Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Pascal Erni

Einsatz und Programmierung des IBM NP4GS3 Netzwerkprozessors für Aktive Netzwerkknoten unter Linux

Diploma Thesis DA-2003.13
October 2002 to February 2003

Supervisor: Lukas Ruf
Co-Supervisor: Matthias Bossardt
Professor: Bernhard Plattner

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Zusammenfassung

Im Rahmen dieser Diplomarbeit soll die Einsatzmöglichkeit Netzwerkprozessor-basierter Interface Cards zur Unterstützung eines PromethOS-basierten aktiven Netzwerkknotens (ANN) untersucht werden. Durch die Netzwerkprozessoren soll eine Leistungssteigerung für ANNs erreicht werden. Die Arbeit beschreibt eine Architektur, wie Netzwerkprozessoren das NodeOS PromethOS unterstützen können. Diese Architekur wurde für den IBM PowerNP NP4GS3 Netzwerkprozessor implementiert und mit detaillierten Performancemessungen evaluiert. Um die Möglichkeiten von Netzwerkprozessoren noch besser für PromethOS nützen zu können, wird eine Erweiterung der bestehenden PromethOS Architektur durch «Pico Plugins» vorgeschlagen. Mit Pico Plugins erhalten die PromethOS Plugins die Möglichkeit, Teile ihrer Funktionalität in den Netzwerkprozessor auszulagern, um einen weiteren Performancegewinn zu erhalten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Einen grossen Teil zum Erfolg des heutigen Internets hat die einfache und elegante Architektur des Netzes beigetragen. Router in der Mitte des Netzwerkes, die nichts anderes machen als möglichst schnell Pakete weiterzuleiten, während auf den Computern, an den Ecken des Netzwerkes, die komplexeren Aufgaben der end-to-end Verbindungen sowie die Applikationen ablaufen. Dennoch geht der Trend immer mehr Richtung Erweiterung der Routerfunktionalität. Neben der bisherigen Paket-Weiterleitung kommen Aufgaben wie z. B. Pakete filtern, Quality of Service Reservation, Proxies sowie die Unterstützung diversester Applikationen hinzu.
Aktive Netzwerkknoten (Active Network Node (ANN)), die dem Plugin Modell folgen, sind ein Vertreter dieser neuen Routertechnologie. Sie bieten die Möglichkeit, zur Laufzeit ihre Funktionalität durch Installation von Service Code zu erweitern. Es soll Code von Service Providern auf einem ANN installiert werden können, ohne dass die Funktion des ANNs beeinträchtigt wird. Auf dem ANN führt ein Node Operating System (NodeOS) die Funktion der Code-Verwaltung durch. Am Institut für Technische Informatik und Kommunikationsnetze (TIK) der ETH Zürich wurde ein solches NodeOS, PromethOS, durch Erweiterung des Linux Kernels 2.2.17 entwickelt. Es ermöglich zur Laufzeit das Laden von PromethOS Plugins, die als Linux Kernel Module ausgeführt werden.
Auch die Chiphersteller haben diesen Trend erkannt und ihre Entwicklung entsprechend ausgerichtet. Seit relativ kurzer Zeit werden sogenannte Netzwerkprozessoren hergestellt. Sie bestehen in der Regel aus einem Core, welcher für das Management des Netzwerk Prozessors zuständig ist, und sogenannten Micro oder Pico Engines, welche für typische Netzwerkaufgaben programmiert werden. Beginnend mit einfachen Byte-Vergleichs- oder Tabellen-Abfrageoperationen bis hin zu komplexen forwarding Operationen. Typische Vertreter dieser Netzwerkprozessoren werden von IBM (NP4GS3) und von Intel (IPX1200/2400/2800) hergestellt.

1.2 Aufgabenstellung

Am TIK wird ein Ansatz verfolgt, um diese Netzwerkprozessoren als dynamisch programmierbare Ausführungseinheiten in aktiven Knoten zu verwenden. Bei dieser Diplomarbeit soll die Einsatzmöglichkeit Netzwerkprozessor-basierter Interface Cards zur Unterstützung eines PromethOS-basierten ANN untersucht werden. Als Entwicklungsplattform steht ein IBM Application Reference Board NP4GS3 von Silicon & Software Systems zu Verfügung.

1.2.1 Ziel

Im Rahmen dieser Diplomarbeit soll die Einsatzmöglichkeit Netzwerkprozessor-basierter Interface Cards zur Unterstützung eines PromethOS-basierten ANN untersucht werden. Durch den Einsatz von Netzwerkprozessoren soll eine Leistungssteigerung für der PromethOS-basierten ANN erreicht werden. Um diese Ziel zu erreichen, wurden Zwischenziele definiert:

- Inbetriebnahme des Application Reference Boards (inkl. embedded PowerPC Subsystem)
- Entwickeln eines Modells für den Einsatz von Netzwerkprozessoren in aktiven Netzwerkknoten.
- Entwickeln einer Node-Architektur, die PromethOS ermöglicht, Netzwerkprozessor Interface Cards zu verwenden
- Implementation und Evaluation dieser Node-Architektur
- Aufzeigen von möglichen weiterführenden Arbeiten anhand der Evaluation der daraus abgeleiteten Schlussfolgerungen

Die Resultate und der Weg dazu sollen in dieser Dokumentation klar und ausführlich dokumentiert werden.

1.2.2 Gliederung des Berichtes

Der Bericht gliedert sich in die fünf Teile:

- Einführung: Neben der Motivation für diese Arbeit wird auf die Aufgabenstellung, Gegebenheiten und Voraussetzungen der Arbeit eingegangen.
- Architektur: Durch das Aufzeigen von Möglichkeiten mit Vor- und Nachteil soll auf eine Architektur hingearbeitet werden. Die gewählte Architektur wird erläutert und begründet. Zudem wird auf die Erweiterungsmöglichkeiten dieser Architektur hingewiesen.
- Implementation: Wie wurde die Architektur umgesetzt. Dabei soll dokumentiert werden, welche Änderungen und Erweiterungen in der bestehenden Software eingefügt wurden. Die Funktionalität neu inzukommender Komponenten wird ausführlich beschrieben.
- Evaluierung: Anhand von Messungen soll die Leistungsfähigkeit dieser Implementation ausgewertet und beurteilt werden.
- Zusammenfassung und Ausblick: In diesem Kapitel wird das Erreichte nochmals zusammengefasst und ein mögliches weiteres Vorgehen aufgezeigt.

Im Anhang finden sich die Originalaufgabenstellung, Zeitplan sowie die verwendeten Konfiguration und Skripte.

1.3 Dankesworte

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei meiner Diplomarbeit am Institut für Technische Informatik und Kommunikationsnetze der ETH Zürich und bei der Erstellung dieses Berichtes unterstützt haben.
Ich möchte mich auch für die grosse Hilfe von Patrick Droz und seinem Forschungsteam vom IBM Zurich Research Laboratory in Rüschlikon bedanken. Durch das zur Verfügung stellen eines IBM PowerNP NP4GS3 Netzwerkprozessors wurde diese Diplomarbeit erst möglich. Bei Roman Pletka möchte ich mich für seine grosse Unterstützung bedanken und dafür, dass ich ihn fast zu jeder Tag- und Nachtzeit kontaktieren konnte, wenn ich irgendwo mit dem Netzwerkprozessor anstand. Ihm ist es auch zu verdanken, dass es am Schluss meiner Arbeit möglich war, das embedded PowerPC Subsystem mit in die Tests und Evaluierung miteinzubeziehen.
Prof. Dr. Bernhard Plattner und Lukas Ruf möchte ich für das in mich gesetzte Vertrauen danken. Die vier Monate der Diplomarbeit erlebte ich als sehr lehrreich und interessant. Trotz der vielen Arbeit habe ich nie die Freude daran verloren, was sicher auch auf die gute Betreuung am Institut für Technische Informatik und Kommunikationsnetze der ETH Zürich zurückzuführen ist.
Zuletzt möchte ich mich bei all den Personen bedanken, die während meiner Diplomarbeit oft auf mich verzichten und sehr viel Geduld aufbringen mussten, bis dieses Werk vollendet war.

Vielen Dank!

Hildisrieden, Februar 2003

Pascal Erni

2 Entwicklungsumgebung

Das Application Reference Board für den IBM PowerNP NP4GS3 Netzwerkprozessor wurde durch die Firma Silicon & Software Systems (S3) hergestellt. Die Hauptkomponenten sind ein NP4GS3B PowerNP, drei 1000BaseT Ethernet Netzwerkanschlüsse und eine PCI-GMII Bridge. Für die Kommunikation zwischen dem Host und dem NP4GS3 dient der BroadCom BMC5700 Gigabit-Ethernet Netzwerkcontroller. Dadurch erscheint die Karte dem Host als normale BroadCom Netzwerkkarte und wird auch so angesteuert.

