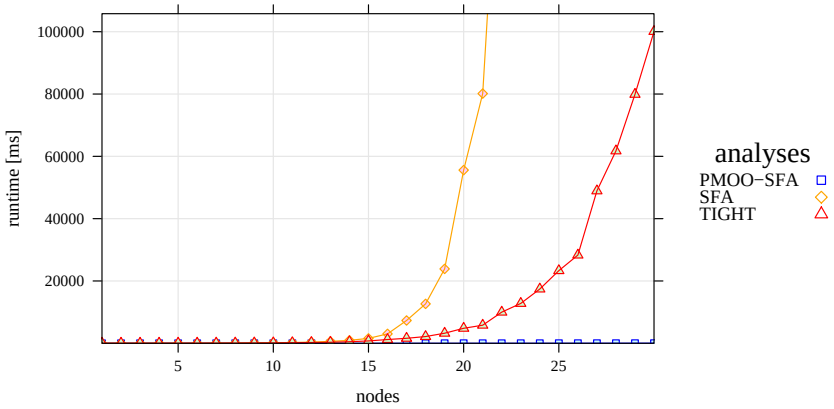


(b) SFA

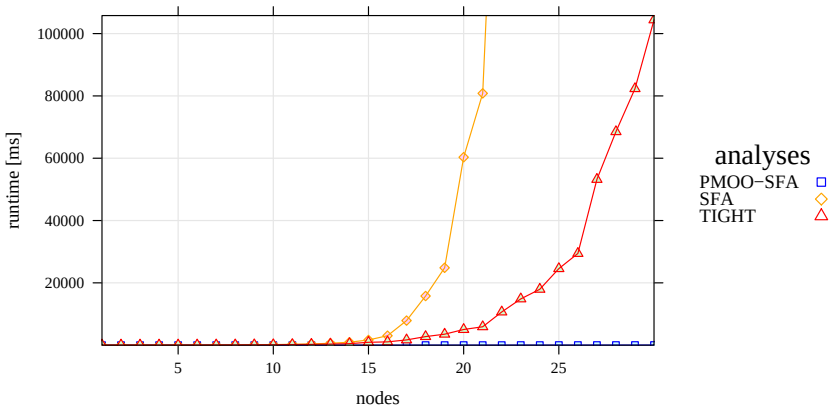


(c) TIGHT

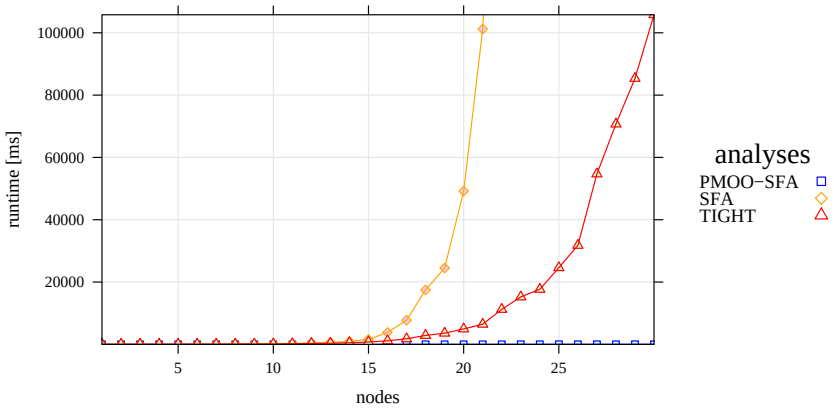
Abbildung 5.4: Runtime comparison grouped by analysis method.



(a) 20% utilization



(b) 50% utilization



(c) 90% utilization

Abbildung 5.5: Runtime comparison grouped by utilization.

5.2 Versuchsreihe 2

Die zweite Versuchsreihe beschäftigt sich mit einer genaueren Analyse des Laufzeitverhaltens. Es war wichtig, noch eine Dimensionierung zu finden, für die noch alle Kombinationen in akzeptabler Zeit berechnet werden können. Darauf aufbauend den Delay Bound und die Laufzeit zu betrachten und zu vergleichen.

Der Versuchsaufbau gestaltet sich wie folgt: ein allgemeines Netzwerk-Modell mit vollbesetztem Querverkehr wird verwendet, das maximal fünf Knoten besitzt. Die Service Curve garantiert eine Latency $T = 0.0001\text{s} = 0.1\text{ms}$ und die Rate wird so gewählt, dass sie eine Auslastung von 50% annimmt. Für die Beschränkung der Arrival Curve werden vier verschiedene Envelopes generiert, deren Anzahl an Token-Buckets wird zwischen zwei und fünf variiert. Die Auslastung der Knoten wurde konstant bei 50% belassen.

Für die Wahl der Arrival Curve wird folgender Versuchsaufbau angenommen. Ausgegangen wird von einem T-SPEC (M, p, r, b) mit einer Burst-Size $b = 1Mb$, einer maximalen Paketgröße von $M = 1500b$, einer maximalen Peak-Rate $p = 10Mbps$ und einer Sustained-Rate $r = 1Mbps$. Eine maximale Peak-Duration wird mit $2s$ festgelegt. So ergibt sich ein weiterer Parameter: die minimale Peak-Rate.

Über eine äquidistante Aufteilung des Intervalls zwischen minimaler und maximaler Peak-Duration können verschiedene T-SPECs generiert werden. Ebenso kann das Intervall zwischen minimaler und maximaler Peak-Rate aufgeteilt werden. Für jede Möglichkeit ergeben sich also bei n äquidistanten Teilen gerade $n + 1$ verschiedene T-SPECs. Für jeden so generierten T-SPEC ändert sich also nur der Token-Bucket für das Verhalten während eines Peaks. Alle Parameter und das Vorgehen sind in Abbildung 5.6 verdeutlicht.

