Modell-basierte Leistungsbewertung und Optimierung von Multi-Core-Architekturen zur Paketverarbeitung in Kommunikationsnetzen

Torsten M. Runge[1]

Abstract: Software-basierte Paketverarbeitung mit Multi-Core-Prozessoren kann in vielen Bereichen spezialisierte Netzwerkhardware ersetzen. Um die relevanten Leistungsengpässe in solchen komplexen Systemen zu verstehen und durch entsprechende Modifikationen optimieren zu können, wird in dieser Arbeit ein Modellierungsansatz für die Datenverarbeitung und Ressourcenverwaltung in Rechnersystemen vorgeschlagen, welcher für den Netzwerksimulator ns-3 implementiert wurde. Es werden verschiedene Fallstudien mit unterschiedlichem Verkehrsaufkommen (z.B. Traces), Datenverarbeitungsfunktionalitäten (z.B. IPsec-Verschlüsselung) und Systemkonfigurationen (z.B. Treiber) durchgeführt, welche mit Hilfe von realen Testbed-Messungen kalibriert und validiert werden. Darauf basierend wird ein neues Konzept zur QoS-sensitiven Paketverarbeitung entwickelt, wodurch insbesondere die Latenz von QoS-sensitivem Verkehr in Linux-Routern verbessert wird.

1 Einleitung

Aktuelle auf Standard-Hardware basierende Rechnersysteme (z.B. PC) kommen heutzutage nicht nur als Arbeitsplatzrechner, sondern immer häufiger auch als Netzwerkknoten (u.a. Router, Firewall) zum Einsatz. Grund hierfür sind die neuesten technischen Entwicklungen bei Multi-Core-Prozessorsystemen. Diese werden immer attraktiver für den Einsatz in Kommunikationsnetzen, da die Paketverarbeitung sehr gut parallelisierbar ist. Außerdem ist Standard-Hardware weitaus kostengünstiger im Vergleich zu spezialisierten Netzwerkkomponenten und erlaubt eine flexible Erweiterung der Funktionalitäten durch Software-Anpassungen. Durch den Einsatz von Virtualisierungstechniken können die Kostenvorteile und die Flexibilität zusätzlich verstärkt werden, indem mehrere virtuelle Systeme gleichzeitig auf ein und derselben Hardware betrieben werden. Im Rahmen von *Software-Defined Networking* (SDN) und *Network Functions Virtualization* (NFV) beschäftigt sich die Forschung nicht nur damit Endsysteme (z.B. Server), sondern auch Netzwerkknoten (z.B. Router, Firewalls) zu virtualisieren, um neuartige Dienste, die eine besondere Netzfunktionalität erfordern, schnell und einfach realisieren zu können.

Um Standard-Hardware in Kommunikationsnetzen bei hohem Verkehrsaufkommen einzusetzen und neben dem Senden und Empfangen auch erweiterte Datenverarbeitung (z.B. IPsec-Verschlüsselung, Deep Packet Inspection) vornehmen zu können, müssen Treiber, Netzwerkprotokollstapel und Anwendungen besser an die Hardware-Architektur und deren Parallelisierungsmöglichkeiten angepasst werden, um eine optimale Systemleistung zu erreichen. Moderne Multi-Core-Rechner sind komplexe Systeme, deren Arbeitsabläufe und die dazu benötigten, begrenzten Systemressourcen zu nicht trivial vorhersehbarem

[1] Universität Hamburg, FB Informatik, Telekommunikation und Rechnernetze,
runge@informatik.uni-hamburg.de

Leistungsverhalten führen. Die Identifizierung von Leistungsengpässen und die Untersuchung von Auswirkungen spezieller Verarbeitungsoptimierungen sind daher eine wichtige Voraussetzung zur Leistungssteigerung dieser Systeme und stellen als solches ein bedeutendes Forschungsvorhaben dar. Dabei leistet diese Arbeit die folgenden Beiträge.

- Die Arbeit erklärt das komplexe Zusammenspiel zwischen Hardware und Software, insbesondere für die Paketverarbeitung mit Standard-Hardware.
- Es wird ein neuer, allgemeiner Modellierungsansatz für die Datenverarbeitung und Ressourcenverwaltung in Rechnersystemen vorgeschlagen.
- Der Modellierungsansatz wird exemplarisch für den weitverbreiteten Netzwerksimulator ns-3 als Erweiterungsmodul `resource-management` implementiert, welches als Open Source Software unter der Lizenz GNU GPLv2 für den nichtkommerziellen Gebrauch in Forschung und Entwicklung frei zur Verfügung steht.
- Es werden mit Hilfe des Modellierungsansatzes sowie mit Messungen am Realsystem verschiedene Fallstudien zur Analyse und Evaluierung der Leistungsfähigkeit von Standard-Hardware in Hinsicht auf Leistungsengpässe bei der Paketverarbeitung durchgeführt.
- Es wird ein neues QoS-Konzept für die Paketverarbeitung mit Standard-Hardware vorgeschlagen, wodurch die Wartezeit von QoS-sensitivem Verkehr reduziert wird.