2.1 IBM PowerNP NP4GS3 Netzwerkprozessor

Der in Abbildung 1 schematisch dargestellte IBM PowerNP NP4GS3 Netzwerk Prozessor besteht aus einem Embedded Processor Complex (EPC), Physical MAC Multiplexer (PMM), zwei Enqueuer/Dequeuer/Scheduler (EDS), zwei Switch Interface (SWI) und mehreren Speicherkomponenten.

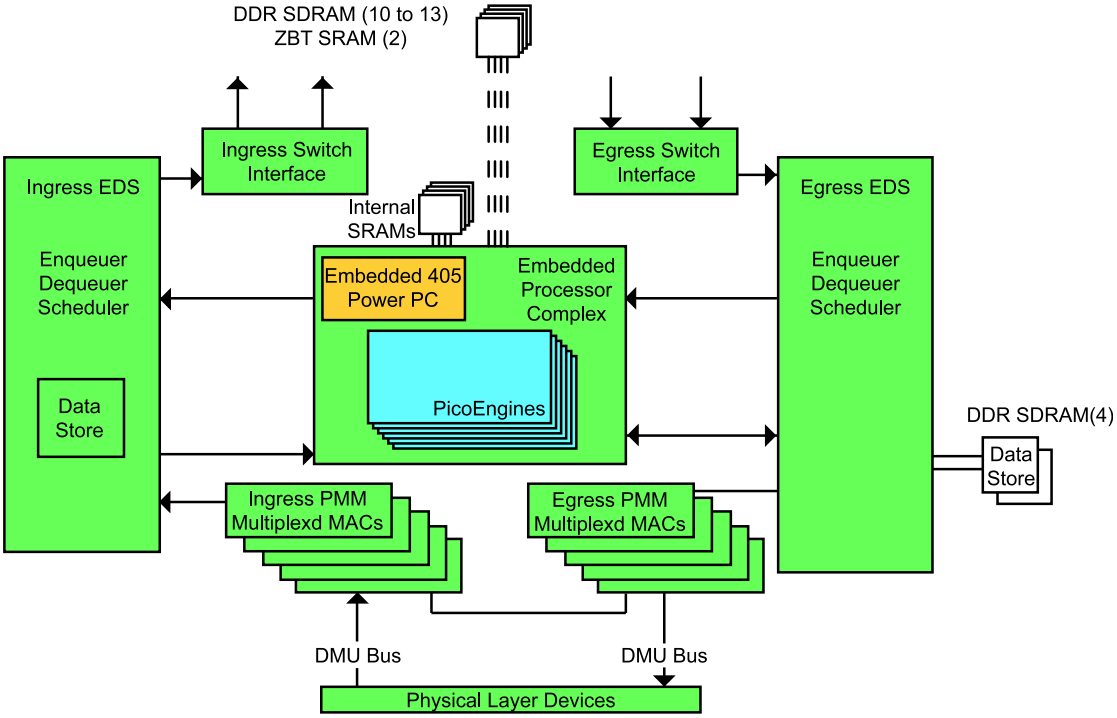


Abbildung 1: Aufbau des IBM PowerNP NP4GS3

2.1.1 Data Mover Units

Der PMM besteht aus fünf Data Mover Units (DMU), wovon vier dieser DMUs (A, B, C und D) unabhängig voneinander als Ethernet Medium Access Control (MAC) oder als Packet Over SONET (POS) MAC konfiguriert werden können. Jede DMU hat eine maximale Transferrate von 1 Gigabit pro Sekunde in die eingehende sowie ausgehende Richtung. Die fünfte DMU wird verwendet, um den vom NP4GS3 generierten Datenverkehr vom Egress EDS zum Ingress EDS zu leiten.

2.1.2 Ingress Enqueuer / Dequeuer / Scheduler

Datenpakete, die an einem DMU ankommen, werden mittels dem PMM an den Ingress Enqueuer / Dequeuer / Scheduler (Ingress EDS) weitergeleitet. Der Ingress EDS speichert die einkommenden Daten in seinem internen Speicher ab, bis genügende Daten gespeichert sind

und er die Daten zur weiteren Verarbeitung an den EPC weitergibt. Sobald der EPC das Datenpaket abgearbeitet hat, leitet er die nötigen forwarding und quality of service Informationen an den Ingress EDS und schreibt das Datenpaket in dessen Speicher zurück. Der Ingress EDS Scheduler leitet die Pakete anschliessend weiter zum Ingress Switch Interface (SWI).

2.1.3 Switch Interface

Das Switch Interface ermöglicht den Anschluss des NP4GS3 an eine externe Switch Fabric oder an einen zweiten PowerNP NP4GS3. Die eintreffenden Datenpakete werden in Datenzellen fragmentiert und via Ata-Aligned Synchronous Link (DASL) Schnittstelle zu einer Switch Fabric oder zu einem zweiten NP4GS3 übertragen oder ans Egress SWI weitergeleitet.

2.1.4 Egress Enqueuer / Dequeuer / Scheduler

Der Egress EDS speichert die eintreffenden Datenzellen im Speicher ab. Sobald ein Datenpaket komplett ist, wird das Paket eingereiht, um vom EPC bearbeitet zu werden. Danach werden die Pakete via dem Egress PMM zum entsprechenden physical layer device gesendet.

2.1.5 Embedded Processor Complex

Der EPC ist der eigentliche programmierbare Teil des Netzwerkprozessors und besteht aus acht Dyadic Protocol Processor Units (DPPUs). Jeder dieser DPPUs hat zwei Core Language Processors (CLPs), 10 Coprozessoren, einen Coprozessor Data Bus, einen Coprozessor Command Bus und einen Shared Memory Pool. Oft wird für die Core Language Processors der Ausdruck Pico Engines verwendet. Pro CPL können je zwei Picocode Threads laufen, somit hat jede DPPU vier Threads. Obwohl 32 unabhängige Threads am Laufen sind, kann ein CPL nur eine Instruktion eines Threads ausführen und nicht zwei Instruktionen beider Threads parallel. Somit werden über alle DPPUs hinaus gesehen immer 16 Threads parallel ausgeführt. Der EPC führt alle Verarbeitungsfunktionen für den NP4GS3 durch. Im Allgemeinen nimmt der EPC die Daten für die Verarbeitung vom Ingress und Egress Enqueuer/Dequeuer/Scheduler an. Der EPC, der mittels Picocode gesteuert und von diversen Coprozessoren unterstützt wird, entscheidet, wie die Datenpakete weitergeleitet werden. Die Datenpakete können an einen abschliessenden Bestimmungsort geleitet oder verworfen werden.
Neben den DPPUs ist im EPC ein embedded PowerPC Subsystem integriert, das aus einem PowerPC 405 133 MHz Core und 64 MBit DRAM Speicher (D6 DRAM) besteht.

2.2 IBM PowerNP Advanced Software Offering

IBM PowerNP Advanced Software Offering (ASO) ist ein Softwarepaket für Entwickler, die mit dem IBM PowerNP NP4GS3 arbeiten. Es soll den IBM Kunden die Entwicklungszeit verkürzen und somit die Zeitspanne Time-to-Market verkleinern, um einen Marktvorteil gegenüber anderen Produkten zu erhalten. Das komplette Advanced Software Offering beinhaltet Sourcecode, User Guides und Dokumentationen der Architektur von Software und Hardware.
Dabei sind folgende Grundfunktionalitäten im ASO enthalten, das in Form von C, C++ und Asembler (Picocode) Code ausgeliefert wird:

- IPv4 Forwarding
- IPv4 Classification
- Multicast
- Diffserv
- MPLS (Edge, Core)
- VLAN, L2 switching

Das ASO besteht aus den in Abbildung 2 dargestellten Hauptkomponenten:

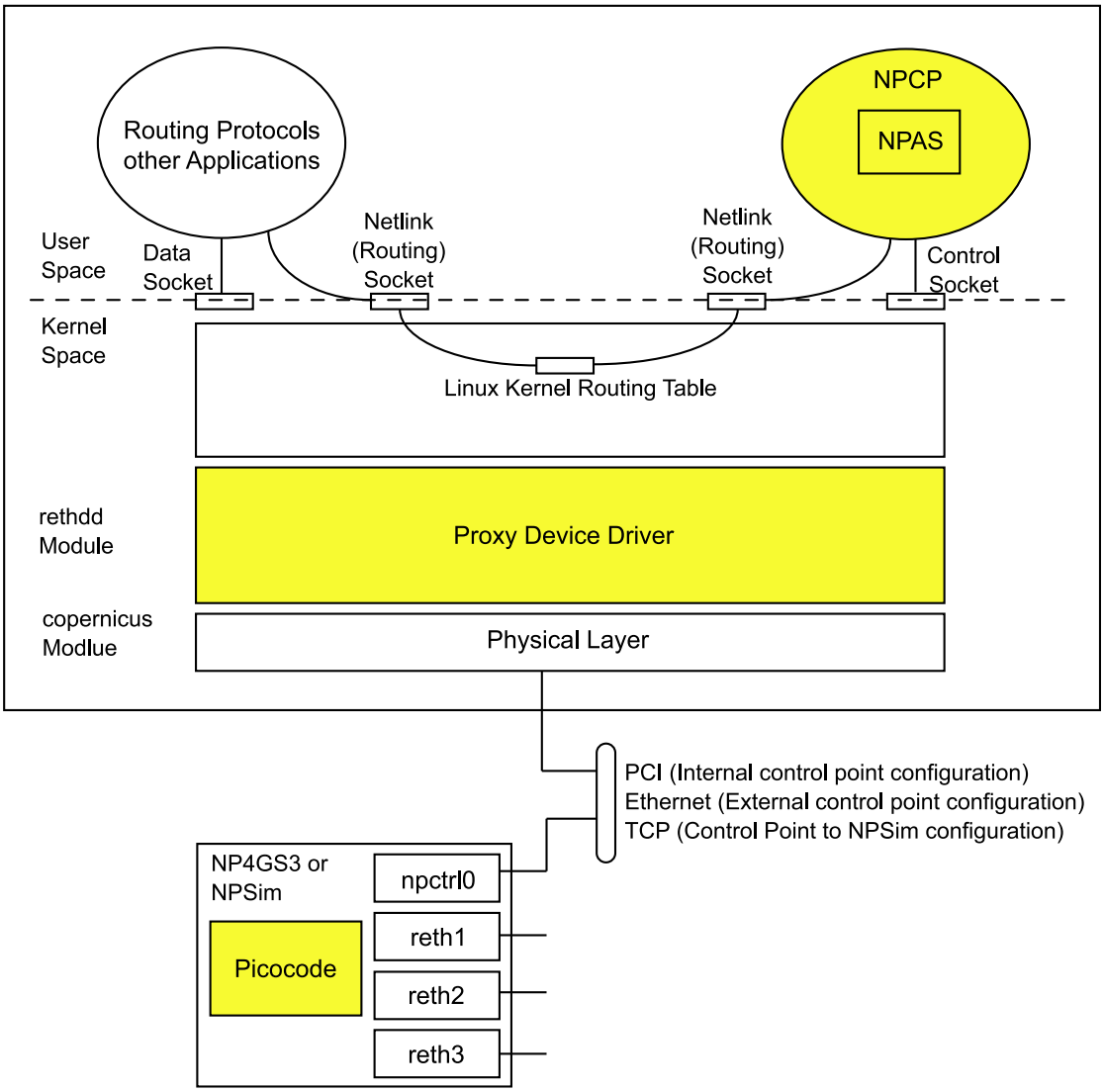


Abbildung 2: Architektur des IBM PowerNP NP4GS3 Advanced Software Offering

- Der Picocode ist in einem speziellen, für den Netzwerk Prozessor ausgelegten Assembler Dialekt geschrieben. Der Picocode wird auf den DPPUs des Netzwerkprozessors ausgeführt.
- Der Proxy Device Driver ist ein in C geschriebenes Linux Kernel Modul, das die Kommunikation zwischen dem Linux Host und dem Netzwerkprozessor ermöglicht. Zudem spiegelt es die Ports des Netzwerkprozessor im Linux Kernel.
- Des weiteren bietet das ASO unfassende Network Processor Application Services (NPAS) APIs zur Konfiguration und Kontrolle des Netzwerk Prozessors und seiner Weiterleitungsfunktionalität.
- Der Network Processor Control Point (NPCP) ist eine Möglichkeit, wie der Prozessor konfiguriert und kontrolliert werden kann. Er verwendet dafür die diversen APIs. Für weitere Entwicklungen kann man auf diesem Control Point aufbauen oder ihn als Beispiel für eine Eigenentwicklung verwenden.

Der Picocode ist zusammen mit den DPPUs der programmierbare Teil des Netzwerkprozessors, währenddem alle anderen Komponenten des Netzwerkprozessors durch eine reine Hardwarelösung realisiert sind. Die Arbeit wird sich deshalb auch auf die EPC mit seinen DPPUs und den Picocode konzentrieren. Der Picocode des ASO 1.3.4 unterstützt viele protokollspezifische

IETF Standards und ist deshalb umfangreich und komplex (siehe Tabelle 1), so dass bei der Diplomarbeit nur auf einen kleinen Teil des Picocodes eingegangen wird.

RFC/Draft	Title	Comments
Draft-ietf-mpls-diff-ext-07.txt	DiffServ-over-MPLS	Description of how IP Differentiated Services is supported in an MPLS network
RFC 791	Internet Protocol	
RFC 826	Ethernet Address Resolution Protocol	Picocode can be configured to recognize ARP packets and redirect them to the control point.
RFC 1075	Distance Vector Multicast Routing Protocol	Picocode can forward IP multicast packets according to DVMRP forwarding tables.
RFC 1112	Host Extensions for IP Multicasting	
RFC 1584	Multicast Extensions to OSPF	Picocode can forward IP multicast packets according to MOSPF forwarding tables.
RFC 1812	Requirements for IP Version 4 Routers	
RFC 2236	Internet Group Management Protocol, Version 2	Picocode can be configured to redirect IGMP packets to the control point.
RFC 2474	Definition of DiffServ Field in IPv4 and Ipv6 headers	The encoding of the DiffServ Code Point (DSCP).
RFC 2475	An Architecture for DiffServ	Definition and general principles of Differentiated Services to enable IP QoS.
RFC 2597	Assured Forwarding PHB	DiffServ behavior definition for better than best-effort services.
RFC 2598	An Expedited Forwarding PHB	DiffServ behavior definition suited for leased-line like services with strict guarantees.
RFC 2697	A Single Rate Three Color Marker	Metering algorithm supported by the Policy Manager/DiffServ.
RFC 2698	A Two Rate Three Color Marker	Metering algorithm supported by the Policy Manager/DiffServ.
RFC 3031	MPLS Architecture	Definition and general principles of MPLS.
RFC 3032	MPLS Label Stack Encoding	Definition of the MPLS shim header bits and IP TTL handling.

Tabelle 1: Durch ASO unterstützte protokollspezifische IETF Standards

3 Architektur

Der erste Teil des Kapitels Architektur geht auf die Möglichkeiten des Netzwerkprozessors ein, die man für PromethOS nutzen könnte. Der mittlere Teil stellt die Architektur vor und erläutert die zu erwartende Leistung. Der Abschluss des Kapitels zeigt die Erweiterungsmöglichkeiten der Architektur.

3.1 IP Routing

Der NP4GS3 hat eine eigene Routing Tabelle, die jeweils vom Control Point aktualisiert wird. Dadurch ist der Netzwerkprozessor in der Lage, selbstständig Datenpakte weiterzuleiten. Zudem besitzt jeder DPPU eine eigene Tree Search Engine, womit ein äusserst effizientes Suchen in Tabellen, und dazu gehört auch die Routing Tabelle, möglich ist. Des weiteren besteht auch die Möglichkeit, dass der Netzwerkprozessor das IP Routing für Pakete übernimmt, die vom Control Point aus gesendet werden, d. h. es werden alle Datenpakete, die vom Control Point aus gesendet werden, dem Netzwerkprozessor übergeben. Dieser sucht dann die entsprechende Route aus der Routing Tabelle und bildet den Ethernetheader, der dem IP Datenpaket vorangestellt wird. Diese Forwarding Funktionalität des Netzwerkprozessors hat den Vorteil, dass das NodeOS von Datenpaketen entlastet wird, die für kein PromethOS Plugin bestimmt sind. Es kann aber auch als Nachteil angesehen werden, dass das NodeOS nicht mehr direkt auf jedes Datenpaket Einfluss nehmen kann.

3.2 Paket Klassifikation

Der Multi-Field Classifier (MF CLS) ist bereits im Picocode des IBM PowerNP NP4GS3 Advanced Software Offering implementiert. Müsste man diesen Multi Field Classifier im OSI Referenzmodel einreihen, so würde er zur Schicht 4 gehören. Das bedeutet, dass nur Datenpakete zum Classifier gelangen, die nicht von einer tieferen Schicht verworfen wurden (z. B. Checksummenfehler).
Das Verhalten des MF CLS wird durch Regeln konfiguriert. Es können bis zu 500 Regeln standardmässig definiert werden. Wird dem MF CLS mehr Speicher zur Verfügung gestellt, können auch noch mehr als 500 Regeln definiert werden. Für eine Regel werden jeweils die oberen und unteren Grenzen eines Wertes festgelegt (z. B. die Ziel IP Adresse eines Datenpaketes). Neben der Regel an sich wird zusätzlich ein Verweis auf eine Aktion gespeichert, die definiert, was mit einem Datenpaket passieren soll, das auf eine Regel zutrifft. Jede Regel hat auch eine Priorität. Trifft ein Datenpaket auf mehrere Regeln zu, so wird die Regel mit der höheren Priorität angewendet. Haben beide Regeln dieselbe Priorität, so wird die zuerst definierte Regel angewandt.
Mit dem Multi Field Classifier wäre somit eine Auslagerung der Datenpaket Klassifikation auf den Netzwerkprozessor möglich. Dadurch würde der Netfilter Prozess im Linux Kernel entlastet. Neben der Selektion, ob ein Datenpaket für PromethOS bestimmt ist, muss zudem noch eruiert werden, für welches PromethOS Plugin das Datenpakete ist. Diese zwei Vorgänge sollen im Multi Field Classifier zu einem einzigen Schritt zusammengefasst werden. Dazu muss zu jeder Regel in der Tabelle noch die zugehörige Plugin ID hinterlegt werden.
Wird der Multi Field Classifier für PromethOS verwendet, muss dieses Framework so erweitert werden, dass die aussortierten Datenpakete mit Plugin ID zum PromethOS Plugin Manager weitergeleitet werden könnten.