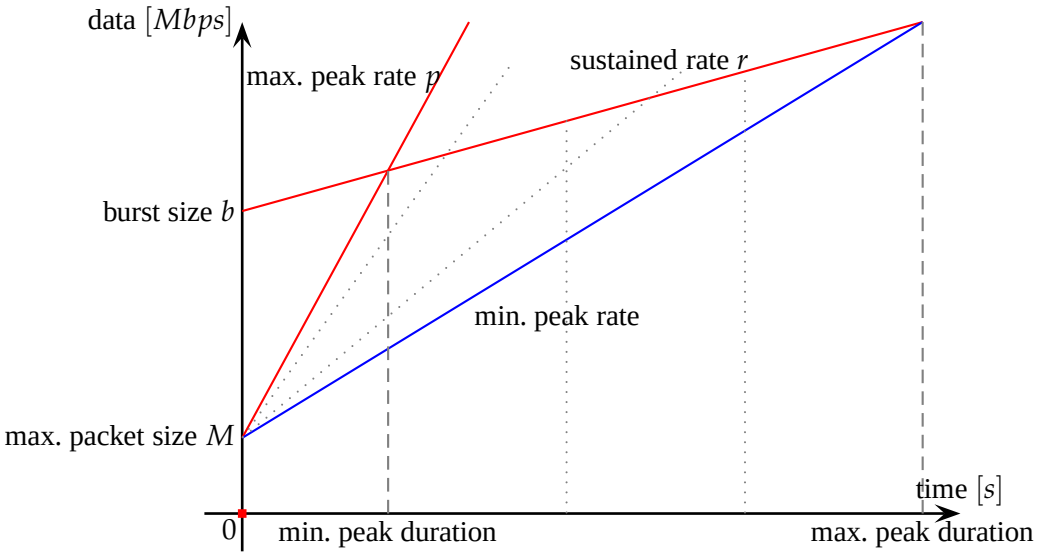


Abbildung 5.6: Parameter for the creating T-SPECs.

Um einen beliebigen Datenstrom zu erhalten, wird nun eine Überlagerung, also Addition,

verschiedener T-SPECs vorgenommen. So können die Envelopes für den Querverkehr als aus mehren Datenströmen bestehender Gesamtdatenstrom interpretiert werden. Mit einer 50-50 Wahrscheinlichkeit wird eine der beiden Aufteilungen gewählt und ein beliebiger T-SPEC gewählt. Über eine Binärbaum-Struktur werden mehrere so gewählter Envelopes aufaddiert, wie in Abbildung 5.7 gezeigt.

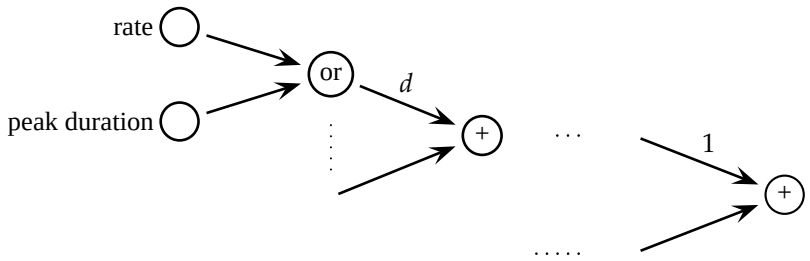


Abbildung 5.7: Using a tree structure to create a random envelope for the arrival curves.

Über die Anzahl der T-SPECs pro Menge und die Baumtiefe wird die Form der resultierenden Arrival Curve ein wenig gesteuert.

Für diese Versuchsreihe wurde das Verfahren mehrfach angewandt, um Envelopes aus zwei bis fünf Token-Buckets zu erhalten. Die Ergebnisse für die Arrival Curves gestalten sich wie folgt:

2 Envelope Paare: $\bigwedge\{\gamma_{7.47,0.06}, \gamma_{5.0,5.0}\}$

3 Envelope Paare: $\bigwedge\{\gamma_{11.20,0.06}, \gamma_{7.81,2.04}, \gamma_{5.0,5.0}\}$

4 Envelope Paare: $\bigwedge\{\gamma_{19.74,0.06}, \gamma_{10.74,1.05}, \gamma_{5.65,4.01}, \gamma_{5.0,5.0}\}$

5 Envelope Paare: $\bigwedge\{\gamma_{17.93,0.06}, \gamma_{8.93,1.05}, \gamma_{7.23,2.04}, \gamma_{6.29,3.02}, \gamma_{5.0,5.0}\}$

In der Versuchsreihe wurden 20 Experimente durchgeführt, sämtliche Kombinationen zwischen den vier verschiedenen Arrival Curves und der maximalen Anzahl von fünf Knoten. Allerdings konnten nicht alle Versuche beendet werden, da mehrere wegen Speichermangels abgebrochen wurden. Dies war bei fünf Knoten und mehr als drei Segmenten der Fall. Alle anderen liefen durch. Für die Optimisation-Based Bounding Methode wurde der „Brute Force“ Ansatz zum Durchlaufen aller Kombinationen gewählt und ist deshalb mit OBA(BF) bezeichnet.

Zuerst einmal folgt die Auswertung des Delay Bounds. In den beiden Abbildungen 5.8 und 5.9 sind die Werte graphisch dargestellt, aber nach unterschiedlichen Faktoren gruppiert. Auch hier liefert die SFA einen schlechteren Delay Bound, die anderen beiden liegen aber sehr dicht beieinander. Aus Abbildung 5.9 wird der Einfluss der Arrival Curve auf das Ergebnis deutlich.

Ein anderes Bild liefert jetzt aber die Laufzeitbetrachtung im Vergleich zu Versuchsreihe 1. Da sämtliche Kombinationen für die Optimisation-Based Bounding Methode durchlaufen wurden, ist es auch keine Überraschung, dass sie am längsten braucht. Bei Vergleich der Laufzeitwerte für PMOO-SFA und SFA kann der Schluss gezogen werden, dass sich die Erhöhung der