2 Modellierung von Rechnersystemen

2.1 Konzept des Modellierungsansatzes

Ein Modell stellt eine vereinfachte Abbildung eines realen Systems dar. Ziel der Modellierung ist es, die Komplexität des Realsystems zu reduzieren, um es untersuchen und verstehen zu können. Prinzipiell existieren in der Modellierung die Ansätze *Top-Down* und *Bottom-Up*. Beim Top-Down-Ansatz wird vom allgemeinen Übergeordneten schrittweise zum speziellen Untergeordneten modelliert, wohingegen beim Bottom-Up-Ansatz in umgekehrter Richtung vorgegangen wird. In dieser Dissertation wird ein Konzept für einen universell anwendbaren Modellierungsansatz zur Berücksichtigung der rechnerinternen Datenverarbeitung und Ressourcenverwaltung vorgeschlagen [Ru15b]. Dabei wird nach dem Top-Down-Ansatz vorgegangen, wodurch ein Modell beliebig erweitert und detailliert werden kann, um weitere Eigenschaften des Realsystems präziser abzubilden.

In dem Modellierungsansatz wird die Datenverarbeitung eines Rechnersystems aus einer Menge von verschiedenen Datenverarbeitungsschritten, den sogenannten Aufträgen, abgebildet. Dabei generiert ein Ereignis (z.B. Tastatureingabe, Paketempfang) einen *Auftrag* bzw. *Task* zur Datenverarbeitung in einem Rechnersystem. Die Bedienung eines Auftrags wird durch eine *Task Unit (TU)* modelliert, welche eine spezifische Funktionalität F der Datenverarbeitung (z.B. IP-Routing-Tabellen-Lookup) repräsentiert, für dessen Erledigung bestimmte Ressourcen R (z.B. CPU, Hauptspeicher) benötigt werden. Durch die Bedienung eines Auftrags können ein oder mehrere Folgeaufträge für nachfolgende Task

Units generiert werden. Außerdem kann ein Auftrag aus mehreren Teilaufträgen bestehen, welche in Abhängigkeit zueinander stehen können. Wenn einzelne Aufträge bzw. Teilaufträge unabhängig voneinander sind, dann können diese parallel ausgeführt werden.

Die Ressourcen, welche von einer Task Unit zur Auftragsbearbeitung benötigt werden, sind typischerweise Hardwarekomponenten (z.B. CPU, Speicher), können aber auch Softwarekomponenten (z.B. Semaphore, Locks) zugeordnet werden. Die Verwaltung der begrenzten Ressourcen, das sogenannte *Resource Management*, wird als fundamentale Aufgabe durch das Betriebssystem übernommen. Diese Betriebssystemfunktionalität wird unter anderem durch den sogenannten *Resource Manager* modelliert (vgl. Abb. 1). Der Resource Manager greift dazu auf einen sogenannten *Resource Pool* zu, in welchem sich alle Ressourcen eines Ressourcentyps befinden. Der Modellierungsansatz ist in die folgenden drei Schichten bzw. Ebenen, den sogenannten *Planes*, unterteilt.

- Die *Processing Plane* stellt die Verarbeitungsschicht dar, in welcher die Datenverarbeitung modelliert wird. Sie besteht aus der Zusammenschaltung von mehreren Task Units mit spezifischen Funktionalitäten (F) zu gerichteten Graphen.
- Der *Resource Plane* sind die begrenzten Ressourcen des Rechnersystems zugeordnet. Jede Ressource (R) besitzt genau einen Ressourcentyp. Für jeden Ressourcentyp existiert genau ein *Resource Pool*, welcher nur Ressourcen dieses Typs enthält.
- In der *Resource Management Plane* wird die Ressourcenverwaltung modelliert. Dabei wird jeder Resource Pool durch genau einen *Local Resource Manager* verwaltet. Zur Vermeidung von Deadlocks bei der Ressourcenvergabe der verschiedenen Local Resource Manager existiert zusätzlich ein *Global Resource Manager*.