3.3 Einsatzmöglichkeiten des NP4GS3

In den folgenden drei Abschnitten wird erläutert, welche Möglichkeiten es für den Netzwerkprozessor gibt, um PromethOS Plugin auszuführen.

3.3.1 Pico Engines und PromethOS Plugins

Auf dem NP4GS3 laufen 32 Threads im EPC. Wieso sollen nicht einige dieser Threads für das Ausführen von PromethOS Plugins eingesetzt werden? Damit wäre es möglich, die gesamte

Behandlung des Paketes im EPC auszuführen. Engpässe an Schnittstellen zu einer externen CPU wären somit Vergangenheit.
Der Instruktionsspeicher hat eine Grösse von 32 Kilobytes und wird von allen Threads gemeinsam benutzt. Diese 32 Kilobytes müssten somit für den Picocode der Grundfunktionalität des NP4GS3 sowie für den Picocode aller zur selben Zeit geladenen Plugins reichen. Da der Picocode nicht zur Laufzeit geändert werden kann, müssten alle Plugin Module von Anfang an in den Instruktionsspeicher geladen werden. Dies würde einerseits die zur Verfügung stehenden 32 Kilobyte bei weitem übersteigen, und anderseits wären dies alles andere als dynamisch ladbare PromethOS Plugin Module.
Neben der Einschränkung durch die Grösse des Instruktionsspeichers gibt es auch weitere Einschränkungen durch die Instruktionen. So sind zum Beispiel der Datentyp Fliesskommazahl und die damit verbundenen Operationen nicht im Netzwerkprozessor implementiert.
In der heute vorliegende Version des ASO 1.3.4 mit dem dazugehörenden Picocode ist es nicht möglich, dynamisch Picocode und somit PromethOS Plugins zu laden. Zudem ist es nicht möglich, jedes PromethOS Plugin, das für die Ausführung auf einem universellen Prozessor entwickelt wurde, so zu implementieren, dass auf den Pico Engines des Netzwerkprozessors ausgeführt werden kann.
Obwohl der EPC ungeeignet ist für das Ausführen von PromethOS Plugins, geht Abschnitt 3.6 darauf ein, welche weiteren Möglichkeiten durch den EPC bestehen, um das NodeOS zu entlasten und einen Performancegewinn für PromethOS zu erhalten.

3.3.2 Embedded PowerPC und PromethOS Plugins

Der NP4GS3 hat im EPC einen embedded PowerPC Subsystem integriert, das aus einem 133 MHz PPC405 Processor Core besteht. Von der Softwarefirma Monta Vista Inc. gibt es eine Linux Distribution [16], die auf dem embedded PowerPC läuft und für den NP4GS3 angepasst wurde. Durch eine Linux Installation auf dem embedded PowerPC wäre es möglich, PromethOS Module auf dieser im Netzwerkprozessor integrierten CPU auszuführen. Die Einschränkungen bei Instruktionen und Datentypen, wie sie bei den Pico Engines vorkommen, wären hier nicht mehr vorhanden. Ein weiterer Vorteil ist die Nähe zu den Pico Engines, womit man sich ein effizientes Weiterleiten von Datenpaketen an der Control Point versprechen kann. Fraglich hingegen ist, ob ein Prozessor mit einer Taktrate von 133 MHz ausreichend ist.

3.3.3 Externe CPU und PromethOS Plugins

Als dritte Variante kann man die PromethOS Plugins auf einer externern CPU ausführen, d. h. die Datenpakete werden vom Netzwerkprozessor via einer Netzwerkverbindung an den PromethOS Rechner zur Bearbeitung durch ein Plugin weitergeleitet. Beim Application Reference Board drängt sich auf, dafür die auf dem Board vorhandene Ethernet-PCI Bridge zu verwenden. Hierfür ist die Data Mover Unit A fest mit einem BroadCom BCM5700 Chip verdrahtet. Über diese on Board Ethernetverbindung kann der Netzwerkprozessor Datenpakete zum BroadCom Netzwerkkartenchip senden und empfangen. Der BCM5700 ist mit den PCI Bus des Rechners verbunden, indem das Application Reference Board eingebaut ist. Der BCM5700 selbst wird über einen gewöhnlichen Treiber für BroadCom Netzwerkkarten angesteuert.
Der Vorteil bei dieser Variante ist, dass man den PromethOS Plugins genügend Systemressourcen zur Verfügung stellen kann. Dieser Rechner kann im Bezug auf CPU Leistung und Arbeitsspeicher je nach Anforderung des NodeOS dimensioniert werden. Als Nachteil muss der weite Signalweg zum PromethOS Plugin gewertet werden.

3.4 Entscheid

Als Grundsatz für den Einsatz von Netzwerkprozessoren soll gelten: So viele Netzwerkaufgaben wie möglich vom Linuxkernel auf den Netzwerkprozessor auslagern, um die Optimierungen des Prozessors nützen zu können. Die aus diesem Grundsatz entstandene Architektur ist in Abbildung 3 dargestellt. Bei der Verwendung des NP4GS3 für PromethOS werden diese Aufgaben auf die DPPUs des NP4GS3 ausgelagert. Datenpakete, die nicht von einem PromethOS Modul bearbeitet werden müssen, sollen nicht in den Linuxkernel gelangen, sondern direkt vom NP4GS3 ausgewertet und weitergeleitet werden. Somit müssen die für PromethOS relevanten

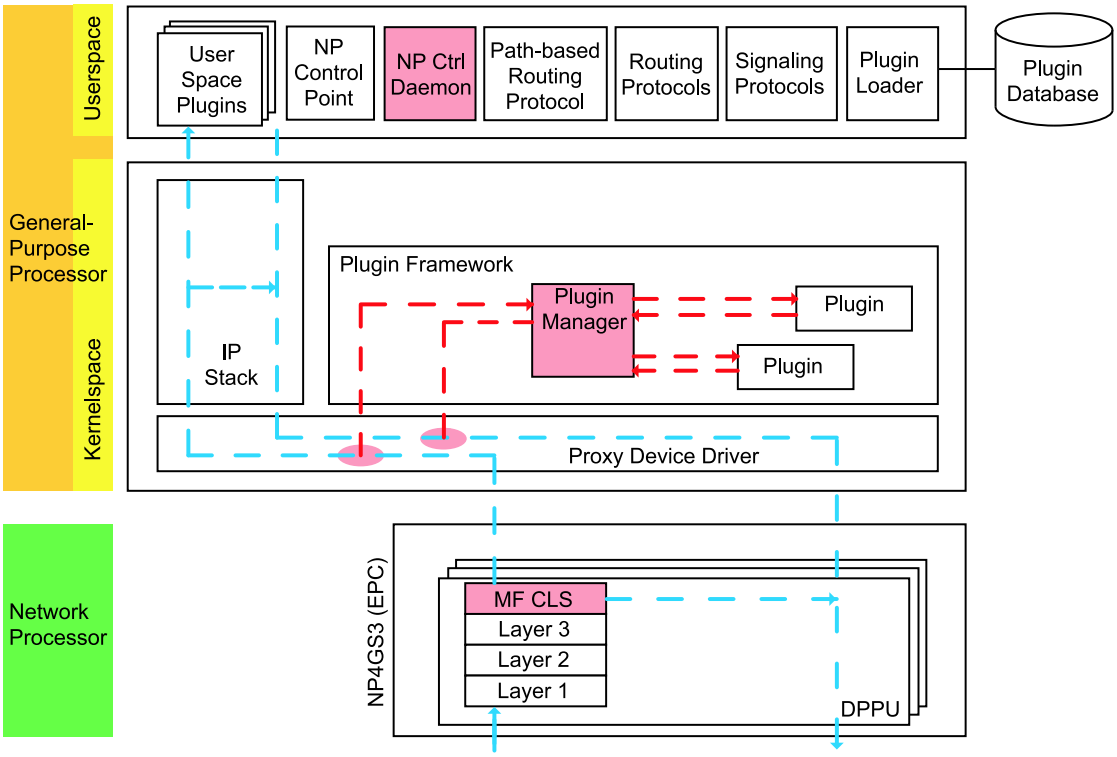


Abbildung 3: Architektur für den Einsatz des NP4GS3 unter PromethOS

Datenpakete aussortiert werden und mit einer möglichst effizienten Technik an den PromethOS Plugin Manager übergeben werden. Alle anderen Datenpakete werden gemäss Routing Tabelle auf den entsprechenden Port weitergeleitet.
Dafür wird der Classifier mit Regeln konfiguriert, die definieren, welche Datenpakete für ein Plugin bestimmt sind. Es können auch mehrere Regeln pro Plugin konfiguriert werden. Jedes geladene Plugin bekommt vom Plugin Manager eine eindeutige Plugin ID zugeteilt. Jedes beim Plugin Manager eintreffende Datenpaket muss eine Plugin ID haben, damit dieser entscheiden kann, welches Plugin das Datenpaket bearbeiten muss. Um pro Datenpaket ein zweifaches Suchen in den Regeln zu vermeiden, muss bereits im Classifier ermittelt werden, für welches Plugin das Datenpaket bestimmt ist. Dazu soll die Plugin ID zu jeder Regel mitgespeichert werden. Trifft eine Regel auf ein Datenpaket zu, wird es zusammen mit der Plugin ID an den Plugin Manager weitergegeben.
Die PromethOS Plugins sollen auf einer externen CPU und auf dem embedded PowerPC ausgeführt werden können. Bei beiden Varianten kann man die Kommunikation zwischen Netzwerkprozessor und dem Plugin Manager auf dem bestehenden Proxy Device Driver aufbauen. Für die Wahl dieser beider Varianten sprechen folgende Gründe:

- PromethOS basiert in der Version v1.0 auf einem Linux Rechner ohne spezielle Hardware. In der Version v2.0 soll es die Option geben, PromethOS durch einen Netzwerkprozessor zu unterstützen. Mit dem Einsatz des Netzwerkprozessors erhofft man sich eine markante Leistungssteigerung von PromethOS. Dieser Linuxrechner entspricht der externen CPU. Um die bis jetzt verwendete Hardware weiter zu unterstützen, wurde die Variante einer externen CPU gewählt. Ein weiteres Argument für die Wahl dieser Variante ist die grosse Rechenleistung, die mit einer externen CPU zur Verfügung gestellt werden kann.
- Ein Ziel der Diplomarbeit ist die Installation von Linux auf dem embedded PowerPC. Einerseits geht es um die grundlegende Frage, wie man Linux auf dem Application Reference Board starten kann, andererseits will man Aufschluss über die Leistungsfähigkeit des embedded PowerPC im Bezug auf die Verwendung für PromethOS haben.
- Die Variante, Pico Engines für das Ausführen von PromethOS Plugins zu verwenden, wird aus den in Abschnitt 3.3.1 erläuterten Gründen verworfen.

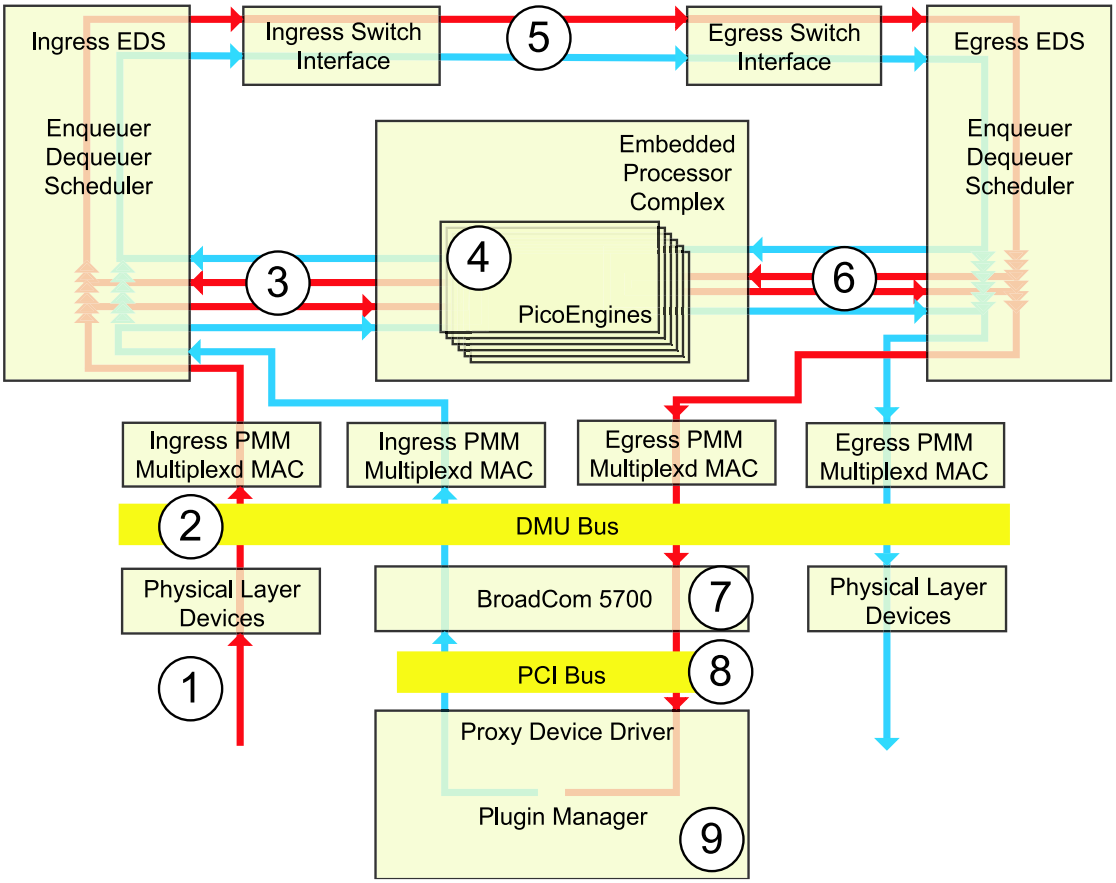


Abbildung 4: Verlauf eines Datenpaketes (Port - NP - Plugin Manager - NP - Port)

Weil das gesamte IP Routing im Netzwerkprozessor ausgeführt wird, ist es nicht notwendig, dass die Datenpakete an den IP Stack übergeben werden. Die Datenpakete sollen direkt von Proxy Device Driver zum Plugin Manager übergeben werden, um wertvolle Zeit zu sparen und die Leistungsfähigkeit von PromethOS hoch zu halten.

3.5 Performance

Um die erwartete Performance zu berechnen, muss man sich Überlegungen zu möglichen Limiten im System machen. Es gibt Limiten, die sich unterscheiden je nach dem, ob der Plugin Manager auf dem embedded PowerPC oder auf einer externen CPU abläuft. In Abbildung 4 ist der Weg, den ein Datenpaket vom Empfangen bis zum Senden durchläuft, illustriert. Schematisch abgebildet sind die Hauptkomponenten des Netzwerkprozessors sowie der Host (9), in dem das Application Reference Board eingebaut ist. Mit Rot ist der Pfad eingehender Datenpakete bis zum Plugin Manager gekennzeichnet, mit Blau der Weg vom Plugin Manager bis zum sendenden Port. Anhand dieser Grafik werden die vorhandenen Limits erklärt und beurteilt.

Data Mover Units und Physical Layer Devices Das Application Reference Board ist mit 3 externen GBit-Ethernet Anschlüssen (nur einer dargestellt) ausgestattet. Jeder dieser Physical Layer Devices (1) kann maximal 1 GBits senden und 1 GBits empfangen (Full Duplex). Der vierte GBit-Ethernet Anschluss wird für die Ethernet-PCI Bridge verwendet. Für den Anschluss der Physical Layer Devices hat es auf der Ingress wie auf der Egress Seite je 5 Data Mover Units (2), die eine maximale Datentransferrate von 1 GBits pro Sekunde haben. Drei Paare (jeweils Ingress und Egress) dieser Data Mover Units sind mit den Physical Layer Devices verbunden, ein weiteres Paar übernimmt den Anschluss des BroadCom 5700 (7). Das letzte Paar ist miteinander verbunden, um Datenpakete von der Egress Seite auf die Ingress Seite des NP4GS3 zu transferieren (hier nicht abgebildet). Die verwendete Hardware ist also in der Lage, auf allen

vier Anschlüssen die volle Bandbreite von GBit-Ethernet auszunützen. Eine wichtige Feststellung ist, dass es nicht möglich ist, mehr als 1 GBits pro Sekunde von der Egress Seite auf die Ingress Seite zu transferieren.

Switch Interface Über das Switch Interface (5) kann der Netzwerkprozessor an eine Switch Fabric oder an einen zweiten Netzwerkprozessor angeschlossen werden. Ist der Prozessor weder auf die eine noch auf die andere Art angeschlossen, werden die Datenpakete von Ingress Switch Interface direkt an das Egress Switch Interface übergeben. In [11] wird dem Switch Interface eine Transferrate von 3.25 bis zu 4 GBits pro Sekunde zugeschrieben. Beachtet man, dass die Netzwerkanschlüsse des NP4GS3 zusammen eine Bandbreite von 4 GBits pro Sekunde haben, so ist das Switch Interface genügend dimensioniert, und es sollte keine Probleme bezüglich der Performance geben.