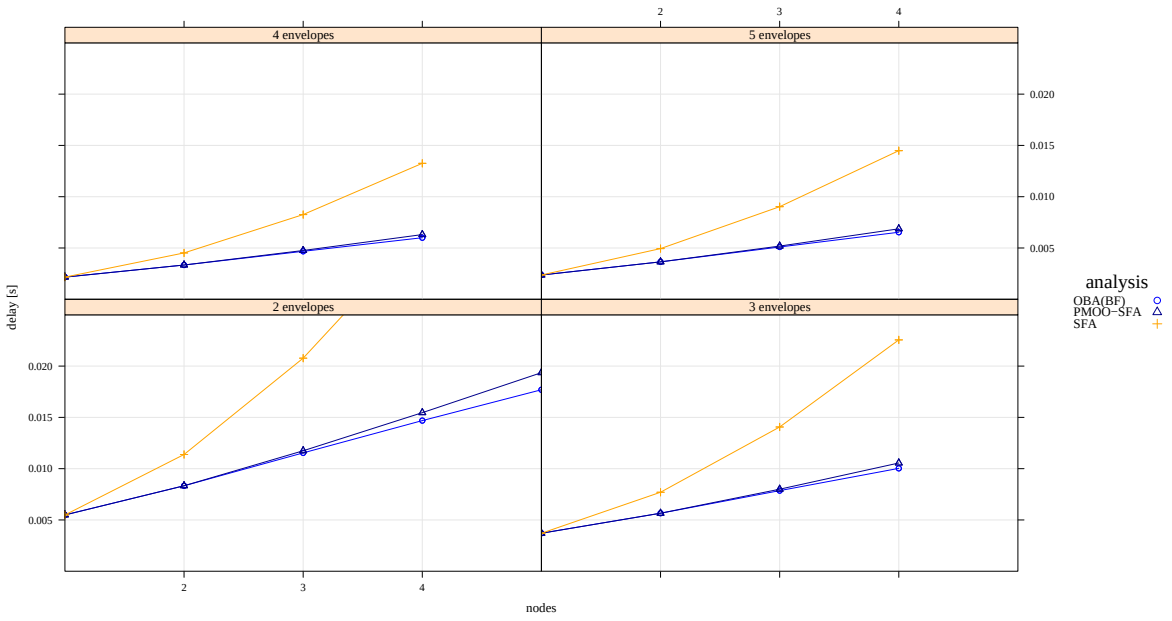


Abbildung 5.8: Delay grouped by envelopes.

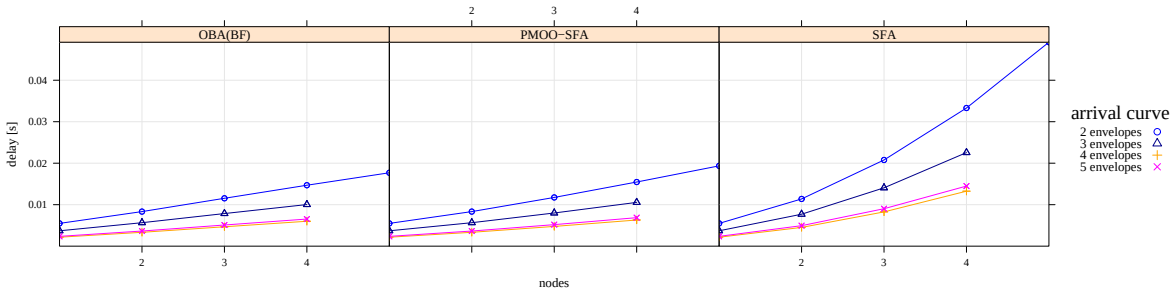


Abbildung 5.9: Delay grouped by analysis method.

Envelope Paare negativ auf die PMOO-SFA auswirkt. Aus Versuchsreihe 1 wurde ein Unterschied nur bei einer größeren Anzahl an Knoten deutlich, hier dagegen bietet sich ein anderes Bild. Die Ursache hierfür ist in der Implementierung der Analysemethoden zu suchen. Die

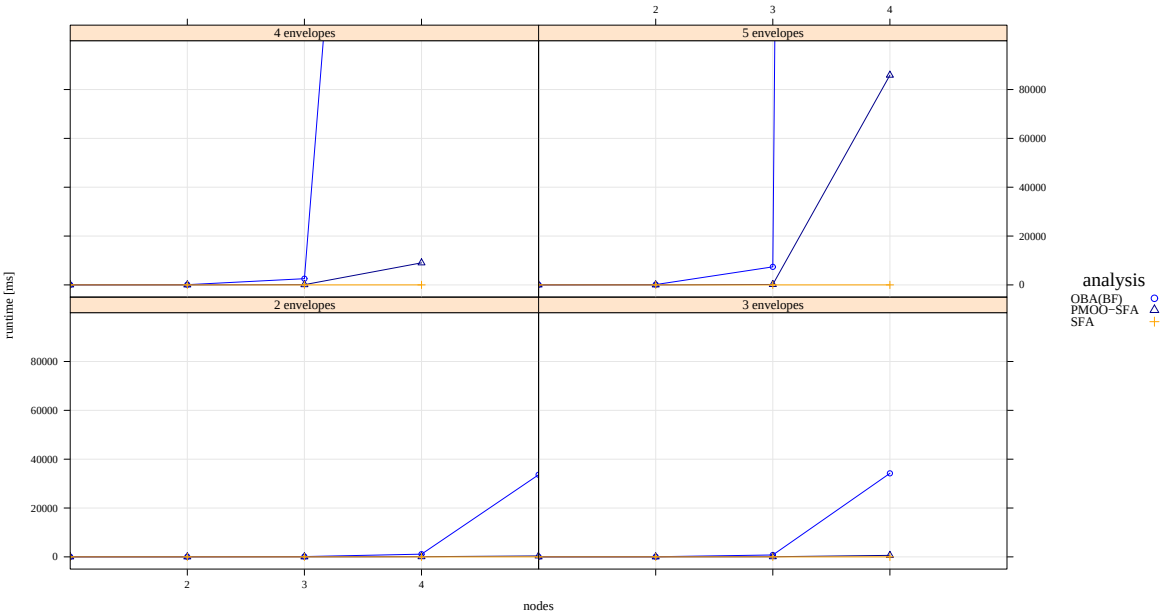


Abbildung 5.10: Runtime grouped by envelopes.

PMOO-SFA benutzt intern auch eine Zerlegung ähnlich der OBA, welche nun zu dem erhöhten Zeitbedarf führt. Die Ergebnisse sind in Abbilung 5.10 graphisch aufgetragen. Eine kleine Auswahl an sehr großen Werten wurde noch in Tabelle 5.1 zusammengefasst.