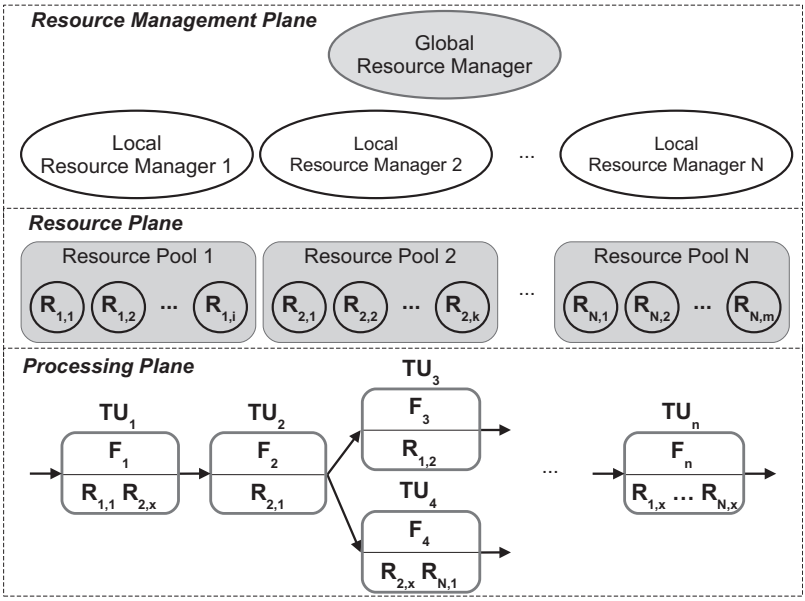

Abb. 1: Überblick zum Modellierungsansatz

2.2 Anwendung des Modellierungsansatzes für SDN und NFV

Der Modellierungsansatz für die Datenverarbeitung und das Ressource Management kann beispielsweise für die aktuellen Forschungsgebiete SDN und NFV angewendet werden (vgl. Abb. 2). In diesem Fall repräsentieren die Task Units die *virtuellen Netzwerkfunktionen* (VNF). Die modellierten physikalischen Ressourcen (z.B. CPU, Cache, Hauptspeicher) stellen die *Network Functions Virtualization Infrastructure* (NFVI) dar. Mit den Verbindungen zwischen den einzelnen Task Units wird ein gerichteter Graph gebildet, welcher in der NFV-Architektur das VNF Service Chaining abbildet. Die NFV-Managementkomponenten Orchestrator, VNF Manager und Virtual Infrastucture Manager werden durch den Global Resource Manager sowie die Local Resource Manager modelliert. Damit ermöglicht der vorgeschlagene Modellierungsansatz die umfangreiche Untersuchung komplexer Szenarien in virtuellen Netzumgebungen, um beispielsweise Fragestellungen wie ein effizientes VNF Placement, Migration, Leistungsfähigkeit, Skalierbarkeit oder Energieverbrauch zu analysieren.

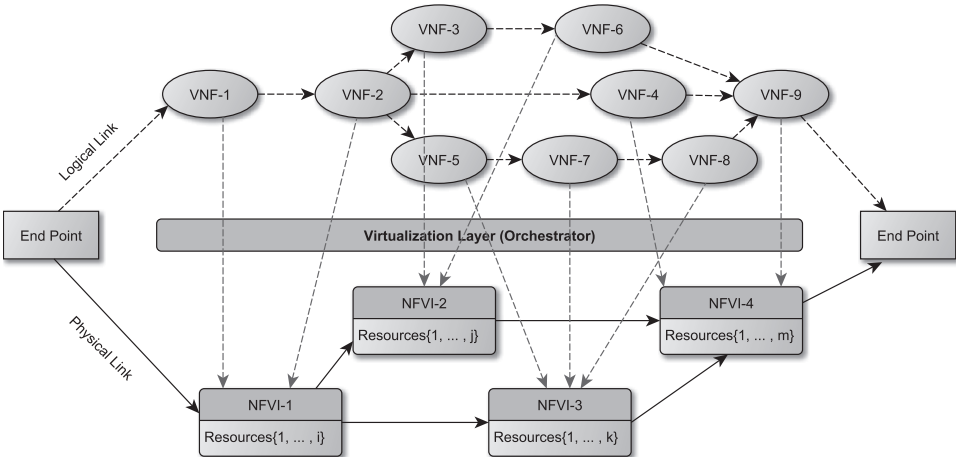

Abb. 2: Abbildung von VNFs zu physikalischen Ressourcen

3 QoS-sensitive Paketverarbeitung mit Standard-Hardware

Mit Hilfe des vorgeschlagenen Modellierungsansatzes wurde ein neues Konzept für die QoS-sensitive Paketverarbeitung mit Standard-Hardware entwickelt [Ru15a]. In der klassischen Linux NAPI mit lediglich einer Empfangswarteschlange (Rx-Ring) pro CPU-Kern und Netzwerkkarteneingangsport wird nicht zwischen Verkehrsklassen unterschieden (Single Queue, SQ). Jedoch ist dies für die differenzierte Behandlung von QoS-sensitivem Verkehr wichtig, um ein vorhersagbares Leistungsverhalten und garantierte Dienstqualität zu gewährleisten. Dazu müssen die Pakete beim Empfang entsprechend ihrer QoS-Anforderungen klassifiziert und in dedizierte Empfangswarteschlangen einsortiert werden. Anschließend erfolgt eine Priorisierung der jeweiligen Rx-Ringe mit QoS-sensitivem Verkehr entsprechend einer Scheduling-Strategie wie beispielsweise Round-Robin (RR) oder Weighted Fair Queueing (WFQ).