Embedded Processor Complex Beim EPC wird nicht das komplette Datenpaket geladen, sondern nur die notwendigen Headers. In der Regel werden die ersten 64 Bytes des Datenpaketes geladen. Wie und was genau in den EPC geladen wird, ist durch den Picocode gesteuert und ist somit anpassbar. Das Kopieren der Daten von Ingress EDS Speicher (3) oder vom Egress EDS Speicher (6) in den EPC wird vom DataStore Coprozessor ausgeführt. Der maximale Durchsatz ist also abhängig davon, wie viele Bytes pro Datenpaket in den EPC kopiert werden, wie schnell der DataStore Coprozessor kopieren kann und wie der Picocode programmiert ist (wird das Kopieren parallel zu anderen Operationen des Threads ausgeführt, oder muss der Thread auf den Coprozessor warten). Dieser Kopiervorgang ist aber nur ein Teil des Ganzen. Jedes Datenpaket wird mindestens einmal auf der Ingress und der Egress Seite von einem Thread (4) bearbeitet. Schliesslich ist entscheidend, wie lange die Threads zusammen für die Bearbeitung eines Datenpaketes brauchen. Diese Zeit ist umgekehrt proportional zur Leistungsfähigkeit des EPCs. Diese Ausführungszeit der Threads pro Datenpaket ist von der Komplexität der Aufgabe des Netzwerkprozessors abhängig. In [13, Table 5. Empirical Performance of NP4GS3B] ist ersichtlich, dass der Netzwerkprozessor mit der Aufgabe des Layer 3 IP Forwarding eine maximale Transferrate von 2.66 bis 3.05 GBits pro Sekunde hat. Da beim Einsatz für PromethOS noch die Paketklassifikation hinzukommt, kann man daraus schliessen, dass die Ausführungszeit grösser ist als beim ausschliesslichen Layer 3 IP Forwarding. Zudem passiert ein Datenpaket vom Eintreffen am empfangenden Port bis zum Verlassen des sendenden Ports zweimal den Netzwerkprozessor (siehe rote und blaue Linie). Aus der Erkenntnis, dass der Aufwand für PromethOS somit mindestens doppelt so gross ist wie für Layer 3 IP Forwarding, lässt sich die Schlussfolgerung ziehen, das der maximale Durchsatz nicht grösser als 1.5 GBits pro Sekunde ist.

Schnittstelle externe CPU Eine externe CPU ist über eine speziell dafür vorgesehene Data Mover Unit angeschlossen. Durch die Begrenzungen dieser DMU wird der maximale Durchsatz auf 1 GBits pro Sekunde gesenkt. Das bedeutet, dass die errechnete Leistungsfähigkeit des EPC von 1.5 GBits pro Sekunde gar nicht ausgeschöpft werden kann. Die DMU ist an einem BroadCom 5700 Netzwerkkarten Chip angeschlossen, der in der Lage ist, ein GBits pro Sekunde Full Duplex zu verarbeiten. Der BroadCom Chip wiederum ist mit dem PCI Bus (8) des Hosts verbunden.

PCI 2.3: 32-bit/33 MHz	1.1 GBits/sec
PCI 2.3: 64-bit/66 MHz	4.3 GBits/sec
PCI-X 1.0: 64-bit/133 MHz	8.5 GBits/sec

Tabelle 2: Von BroadCom 5700 unterstützte PCI Standards und max. Durchsatz

Um den BroadCom Gigabit-Ethernet Controller mit genügend Daten zu versorgen, muss der PCI Bus mindestens einen Durchsatz von 2 GBit haben, da der PCI Bus kein Full Duplex unterstützt. In Tabelle 2 sind die vom BroadCom Gigabit-Ethernet Controller unterstützten PCI Standards aufgelistet. Kommt ein PCI 32-bit/33 MHz PCI auf dem Host zum Einsatz, so hat man mit einen maximalen Durchsatz von 0.55 GBits/sec Full Duplex zu rechnen. Dieser Maximalwert ist aber in der Praxis nicht erreichbar, da der PCI Bus noch von anderen Komponenten

des Hosts verwendet wird. Um die geforderten 1 GBits/sec Full Duplex garantieren zu können, muss somit mindestens ein PCI 64-bit/66 MHz Bus verwendet werden.

Schnittstelle embedded PowerPC Die Schnittstelle zum embedded PowerPC ist über einen gemeinsamen Speicherbereich im D6 DRAM Speicher realisiert. Der General PowerPC Handler (GPH) Thread kopiert die Datenpakete in den D6 DRAM Speicher und signalisiert dies dem embedded PowerPC über einen Interrupt. Der umgekehrte Vorgang führt der embedded PowerPC aus, um ein Datenpaket zu senden. Es ist relativ schwer abzuschätzen, wie gross die Leistungsfähigkeit dieser Schnittstelle ist. Aufschluss darüber geben die in Abschnitt 5.4 dokumentierten empirischen Messungen.

Fazit Werden die PromethOS Plugins auf einer externen CPU ausgeführt, kann unter der Voraussetzung eines 64-bit PCI Busses mit einem maximalen Durchsatz von 1 GBits/sec gerechnet werden. Dies lastet den EPC mit seinen Pico Engines zu zwei Drittel aus. Mit der noch vorhandenen Kapazität wäre der Netzwerkprozessor in der Lage, das Layer 3 IP Forwarding für zusätzliche 1 GBits/sec zu übernehmen.

3.6 Erweiterungsmöglichkeiten

In der bis jetzt beschriebenen Architektur wird der Netzwerkprozessor für die Aufgaben der Klassifikation der Datenpakete und das IP Routing verwendet. An dieser Architektur ist aber noch unbefriedigend, dass die PromethOS Plugins an sich keine Möglichkeiten haben, die Vorteile und Optimierungen des Netzwerkprozessors für sich zu nutzen, denn die Plugins werden komplett auf einem universellen Prozessor ausgeführt. In den folgenden Abschnitten wird ein Ansatz aufgezeigt, wie dieser Schwachpunkt der oben erläuterten Architektur behoben werden kann. Neben der Beschreibung der Architektur wird im Abschnitt 6 eine Performance Messung der Pico Engines durchgeführt. Eine Implementation dieser Architekturerweitungen würden den Rahmen dieser Diplomarbeit sprengen. Es ist wichtig zu zeigen, dass diese Erweiterung auf der Architektur, die in dieser Diplomarbeit implementiert wird, aufbaut. In Abschnitt 3.3.1 wurde aufgezeigt, dass es nicht möglich ist, PromethOS Plugins generell auf den Pico Engines auszuführen. Der gesuchte Architekturansatz muss auf diese Einschränkungen der Pico Engines sowie auf die Anforderungen eine PromethOS Plugins eingehen.

3.6.1 Pico Plugins

Durch Pico Plugins haben die PromethOS Plugins die Möglichkeit, Teile ihre Funktionalität auf die Pico Engines auszulagern. Dadurch ergeben sich für die PromethOS Plugins neue Möglichkeiten:

- Datenpakete können direkt auf den Pico Engines bearbeitet und die zu Verfügung stehenden Hardwareoptimierungen wie z. B. Checksummen-Coprozessor oder TreeSearch-Coprozessor verwendet werden. Wird das Datenpaket komplett in den Pico Engines bearbeitet, entfällt auch die Weiterleitung an den Plugin Manager.
- Durch Pico Plugins wäre auch die Möglichkeit für eine zusätzliche Selektion der Datenpakete geschaffen. Datenpakete, die im Multi Field Classifier auf eine Regel zutreffen, können vom Pico Plugin, z. B. anhand des Inhaltes eine Datenpaketes, weiter selektiert werden.
- Ein mögliches Szenario wäre auch, dass im Pico Plugin entschieden wird, ob das Datenpaket an das eigentliche PromethOS Plugin weitergeleitet wird oder ob das Datenpaket direkt im Pico Plugin selbst verarbeitet werden kann.