Knoten	Envelopes	SFA	PMOO-SFA	OBA(BF)
4	4	6	9030	607091
4	5	7	85880	5441641

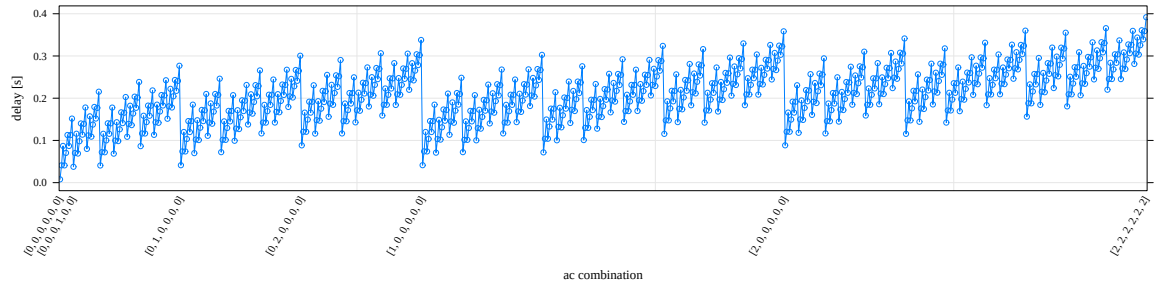
Tabelle 5.1: Ausgewählte Laufzeitwerte in Millisekunden.

Für die betrachtete Dimensionierung ließen sich noch sämtliche Kombinationen aus Rate-Latencies und Token-Buckets bilden. Mit jeder einzelnen zurückgelieferten Service Curve wird nun mit der Arrival Curve des betrachteten Datenstroms der Delay Bound berechnet. So ergibt sich ein Bild über den zu optimierenden Suchraum. Verschiedene Beispiele sind in den Abbildungen 5.11 zusammengefasst.

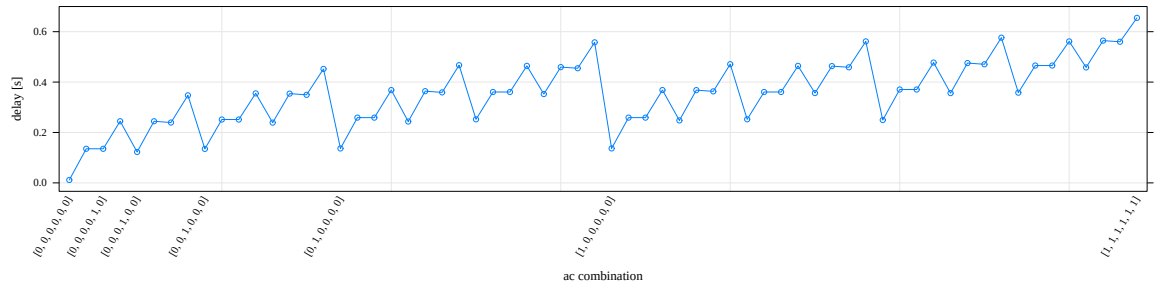
Sie ergeben ein Bild eines schwierig zu optimierenden Suchraums, weil die Werte eine Periodizität aufweisen und deshalb sehr viele lokale Extrempunkte liefern. Die Parametrisierung der Heuristik eines Suchverfahrens ist hier die Herausforderung. Andererseits liegen für die gewählten Workloads die globalen Optima gerade an den Rändern. Die Envelope Segmente

werden ja gerade nach der absteigenden Rate nach indiziert. Die Selektion der Token-Buckets mit dem geringsten Index, also der höchsten Rate, liefert gerade das globale Minimum. Der höchste Index das globale Maximum. Dieser Zusammenhang ist auch insofern logisch, da die absteigende Rate auch eine aufsteigende Burstiness zur Folge hat.

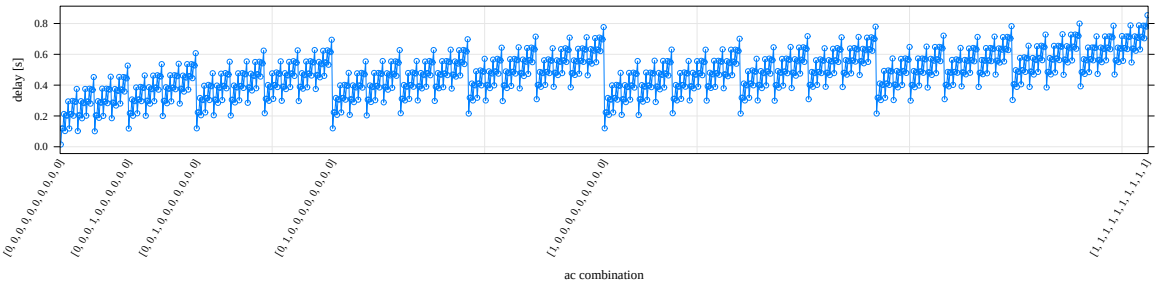
Wenn der übrigbleibende Service für den betrachteten Datenstrom berechnet wird, so wird dieser durch das punktweise Maximum aller berechneten Left-Over Service Curves gebildet. Die Proposition 1 gibt eine Abschätzung für diesen Service. Also wäre es für dieses Szenario möglich, den Envelope mit dem höchsten Index zu wählen.



(a) 3 nodes and 3 envelope pairs.



(b) 3 nodes and a T-SPEC.



(c) 4 nodes and a T-SPEC.

Abbildung 5.11: Delay bounds for all combinations of 3 nodes and a T-SPEC.

5.3 Versuchsreihe 3

Bislang wurden in den Versuchsreihen alle Kombinationen durchgespielt. Die aktuelle Versuchsreihe wird nun die implementierten Optimierungsverfahren untersuchen. Zuerst wird die Frage beantwortet, wie gut die Suchverfahren im Vergleich zu den in Versuchsreihe 2 untersuchten Analysen sind. Als nächstes folgt dann die Untersuchung von größeren Netzen und variablen Arrival Curves.

Um die Suchverfahren *Monte Carlo* (*OBA(MC)*), *Hooke-and-Jeeves* mit zwei (*OBA(Two)*) und mehreren Dimensionen (*OBA(Multi)*) mit den schon berechneten Ergebnissen vergleichen zu können wurde der Workload erneut ausgeführt, diesmal aber mit den Suchverfahren.

Die Ergebnisse in den Abbildungen 5.12 und 5.13 bestätigen, was am Ende der Versuchsreihe 2 vermutet wurde. Die periodische Struktur des Suchraums in Abbildung 5.11 macht den Algorithmen zu schaffen.