3.1 Konzept der QoS-NAPI

Verkehrsklassifizierung Moderne Netzwerkkarten verfügen über verschiedenste Funktionen, um einzelne Paketverarbeitungsaufgaben von der CPU auf die Netzwerkkarte (Network Interface Card, NIC) auszulagern. Zum Beispiel unterstützt der Intel 82599 NIC Controller Multi-Queueing und Receive-Side Scaling (RSS), um die zu bearbeitenden Pakete effizient auf die verfügbaren CPU-Kerne aufzuteilen. Spezielle Hardware-Filter der Netzwerkkarte erlauben die Klassifizierung auf Basis von MAC- oder IP-Header (z.B Port, ToS, VLAN). Im Fall der Paketverarbeitung mit Standard-Hardware können diese Funktionen der Netzwerkkarte genutzt werden, indem der zu bearbeitende Netzverkehr entsprechend der QoS-Anforderungen des jeweiligen Flows durch NIC-Filter klassifiziert und in einen speziellen Rx-Ring einsortiert wird. Dabei sollten für jeden NIC-Port, jeden CPU-Kern und für jede Verkehrsklasse dedizierte Rx-Ringe vorgehalten werden. Wenn ein empfangenes Paket mit keinem NIC-Filter übereinstimmt, dann wird es mittels RSS in die Verkehrsklasse mit der niedrigsten Priorität einsortiert (z.B. Best Effort).

In Abb. 3 sind die Erweiterungen für einen QoS-sensitiven Software-Router für die Verkehrsklassen Real-Time (RT) und Best Effort (BE) hervorgehoben. Somit wird eine Verkehrsklassifizierung ohne CPU-Belastung mit Hilfe der Netzwerkkarte durchgeführt (Classifier) und empfangene Pakete werden entsprechend ihrer Paketattribute in spezielle Rx-Ringe einsortiert. Danach werden die Pakete ebenfalls ohne CPU-Beteiligung durch Direct Memory Access (DMA) über Peripheral Component Interconnect Express (PCIe) in den Hauptspeicher transferiert. Anschließend wird ein Hardware-Interrupt (IRQ) am zuständigen CPU-Kern ausgelöst, sofern der Rx-Ring nicht leer gewesen ist. Danach erfolgt die Bearbeitung der Rx-Ringe entsprechend der Priorität ihrer Verkehrsklasse durch die *QoS-NAPI*, welche im nächsten Abschnitt beschrieben wird.

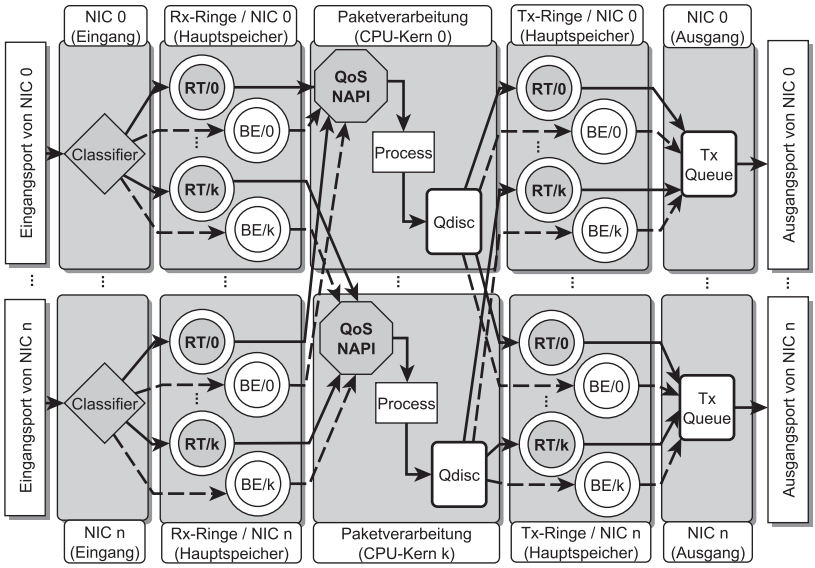

Abb. 3: Software-Router mit Rx/Tx-Ringen für QoS-sensitiven Verkehr

Verkehrspriorisierung Herkömmliche Scheduling-Mechanismen wie Qdisc zur differenzierten Paketbehandlung werden nur am NIC-Ausgangsport unterstützt. Am NIC-Eingangsport werden mehrere Rx-Ringe, die demselben CPU-Kern zugeordnet sind, lediglich im Round-Robin-Verfahren bedient. Eine Verkehrsklassifizierung und Priorisierung ist mit der klassischen Linux NAPI am NIC-Eingangsport nicht möglich. Für eine QoS-sensitive Paketverarbeitung müssen jedoch die Rx-Ringe der NIC-Eingangsports mit QoS-sensitivem Verkehr von der CPU bevorzugt bedient werden. Für eine solche Behandlung von QoS-sensitivem Verkehr mit beliebig vielen Verkehrsklassen muss die Linux NAPI zur *QoS-NAPI* erweitert werden.