Durch diese Pico Plugins werden die Pico Engines für PromethOS Plugins erschlossen. Übersteigt die Anforderung eines Plugins die Möglichkeiten der Pico Engines, so kann das Datenpaket an den Plugin Manager und somit an das PromethOS Plugin weitergeleitet werden und auf einem universellen Prozessor bearbeitet werden.
Die Pico Plugins werden ausgeführt, nachdem die Datenpaketklassifikation statt gefunden hat, d. h. trifft eine Datenpaket auf eine Regel des Classifiers zu, so überprüft der Classifier anhand

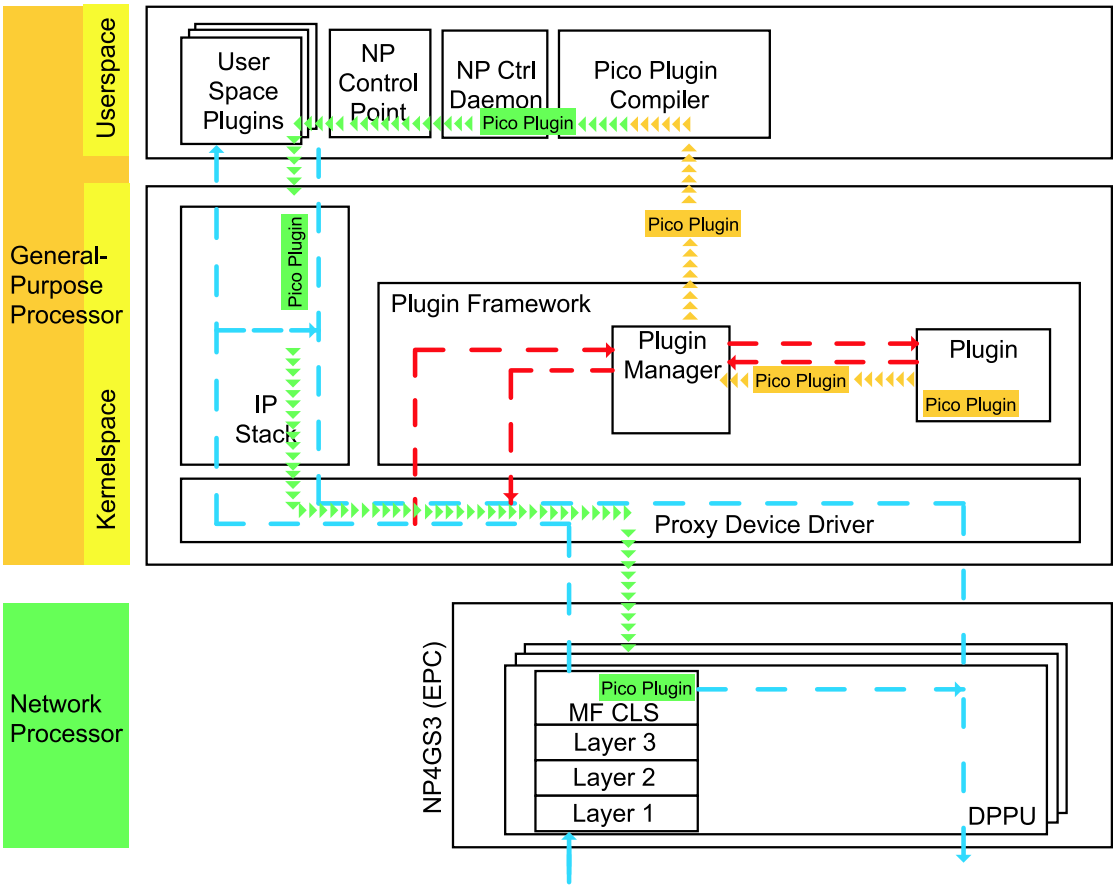


Abbildung 5: Verifizieren und Kompilieren eines Pico Plugins

der Plugin ID, ob ein Pico Plugin mit identischer Plugin ID geladen ist. Falls ja, wird es ausgeführt. Ist kein Pico Plugin mit entsprechender Plugin ID vorhanden, wird das Datenpaket an den Plugin Manager weitergeleitet.
Um den Netzwerkprozessor vor einer Überlast von Pico Plugins zu schützen, muss es die Möglichkeit geben, ein Pico Plugin abzulehnen. Somit muss das PromethOS Plugin entsprechend programmiert sein, dass es auch funktionsfähig ist, wenn es kein Pico Plugin einsetzen kann.

3.6.2 Pico Plugin Programmiersprache

Da der EPC kein Betriebssystem hat, das die einzelnen Threads steuert und kontrolliert, gibt es für die geladenen Pico Plugins auch keine Einschränkung in Bezug auf Speicherzugriff und Zeitdauer einer Belegung der Pico Engine. Somit müsste, bevor das Pico Plugin geladen wird, verifiziert werden, ob das Pico Plugin korrekt abläuft. Das heisst, es müsste geprüft werden, ob es terminiert und keine illegalen Operationen, wie z. B. das Verändern des Instruktionsspeichers oder ein Lesezugriff auf fremde Datenpakete, ausführt. Dies kann sich aber je nach Programmiersprache als äusserst schwierig erweisen.
Man muss also die grundlegende Überlegung machen, in welcher Form diese Pico Plugins programmiert werden sollen. Die gewählte Programmiersprache muss die besonderen Fähigkeiten des Netzwerkprozessors ausnützen können und durch ihre Eigenschaften eine Programmverifikation möglichst gut unterstützen. Die Pico Plugins werden somit in Form dieser höheren Programmiersprache an den Plugin Manager übergeben (in Abbildung 5 gelb dargestellt). Danach wird das Pico Plugin verifiziert und falls korrekt, durch einen Compiler übersetzt (grün dargestellt). Durch den Einsatz eines Compilers wäre es möglich, diese Pico Plugins für verschiedene Netzwerkprozessoren einzusetzen.

3.6.3 Dynamisches Laden von Pico Plugins

In der aktuellen Version des Advanced Software Offering wird an zwei Zeitpunkten Picocode in den Instruktionsspeicher geladen. Das erste Mal, wenn das Application Reference Board bootet, wird der Bootpicocode aus dem Flashspeicher geladen. Der Bootpicocode initialisiert den Netzwerkprozessor und bringt ihn in einen definierten Zustand. Und ein zweites Mal beim Start des Control Points wird der eigentliche Picocode in den Instruktionsspeicher geladen. Ab diesem Zeitpunkt wird am Picocode nichts mehr verändert. Ein Ansatz für das dynamische Laden von Picocode müssten folgenden Gegebenheiten berücksichtigen:

- Der geladene Picocode wird von allen 32 Threads gleichzeitig verwendet, d. h. der Picocode besitzt mehrere Einsprungstellen, bei denen eine Thread gestartet werden kann. Je nach Aufgabe des Threads wird eine andere Einsprungstelle gewählt. Damit ein Bereich des Instruktionsspeichers geändert werden kann, muss garantiert werden können, dass keiner der 32 Threads an dieser Stelle des Instruktionsspeichers Picocode ausführt.
- Der EPC besitzt kein eigenes Betriebssystem, es werden lediglich durch einen in der Hardware implementierten Dispatching Prozess Threads gestartet, die Picocode ausführen und anschliessend wieder terminieren, bis sie erneut gestartet werden. Der EPC kann somit nicht selber das Verwalten des Picocodes übernehmen.
- Ein Anhalten aller Thread, um den Picocode zu ändern, ist aus Performancegründen nicht möglich.

Der Plugin Manager kann die Verifikation der Pico Plugins übernehmen, da diese von der Netzwerkprozessor Architektur unabhängig ist. Der Pico Plugin Compiler ist natürlich sehr abhängig vom Typ des Netzwerkprozessors. Somit müsste man auch mehrere Compiler implementieren, um mehrere Netzwerkprozessoren zu unterstützen. Die Funktionalität der Picocode Verwaltung wird in NP Control Daemon integriert, da der NP Control Daemon über die Schnittstelle des NP Control Point Einfluss auf den Picocode nehmen kann und es eine prozessorspezifische Implementation ist.
Wie genau dieses dynamische Laden von Picocode funktioniert und wie die Codeverwaltung Implementiert sein soll, ist nicht mehr Thema dieser Arbeit und wird eventuell in weiteren Diplom- oder Semesterarbeiten (siehe Abschnitt 7.2) bearbeitet.

4 Implementation

Um die gewählte Architektur zu implementieren, wird der Proxy Device Driver erweitert. Für die Konfiguration des Multi Field Classifiers des Netzwerkprozessors wird eine NP Control Daemon mit NP Control als Client entwickelt. Weiter müssen einige Änderungen am bestehenden Picocode vorgenommen werden, denn es soll bereits im Multi Field Classifier die Plugin ID eruiert werden. Im Abschnitt 4.4 werden die nötigen Schritte erklärt, um PromethOS auf dem embedded PowerPC zu installieren.

4.1 Erweiterung des Multi Field Classifier

Bei der Erweiterung des Classifiers wurde als Voraussetzung angenommen, dass der PromethOS Plugin Manager und der NP Control Point auf dem selben Linux Host ausgeführt werden. Der Classifier wurde nun so erweitert, dass Datenpakete, die auf eine Regel eines PromethOS Plugins zutreffen, an den Plugin Manager weitergeleitet werden. Die oben getroffene Annahme ist aber nicht zwingend, es wäre auch möglich, dass der Control Point und der Plugin Manager auf zwei verschiedenen CPUs laufen. Dafür müsste der Picocode entsprechend angepasst werden.