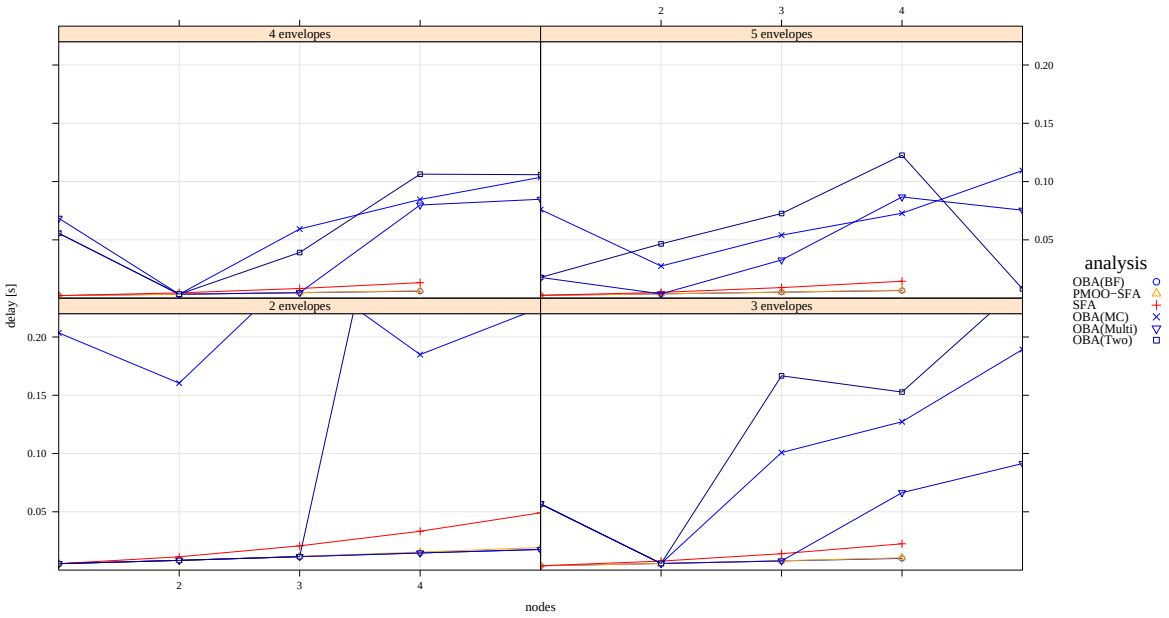


Abbildung 5.12: Delay grouped by envelopes.

Die Betrachtung in Abbildung 5.14 zeigt, einen stetig ansteigene Laufzeit. Diese hält sich für die Optimierungen noch in Grenzen, bis auf die OBA(Two). Bei fünf Knoten ist eine sehr lange Laufzeit zu beobachten. Dies ist insofern sehr ungewöhnlich, da sich die Anzahl der überprüften Kombinationen nicht von denen der OBA(Multi) unterscheiden. Ein ähnliche Ausführungszeit hätte also erwartet werden können. In Tabelle 5.2 sind die Werte, die aus der Reihe tanzen, zusammengefasst.

Für den zweiten Teil dieser Versuchsreihe wurde eine Reihe von Netzwerken von bis zu 15 Knoten betrachtet. Auch hier wurde eine feste Auslastung der Server von 50% gewählt. Die

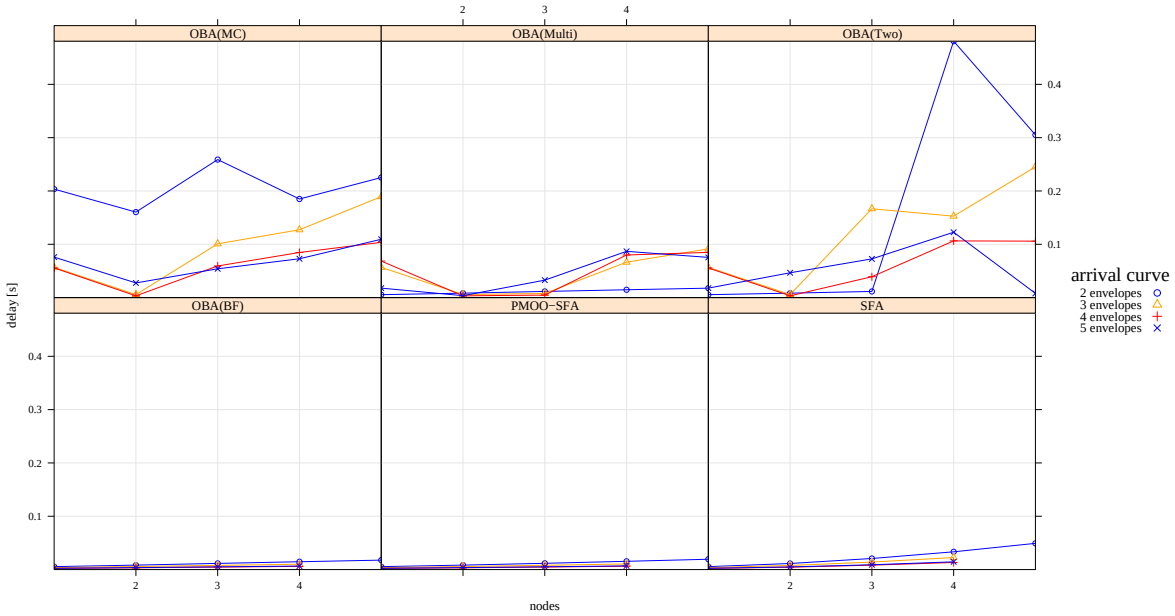


Abbildung 5.13: Delay grouped by analysis method.

Knoten	Envelopes	OBA(Two)	OBA(Multi)
5	4	$2741ms$	$163ms$
5	5	$106704ms = 1.78min$	$196ms$
5	2	$182694ms = 3.05min$	$223ms$

Tabelle 5.2: Ausgewählte Laufzeitwerte.