In den Abb. 3 und 4 sind die dafür notwendigen NAPI-Erweiterungen hervorgehoben. Anstatt einer Poll-Liste für alle Rx-Ringe wird pro Verkehrsklasse (und pro CPU-Kern) eine dedizierte Poll-Liste verwendet. Demnach befinden sich alle Rx-Ringe mit der gleichen Verkehrsklasse in derselben Poll-Liste, sofern sie vom gleichen CPU-Kern bedient werden. Am Anfang prüft die QoS-NAPI die Poll-Listen, ob Rx-Ringe zur Bearbeitung hinzugefügt wurden, beginnend bei der Poll-Liste für die Verkehrsklasse mit der höchsten Priorität. Wenn sich mindestens ein Rx-Ring in einer Poll-Liste befindet, dann wird der erste Rx-Ring dieser Poll-Liste zur Bearbeitung ausgewählt. Nach der Bearbeitung eines Pakets des Rx-Ringes wird geprüft, ob in einer Poll-Liste einer Verkehrsklasse mit höherer Priorität Rx-Ringe hinzugefügt und somit Pakete empfangen wurden. Wenn dies der Fall ist, dann wird die Bearbeitung der aktuellen Poll-Liste unterbrochen und wechselt zu der nichtleeren Poll-Liste mit der höchstprioren Verkehrsklasse. Schließlich werden die Rx-Ringe entsprechend ihrer Priorität auf Basis einer Scheduling-Strategie vom jeweils zugewiesenen CPU-Kern bevorzugt bedient. Durch die damit verbundene differenzierte Paketbehandlung am NIC-Eingangsport wird die Paketverarbeitungszeit, insbesondere für latenz-sensitive Applikationen, deutlich reduziert.

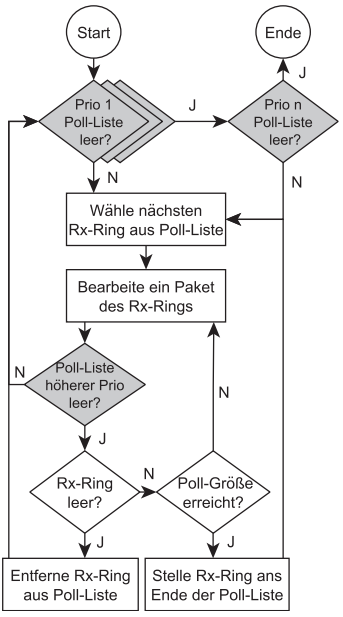

Abb. 4: UML-Aktivitätsdiagramm der Linux QoS-NAPI

3.2 Modellierung der QoS-NAPI

Zur Analyse eines Linux-Software-Routers wird der vorgeschlagene Modellierungsansatz zur Modellierung der Linux NAPI angewendet (vgl. Abschitt 2.1). Das Modell besteht aus $(n+1)$ NIC-Ports ($NIC_0, \ldots, NIC_n$) und einem CPU-Kern C_0 (vgl. Abb. 5). Im Netzwerksimulator ns-3 wird jede Netzwerkkarte durch ein dediziertes *NetDevice* repräsentiert. Jedem NetDevice ist eine *TU NIC* zugeordnet. Somit übergibt jedes NetDevice dessen empfangene Pakete in die Eingangswarteschlange (Rx Queue) einer dedizierten *TU NIC*.

Die *TU NIC* modelliert das Verhalten der NIC, indem die empfangenen Pakete basierend auf den Paketattributen in verschiedene Eingangswarteschlangen der Nachfolger-TUs (*TU NAPI*) einsortiert werden. Dadurch können Pakete mit QoS-Anforderungen klassifiziert und in eine spezielle Eingangswarteschlange einer Nachfolger-TU weitergeleitet werden. Diese Eingangswarteschlange repräsentiert einen Rx-Ring des Realsystems. Nach der Bearbeitung durch eine *TU NAPI* wird das Paket zum Versand in die entsprechende Ausgangswarteschlange (Tx Queue) des NIC-Ausgangsports (NetDevice) einsortiert.