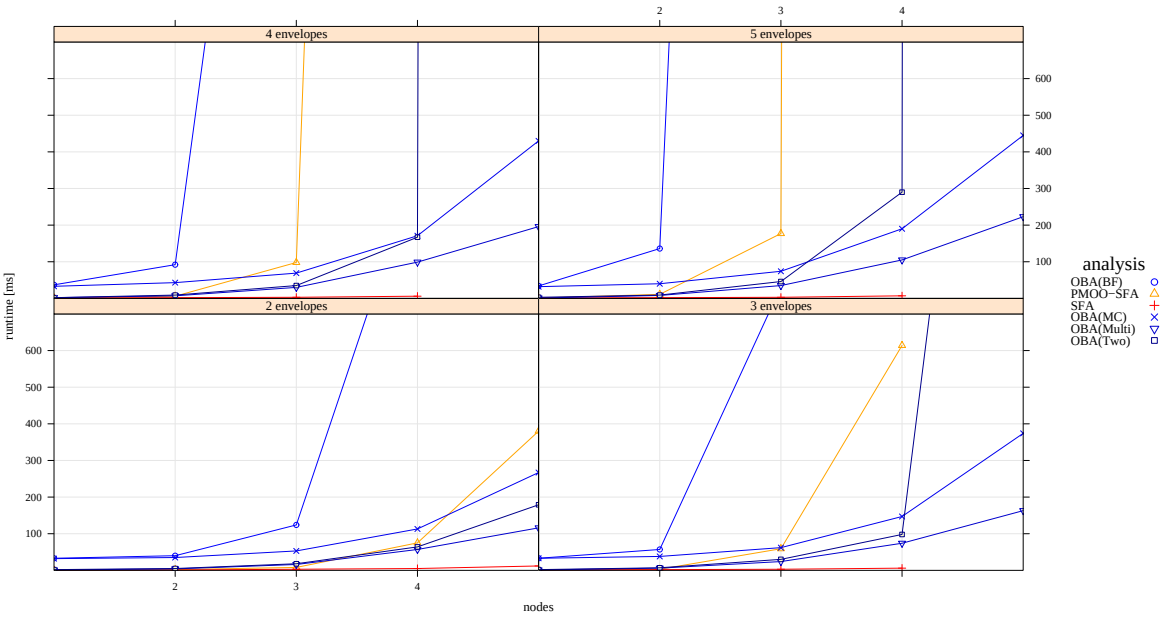


Abbildung 5.14: Runtime grouped by envelopes.

Arrival Curves wurden mit dem Verfahren aus Versuchsreihe 2 bestimmt. Allerdings wurde nicht ein Envelope für jeden Datenstrom des Querverkehrs gewählt, sondern jeder einzelne Datenstrom hat einen eigenen Envelope erhalten. So sollte simuliert werden, dass in einem Netzwerk nur eine begrenzte Anzahl an unterschiedlichen Kategorien von Daten auftreten. Ein Datenstrom aus dem allgemeinen Feed-Forward Netzwerkmodell in Abbildung 3.4 besteht aus unterschiedlich kategorisierten Daten. Für die Herleitung der Envelopes wurde eine Baumtiefe von $d = 5$ gewählt, sowie jeweils 10 unterschiedliche Peak-Rates.

Um einen Vergleich zu einer schon etablierten Methode zu haben, wurde die SFA gewählt. Aus den vergangenen Versuchen hatte die SFA für mehrere Envelope Paare das bessere Laufzeitverhalten als die PMOO-SFA. Außerdem lieferte die SFA bislang den schlechtesten Bound, weswegen dieser Vergleich nicht so drastisch ausfallen sollte.

Eine Betrachtung der Abbildungen 5.15 und 5.16 zeigt die befürchteten Ergebnisse. Einzig und alleine die OBA(Two) scheint vom Kurvenverlauf her in einem Bereich zu liegen, in dem die PMOO-SFA erwartet werden könnte. Eine Anpassung der Heuristik oder andere Optimierungsverfahren müssten in Betracht gezogen werden, um eine Verbesserung zu erlangen.

Die genaue Betrachtung der Laufzeit führt eigentlich zu der Frage, ob die Rechenzeit noch im Verhältnis zum Delay Bound steht. Die maximale Rechenzeit für die OBA(Two) beträgt $22800008ms = 6.3h$, die SFA dagegen braucht nur 4.7 Sekunden.

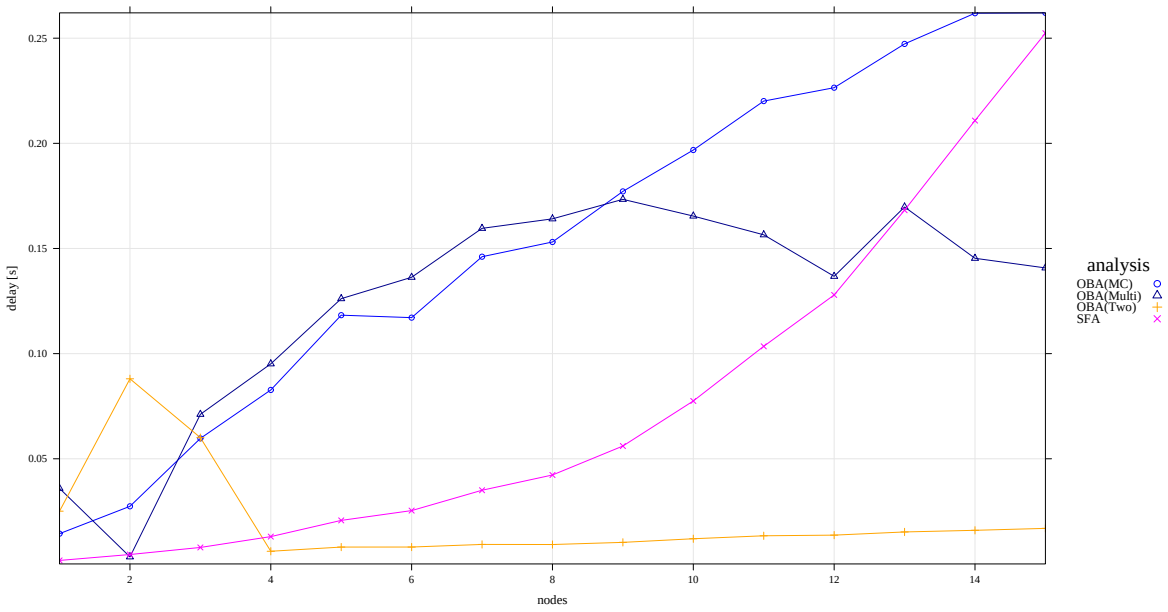


Abbildung 5.15: Delay grouped by envelopes.

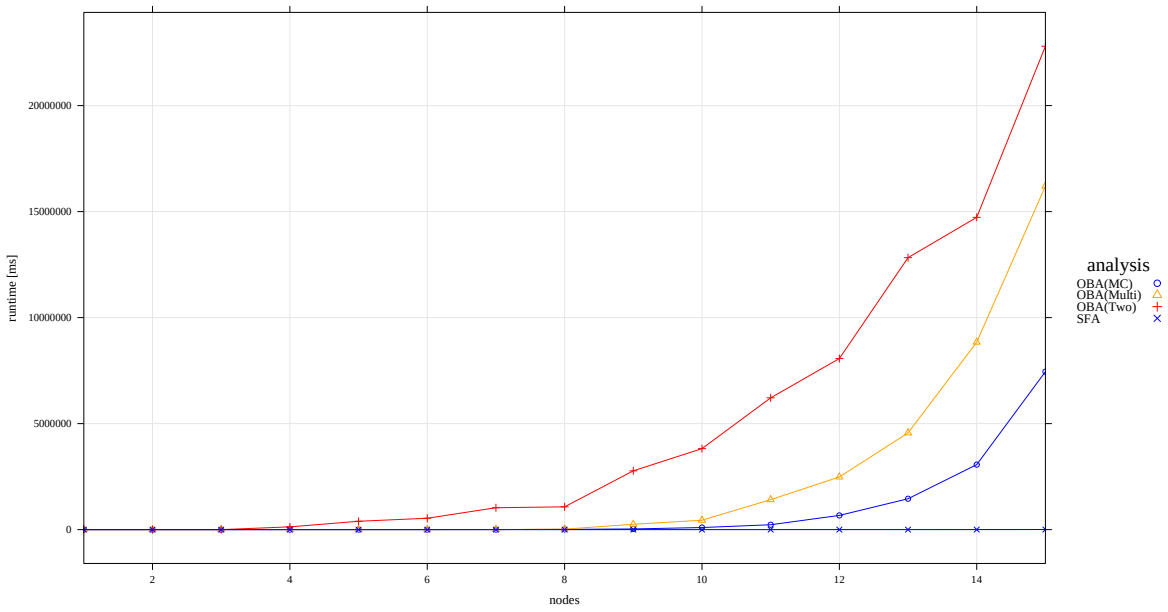


Abbildung 5.16: Runtime grouped by envelopes.

6 Zusammenfassung

In dieser Arbeit wurde ein grundlegendes Problem des Network Calculus bei Feed-Forward Netzwerken abgetastet. Die Versuche haben gezeigt, dass die Berechnung von Tight Bounds bis zu einer bestimmten Netzwerkgröße in einem guten Verhältnis zwischen Laufzeit und Delay Bound stehen.

6.1 Schlussfolgerungen

Ein wichtige Erkenntnis ist, dass die momentane Implementierung dieser Methode sehr viel Heap benötigt. Zwar kann man die entsprechenden Parameter der Java VM verändern und den maximalen Heap erhöhen, dadurch wird das Auftreten des Speicherproblems aber nur verzögert. Deswegen müsste die Implementierung noch genauer analysiert und optimiert werden, so dass man die Anzahl der erstellten Objekte reduzieren kann.

Ein andere Faktor wäre auch die, Betrachtung der Java VM. Einerseits profitiert ein 64-Bit Java zwar dadurch, dass auf einen größeren Speicherbereich zugegriffen werden kann, die JVM erleidet aber einen geringen Leistungsverlust gegenüber der 32-Bit Version, weil ein Zeiger nun mit 8 Bytes doppelt so lang ist. Gerade die erhöhte Datenmenge führt zu, wenn auch sehr geringfügigen, längeren Ladezeiten, aber da für die Implementierung der Optimized Bounds massivst viele Pointer verwendet werden, kann man durchaus davon ausgehen, dass die die Ergebnisse diese Tatsache wiederspiegeln. In Sun JDK 6 Update 10 und JDK 7 kann zwar eine Kompression für Objekte eingeschaltet werden, die Berechnungen liefen aber auf einem System mit Sun JDK 6 Update 7 und eine Veränderung der Systemkonfiguration war keine Option. Diese Idee mit einer geschickten Systemkonfiguration das letzte bisschen Leistung heraus zu kitzeln, kann wohl als letztes Mittel vorgenommen werden, ändert aber an der eigentlichen Problematik nichts.

Die Interpretation der Graphiken zeigt, dass mit einer sehr geringen Variierung eines Faktors gleich eine massive Erhöhung der Komplexität einher geht. Die beschriebenen Möglichkeiten, eine Berechnung sämtlicher Kombinationen durch Suchverfahren zu umgehen, scheint eine gute Möglichkeit zu sein, einen besseren Delay Bound zu berechnen. Jedoch gibt es hier einen Tradeoff zwischen einer besseren Lösung oder einer schnelleren Berechnung. Für kleine, einfache Szenarien haben sich durchaus bessere Ergebnisse gezeigt, aber eine in allen Punkten bessere Analysemethode zur Berechnung von Output Bounds ist es wohl nicht. Vielmehr würde sich die Berechnung eines Szenarios nach dessen Parametern richten. Also Auslastung, Durchmesser des Netzwerks und Art der Arrival und Service Curves. Abhängig davon sollte

eine Analyse ausgewählt werden. In den Versuchen wurden auch Szenarien vorgestellt, deren Delay Verbesserung nicht im Verhältnis zur Laufzeit steht.

Die Berechnung der Optimised Bounds in Feed-Forward Netzen stößt hier an die Grenzen des Network Calculus, dem nur durch massivem Rechenaufwand entgegengewirkt werden kann. Allgemeinere Bedingungen würden das kombinatorischen Problem wohl noch zusätzlich verschlimmern. Es kristallisieren sich eher Spezialanwendungen heraus, die von vornherein einen speziellen Aufbau besitzen, wie die *Sink-Trees* bei diversen *Wireless Sensor Networks*.

Ein weiterer Nachteil ist ebenfalls, dass keine kompletten Netzwerktopologien betrachtet wurden. Zu vermuten ist, dass sich bei realen Beispielen andere Ergebnisse bilden. Die rekursive Berechnung des Querverkehrs führt dann auch zu zusätzlichem Aufwand. Möglicherweise müsste dann die Einschätzung ein wenig revidiert werden.