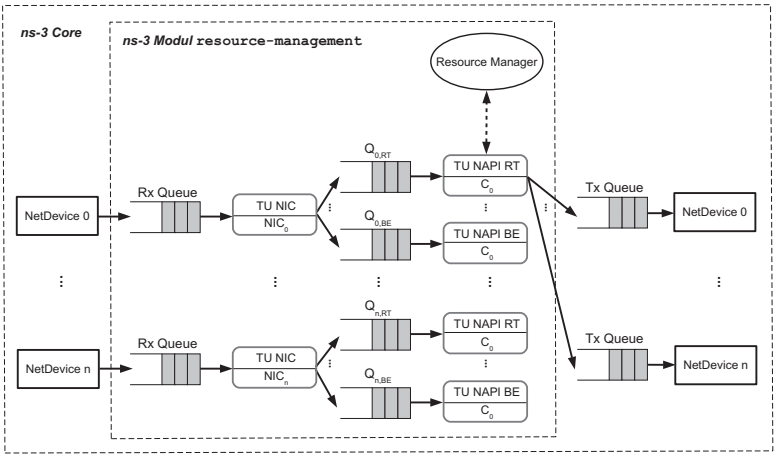

Abb. 5: ns-3-Simulationsmodell eines Linux-Routers mit Rx/Tx-Ringen für QoS-sensitiven Verkehr

Der Interrupt-Moderation-Mechanismus der Linux NAPI wird ebenfalls mit Hilfe des Modellierungsansatzes (vgl. Abschnitt 2.1) abgebildet. Dies beinhaltet die Aktivierung und Deaktivierung von Interrupts (IRQs) sowie die Paketverarbeitung durch Polling. In einem Linux-Software-Router mit Multi-Queueing-Unterstützung existiert für jeden CPU-Kern pro NIC-Port ein dedizierter Rx/Tx-Ring. Im Fall der QoS-NAPI wird zusätzlich für jede Verkehrsklasse ein dedizierter Rx-Ring angelegt, sodass für jede Kombination aus CPU-Kern, NIC-Port und Verkehrsklasse ein dedizierter Rx-Ring vorhanden ist. Ein NAPI-Thread ist an einen bestimmten CPU-Kern gebunden, welcher die zugewiesenen Rx/Tx-Ringe abwechselnd bedient.

Im Software-Router-Modell wird ein NAPI-Thread durch mehrere NAPI Task Units repräsentiert. Dabei korreliert die Anzahl der NAPI Task Units (*TU NAPI*) mit der Anzahl der Rx-Ringe, welche der NAPI-Thread bedient. Jede *TU NAPI* besitzt genau eine Eingangswarteschlange, welche einen bestimmten Rx-Ring abbildet. Zur Modellierung eines Software-Routers mit m Verkehrsklassen, n NIC-Ports und k CPU-Kernen werden demzufolge $m \cdot n \cdot k$ NAPI Task Units benötigt (klassische NAPI: $m = 1$). Im QoS-Software-Router-Modell mit dedizierten Rx-Ringen für RT- und BE-Verkehr wird zwischen den NAPI Task Units *TU NAPI RT* und *TU NAPI BE* unterschieden (vgl. Abb. 5).

Die NAPI Task Units konkurrieren um dieselbe geteilte CPU-Kern-Ressource. Daher können NAPI Task Units, welche dieselbe CPU-Kern-Ressource benötigen, nicht gleichzeitig ausgeführt werden. Diese Tatsache entspricht genau dem Fakt, dass ein NAPI-Thread im Realsystem nur einen Rx-Ring zu einem bestimmten Zeitpunkt bedienen kann.

3.3 Anwendung der QoS-NAPI in der Praxis

Auf Grundlage des im vorherigen Abschnittes beschriebenen Simulationsmodells wurde die Linux NAPI mit den vorgeschlagenen Optimierungen zur QoS-NAPI analysiert.

In Abb. 6 und 7 beschreibt der Echtzeit-Anteil den prozentualen Beitrag von RT-Verkehr am Gesamtverkehr (BE- und RT-Verkehr). Für WFQ wird das Bedienzeitverhältnis RT:BE bezüglich des jeweiligen Rx-Ringes angegeben. Die Resultate zeigen, dass insbesondere mit der Prioritätsstrategie (Prio) die mittlere Paketlatenz des RT-Verkehrs, auch bei hohen Paketankunftsraten von 1,3 Mpps (Millionen Pakete pro Sekunde) und hohem Echtzeit-Anteil von 30 %, von 2,4 μs auf 1,1 μs reduziert wird.

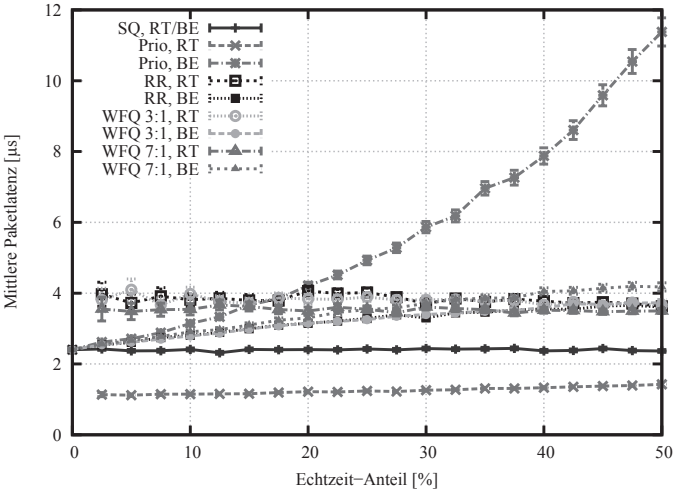

Abb. 6: Mittlere Paketlatenz in Abhängigkeit vom Echtzeit-Anteil für verschiedene Scheduling-Strategien bei 1,3 Mpps Paketankunftsrate

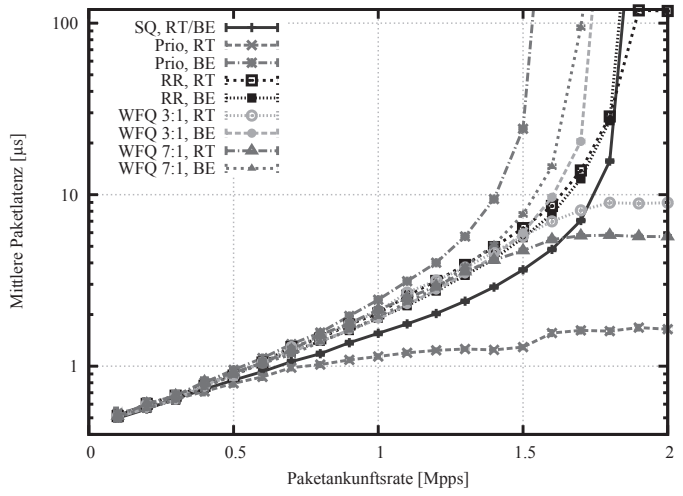

Abb. 7: Mittlere Paketlatenz in Abhängigkeit von der Paketankunftsrate für verschiedene Scheduling-Strategien bei 30 % Echtzeit-Anteil

Auf Basis der Simulationsmodelle wurde schließlich eine prototypische Implementierung der QoS-NAPI entwickelt [Be16]. Mit umfangreichen Testbed-Messungen konnten die Leistungssteigerungen der QoS-NAPI durch einen realen Linux-Software-Router bestätigt werden. Somit wird im Vergleich zum Stand der Technik (SQ, RR) durch die QoS-NAPI (mit Prio) die Paketverarbeitung von latenz-sensitivem Verkehr deutlich verbessert. QoS-NAPI ist außerdem als reine Softwarelösung kostengünstig realisierbar.

4 Zusammenfassung

Das Ziel, mit Software-basierter Paketverarbeitung auf Basis von Standard-Hardware die Leistungsfähigkeit von spezialisierter Netzwerkhardware mit Hunderten von 10 Gbps Netzwerkports zu erreichen, ist anspruchsvoll. In diesem Zusammenhang beschäftigt sich diese Dissertation [Ru16] mit der Leistungsbewertung und Optimierung von Multi-Core-Architekturen zur Paketverarbeitung in Kommunikationsnetzen. Dazu geht die Arbeit auf bestehende Betriebssysteme, insbesondere Linux, und Softwareoptimierungen für die Paketverarbeitung ein. Außerdem werden alternative Networking Frameworks wie Click [Ko00], DPDK [Int15], Netmap [Ri12] und RouteBricks [Fa11, DAR12] diskutiert.

Schließlich wird ein neuer, universeller Modellierungsansatz für die Leistungsanalyse und Optimierung der Datenverarbeitung in ressourcenbeschränkten Rechnersystemen vorgeschlagen [Ru15b], welcher als Open Source Software für den Netzwerksimulator ns-3 verfügbar ist.

Es werden verschiedene Fallstudien zur Leistungsbewertung und Optimierung der Paketverarbeitung durchgeführt. Zur QoS-sensitiven Paketverarbeitung wird ein neues Konzept für die Erweiterung der Linux NAPI zur QoS-NAPI vorgeschlagen [Ru15a, Be15]. Dazu wird QoS-sensitiver Verkehr bereits am NIC-Eingangsport klassifiziert und entsprechend einer Scheduling-Strategie bevorzugt durch die CPU bedient, da diese üblicherweise den Engpass in der Paketverarbeitung mit Standard-Hardware darstellt. Im Gegensatz dazu wird bei den herkömmlichen QoS-Techniken wie DiffServ oder IntServ eine QoS-differenzierte Paketbehandlung nur für den NIC-Ausgangsport berücksichtigt. Damit erfolgt eine QoS-differenzierte Paketbehandlung bereits vor der eigentlichen Paketverarbeitung durch die CPU.