6.2 Ausblick

Der hier vorgestellte Ansatz zur Optimierung der Delay Bounds ist durchaus eine spannende Anwendung. Allerdings bleiben noch einige Fragestellungen offen.

Es wäre interessant zu wissen, wie sich die Aussagen in dieser Arbeit auch auf allgemeineren Netzwerken treffen lassen.

Es wäre auch gut vorstellbar, das kombinatorische Problem schon am Anfang in der Modellierungsebene zum Teil zu lösen. Ein Szenario könnte auf Topologieebene schon vereinfacht werden, indem bestimmte Knoten von vornherein zu einem Subsystem zusammengefasst werden.

Eine weitere Anregung wäre, den Sourcecode noch zu optimieren. Im Hinblick auf verbesserte Laufzeit und Speicherbedarf gibt es auch noch Potential, das sich wohl nur in einer Verbesserung in Faktoren ausdrücken wird. Diese Verbesserung ist aber durchaus aus Benutzersicht sehr sinnvoll, besonders wenn sich die Laufzeit reduzieren ließe.

7 Anhang

Literaturverzeichnis

[BCG+08] Anne Bouillard, Bertrand Cottenceau, Bruno Gaujal, Laurent Hardouin, Alain Jean-Marie, Laurent Jouhet, Sebastien Lagrange, Mehdi Lhommeau, Nicolas Navet, and Eric Thierry. COmputational Issues in Network Calculus (Coinc). `http://perso.bretagne.ens-cachan.fr/~bouillar/coinc/`, 2008.

[BET07] Anne Bouillard and Éric Thierry. An Algorithmic Toolbox for Network Calculus. Technical Report 6094, Institut National de Recherche en Informatique et en Automatique (INRIA), 2007.

[BET08] Anne Bouillard and Éric Thierry. An Algorithmic Toolbox for Network Calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008.

[BT01] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus. A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001. a.

[CB00] Anna Charny and Jean-Yves Le Boudec. Delay Bounds in a Network with Aggregate Scheduling. In *QOFIS*, 2000.

[CKT03] Samarjit Chakraborty, Simon Kunzli, and Lothar Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 190–195, Washington, DC, USA, 2003. IEEE Computer Society.

[Cru91a] Rene L. Cruz. A Calculus for Network Delay, Part I: Network Elemets in Isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, Jan 1991.

[Cru91b] Rene L. Cruz. A Calculus for Network Delay, Part II: Network Analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, Jan 1991.

[DIS08] DISCO Network Calculator v.1.0.1. `http://disco.informatik.uni-kl.de/content/Downloads`, 2008.

[GDNW04] Xiaoyuan Gu, Matthias Dick, U. Noyer, and Lars Wolf. NMP - A New Networked Music Performance System. In *Proceedings of the 1st IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'04)*, 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, mar 1995.

[HJ61] Robert Hooke and T. A. Jeeves. „Direct Search“ Solution of Numerical and Statistical Problems. *J. ACM*, 8(2):212–229, 1961.

[HK06] Horst W. Hamacher and Kathrin Klamroth. *Lineare Optimierung und Netzwerkoptimierung*. Vieweg+Teubner, 2006.

[INR08] Scilab.
`http://www.scilab.org`, 2008.

[KAT06] Anis Koubaa, Mario Alves, and Eduardo Tovar. Modeling and Worst-Case Dimensioning of Cluster-Tree Wireless Sensor Networks. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 412–421, Washington, DC, USA, 2006. IEEE Computer Society.

[lps09] lp_solve.
`http://lpsolve.sourceforge.net/5.5/`, 2009.

[Mat08] MATLAB®.
`http://www.mathworks.com`, 2008.

[MP06] Shiwen Mao and Shivendra S. Panwar. A survey of envelope processes and their applications in quality of service provisioning. *Communications Surveys And Tutorials, IEEE*, 8(3):2–19, 3rd Qtr. 2006.

[MZCW04] Alexander Maxiaguine, Yongxin Zhu, Samarjit Chakraborty, and Weng-Fai Wong. Tuning SoC platforms for multimedia processing: identifying limits and tradeoffs. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 128–133, New York, NY, USA, 2004. ACM Press.

[Per06] Simon Perathoner. Evaluation and comparison of performance analysis methods for distributed embedded systems. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland and Politecnico di Milano, Italy, Apr 2006.

[Sch08] Henrik Schiøler. Cyclic Network Calculus (CyNC).
`http://www.control.aau.dk/~henrik/CyNC/manual/CyNC0.1Manual.html`, 2008.

[Sim08] Simulink®.
`http://www.mathworks.com`, 2008.

[SKZ03] David Starobinski, Mark G. Karpovsky, and Lev A. Zakrevski. Application of Network Calculus to General Topologies using Turn-Prohibition. *IEEE/ACM Trans. Netw.*, 11(3):411–421, 2003.

[SNLJ05] Henrik Schiøler, Jens F. Dalsgaard Nielsen, Kim Guldstrand Larsen, and Jan Jakob Jessen. CyNC - a method for Real Time Analysis of Systems with Cyclic Data Flows. *13 th. RTS Conference on Embedded Systems*, Apr 2005.

[SSH07] Henrik Schiøler, Hans P. Schwefel, and Martin B. Hansen. CyNC: a MATLAB/SimuLink toolbox for network calculus. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[SZ06] Jens B. Schmitt and Frank A. Zdarsky. The DISCO Network Calculator: A Toolbox for Worst Case Analysis. In *valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, page 8, New York, NY, USA, 2006. ACM Press.

[SZF07] Jens B. Schmitt, Frank A. Zdarsky, and Markus Fidler. Delay Bounds under Arbitrary Multiplexing. Technical Report 360/07, University of Kaiserslautern, Germany, Jul 2007.

[SZM06] Jens B. Schmitt, Frank A. Zdarsky, and Ivan Martinovic. Performance Bounds in Feed-Forward Networks under Blind Multiplexing. Technical Report 349/06, University of Kaiserslautern, Germany, Apr 2006.

[TCN00] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[Wan06] Ernesto Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, ETH Zurich, Sep 2006.

[WT06] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.