Zur Bewertung des QoS-Konzepts wird mit Hilfe des vorgeschlagenen Modellierungsansatzes ein Simulationsmodell eines Linux-Software-Routers entworfen. Die Resultate zeigen, dass mit der Prioritätsstrategie die Paketlatenz für QoS-sensitive Netzapplikationen deutlich reduziert wird und QoS-Garantien gewährleistet werden.

Schließlich erfolgte auf Basis des vorgeschlagenen QoS-NAPI-Konzeptes eine prototypische Implementierung für einen Linux-Software-Router als Proof-of-Concept [Be16]. Dazu wurden ausführliche Testbed-Messungen durchgeführt, welche die Leistungssteigerungen und Robustheit der Implementierung für unterschiedlichste Szenarien belegen. Damit wird durch die QoS-NAPI eine Optimierung der Paketverarbeitung innerhalb des Linux-Betriebssystems ermöglicht, welche als Softwarelösung kostengünstig implementierbar ist. Die Implementierung der QoS-NAPI steht als Open Source Software der Linux-Community frei zur Verfügung.

Die aufgeführten Aspekte dieser Arbeit zeigen deutlich, dass das Potential von Software-basierter Paketverarbeitung mit Standard-Hardware sehr vielversprechend ist, welches unter anderem in den aktuellen Forschungsgebieten SDN und NFV aufgegriffen wird.

Literaturverzeichnis

[Be15] Beifuß, A.; Raumer, D.; Emmerich, P.; Runge, T. M.; Wohlfart, F.; Wolfinger, B. E.; Carle, G.: A Study of Networking Software Induced Latency. In: International Conference on Networked Systems (NetSys). S. 1–8, März 2015.

[Be16] Beifuß, A.; Runge, T. M.; Raumer, D.; Emmerich, P.; Wolfinger, B. E.; Carle, G.: Building a Low Latency Linux Software Router. In: International Teletraffic Congress (ITC). S. 1–9, September 2016.

[DAR12] Dobrescu, M.; Argyraki, K.; Ratnasamy, S.: Toward Predictable Performance in Software Packet-Processing Platforms. In: USENIX Conference on Networked Systems Design and Implementation (NSDI). S. 141–154, April 2012.

[Fa11] Fall, K.; Iannaccone, G.; Manesh, M.; Ratnasamy, S.; Argyraki, K.; Dobrescu, M.; Egi, N.: RouteBricks: Enabling General Purpose Network Infrastructure. ACM SIGOPS Operating Systems Review, 45(1):112–125, 2011.

[Int15] Intel Corporation. Data Plane Development Kit: Programmer's Guide, April 2015.

[Ko00] Kohler, E.; Morris, R.; Chen, B.; Jannotti, J.; Kaashoek, M. F.: The Click Modular Router. ACM Transactions on Computer Systems (TOCS), 18(3):263–297, August 2000.

[Ri12] Rizzo, L.: Revisiting Network I/O APIs: The Netmap Framework. Communications of the ACM, 55(3):45–51, 2012.

[Ru15a] Runge, T. M.; Raumer, D.; Wohlfart, F.; Wolfinger, B. E.; Carle, G.: Towards Low Latency Software Routers. Journal of Networks, 10(4):188–200, 2015.

[Ru15b] Runge, T. M.; Wolfinger, B. E.; Heckmüller, S.; Abdollahpouri, A.: A Modeling Approach for Resource Management in Resource-Constrained Nodes. Journal of Networks, 10(01):39–50, 2015.

[Ru16] Runge, T. M.: Modell-basierte Leistungsbewertung und Optimierung von Multi-Core-Architekturen zur Paketverarbeitung in Kommunikationsnetzen. Berichte aus dem Forschungsschwerpunkt Telekommunikation und Rechnernetze. Shaker-Verlag, 2016.

Torsten M. Runge absolvierte das Studium der Informationstechnik/Technischen Informatik an der Universität Rostock (2001-2006). Anschließend arbeitete er als Forschungs- und Entwicklungsingenieur für die Siemens AG und die Deutsche Telekom AG (2007-2012), wo er in mehreren internationalen Forschungsprojekten involviert war und viele Patentanmeldungen sowie Publikationen hervorgingen. Danach promovierte er mit Auszeichnung an der Universität Hamburg im Bereich der Software-basierten Paketverarbeitung mit Multi-Core-Prozessoren (2012-2016). Seine Veröffentlichungen wurden bereits mehrfach mit Best Paper Awards ausgezeichnet. Seit 2017 forscht Dr. Runge am International Computer Science Institute (ICSI) der University of California, Berkeley (USA) in den Bereichen Software-Defined Networking und Network Functions Virtualization.