4.1.1 Das Action Attribute PromethOS

Trifft eine Regel auf ein Datenpaket im Multi Field Classifier zu, so wird im Picocode eine für diese Regel definierte Action ausgeführt. Im Advanced Software Offering 1.3.4. sind die drei verschiedenen Action Attributes «Filter», «DiffServ» und «Redirection» implementiert. Durch diese Attribute und ihre Parameter lassen sich verschiedene Actions bezw. Action Types definieren. Mit dem «Redirection» Attribute ist es bereits möglich, Datenpakete an den Control Point weiterzuleiten. Somit soll die Erweiterung auf dieser Weiterleitung an den Control Point aufbauen.
Um die Möglichkeiten offen zu halten, zu einem späteren Zeitpunkt doch noch Teile eines PromethOS Plugins als Picocode auf dem DPPU auszuführen, wurde ein neues Action Attribute für PromethOS eingeführt. Damit ist es möglich, dass ein Datenpaket, das auf eine Regel zutrifft, im EPC verarbeitet werden kann, ohne dass es an den Plugin Manager weitergeleitet werden muss. Zum jetzigen Zeitpunkt muss aber der Action Type für Regeln eines PromethOS Plugins das Action Attribute «PromethOS» und «Redirection» enthalten.
Die Action Types und Regeln des Classifiers werden mit den Multi Field Classifier APIs [5] konfiguriert. Der Action Type für ein PromethOS Plugin muss wie folgt definiert werden:

```
/*--- Definition Action Type im Control ---*/
/*--- Point (oder via Proxy API)        ---*/
addaction.attributeList = NP_CLS_REDIR_ATTR|NP_CLS_PROMETHOS_ATTR;
...
np_cls_add_ActionType(user_handle, &addaction);
```

Durch das Setzen des Action Attribute `CLS_PROMETHOS_ATTR` wird im Picocode durch den Classifier die PromethOS Plugin ID ermittelt. Damit das Datenpaket zum Control Point und somit zum Plugin Manager weitergeleitet wird, muss zudem das Action Attribute `CLS_REDIR_ATTR` gesetzt sein. Bei der Redirection muss definiert werden, welche Art von Redirection ausgeführt werden soll. Mit

```
actionData.redirectAttr.redir_sel =  NP_CLS_REDIR_SEL_FIXED;
```

wird bestimmt, dass die Datenpakete zum Control Point weitergeleitet werden, weitere Parameter sind dazu nicht nötig.

4.1.2 Zuordnung der Plugin ID

Die Regeln für den Classifier werden in einer speziell dafür angelegten Baumdatenstruktur im Speicher des Netzwerkprozessors abgelegt. Jedes Blatt dieses Baumes enthält eine Regel. Neben den Pattern Feldern, die der Regel entsprechen, gibt es auch ein Feld `UserData` in

jedem Blatt. In `UserData` ist der Action Type und die Parameter zu den Action Attributes abgespeichert, die zur Regel gehören. Trifft eine Regel auf ein Datenpaket zu, wird dieses Feld in ein Array Register geladen. Die Plugin ID zu einer Regel muss somit in diesem Feld gespeichert werden. Da es keine freien Bytes im Feld mehr gibt und eine Erweiterung dieses Feldes sich als äusserst komplex erwies, wurde der Speicherplatz für einen bereits bestehenden Parameter verwendet. Der Parameter `nextHopIpAddr` wird für eine Redirection zu einem anderen IP Host (nicht Control Point) verwendet. Da diese Funktionalität im Zusammenhang mit PromethOS aber nicht benötigt wird, wird bis auf weiteres der Wert der Plugin ID dort gespeichert. Die Werte für das Feld `UserData` leitet das Multi Field Classifier APIs aus den gesetzten Parametern von `actionData` ab.

```
actionData.redirectAttr.u_redir.pbIpRoute.nextHopIpAddr = plugin_id;
np_cls_addStaticRule_tcp(.., actionData, ..);
```

Dieselbe Problematik stellt sich bei der Frage, wie die Plugin ID von Classifier zum Plugin Manager übermittelt wird. Datenpaketen, die an den Control Point weitergeleitet werden, werden vorne ein zusätzlicher Frame Header angefügt. Dieser 32 Bytes umfassende Header enthält Informationen über Sourceblade und Sourceport des Datenpaketes. Eine Erweiterung dieses Headers erwies sich ebenfalls als äusserst aufwendig. Deshalb wurde eine Lösung gesucht, bei der die Plugin ID im Datenpaket selbst gespeichert werden kann. Die eintreffenden IP Datenpakete durchlaufen im EPC des NP4GS3 alle notwendigen Überprüfungen des Datenpaketes, so auch die Kontrolle mittels der Checksumme im IP Header. Eine weitere Überprüfung ist in PromethOS nicht vorgesehen. Diese 16 Bits werden für das Speichern der Plugin ID verwendet. Bei einem allfälligen Senden dieses Datenpaketes ist eine Neuberechnung der IP Checksumme zwingend.

4.2 Erweiterung Proxy Device Driver

Der Proxy Device Driver ist ebenfalls im Advanced Software Offering enthalten und als Kernelmodul implementiert. Durch das Modul wird die Kommunikation mit dem Netzwerkprozessor ermöglicht. Die Kommunikation zwischen Control Point und dem Guided Frame Handler, der auf dem EPC als Enhanced Thread läuft, nennt sich Guided Traffic. Der Guided Traffic beinhaltet sämtliche Kommunikation, die für die Konfiguration und Steuerung des Netzwerkprozessors nötig ist.
Des weiteren spiegelt der Proxy Device Driver die physikalischen Ports des Netzwerkprozessors als Linux Netzwerk Interface mit den Namen reth1, reth2 und reth3. Pakete, die von einer Applikation an eines dieser Linux Netzwerk Interfaces gesendet werden, werden vom Proxy Device Driver zusammen mit einem zusätzlichen Header an den Netzwerkprozessor geschickt und anschliessend vom Netzwerkprozessor auf den entsprechenden Port gesendet. Das Umgekehrte gilt für eintreffende Pakete, die an eine IP Adresse der reth Linux Interfaces adressiert sind. Diese Datenpakete werden vom Netzwerkprozessor mittels des Proxy Device Drives an den IP Stack übergeben. Als empfangendes Linux Interface wird die Referenz auf das entsprechende reth Netzwerk Interface im Socket Buffer gespeichert. Diese Kommunikation zwischen einer Applikation im Userspace und dem Netzwerkprozessor nennt sich Data Traffic.
Für das Senden und Empfangen von Guided Traffic stellt der Proxy Device Driver das Linux Netzwerk Interface npctl0 bereit. Das Netzwerk Interface npctl0 abstrahiert somit den Kommunikationsweg für den Control Point. Analog abstrahieren die reth1 bis reth3 den Kommunikationsweg für Data Traffic. Es gibt grundsätzlich drei mögliche Konfigurationen für diese Kommunikationswege.

- PCI-Attached Internal Control Point Configuration: Der Control Point wird auf dem K2 Control Processor Board ausgeführt und die Kommunikation zwischen dem Control Point und dem Netzwerkprozessor findet über den PCI Bus statt. Diese Variante der Konfiguration ist nur bei der Reference Plattform und nicht beim Application Reference Board möglich, da beim Application Reference Board der PCI Bus des Netzwerkprozessors nicht mit dem PCI Bus des Rechners, auf dem der Control Point läuft, verbunden ist.
- Embedded PowerPC Processor Internal Control Point Configuration: Der Control Point läuft auf dem embedded PowerPC und die Kommunikation zwischen dem Control Point

und dem Netzwerkprozessor findet über einen gemeinsamen Speicherbereich im D6-Memory statt.

- Ethernet-Attached External Control Point Configuration: Bei dieser Variante wird zwischen dem Control Point und dem Netzwerkprozessor via einer Ethernetverbindung kommuniziert. Der Control Point kann somit auf einem beliebigen Linuxhost installiert sein. Beim Application Reference Board [2] ist diese Verbindung auf dem Board integriert. Auf dem Board hat es einen Broadcom BCM5700 PCI-X 1000BASE-T Media Access Controller [14], der fest mit der Data Mover Unit A verbunden ist. Auf dem Linuxrechner, in dem das Application Reference Board installiert ist, wird durch das Laden des BroadCom Treibers die Ansteuerung des BroadCom Netzwerkkarten Chips ermöglicht. Über diese Punkt-Punkt Ethernetverbindung zwischen dem Netzwerkprozessor und dem Linuxrechner mit dem Control Point wird der Guided Traffic sowie der Data Traffic gesendet.

Der Proxy Device Driver soll nun so erweitert werden, dass Datenpakete, die vom Multi-Filed Classifier als Datenpakete für eine PromethOS Plugin erkannt wurden (PromethOS Traffic), auf eine möglichst effiziente Art an den Plugin Manager übergeben werden. Bei der Erweiterung des Plugin Managers muss sichergestellt werden, dass die bestehenden Kommunikationswege für den Control Point oder für die gespiegelten Linux Interfaces nicht unterbrochen oder verlangsamt werden.

4.2.1 Umgehung des IP Stacks

Es wird bereits im Multi-Field Classifier bestimmt, ob ein Datenpaket von einem PromethOS Plugin bearbeitet werden muss, und falls ja, welche ID das Plugin hat. Der Bestimmungsort für das Datenpaket ist beim Eintreffen im Proxy Device Driver somit bestimmt. Es ist keine weitere Klassifizierung (z. B. durch Netfilter) mehr nötig. Um eine hohe Performance zu erreichen, soll deshalb der IP Stack umgangen und die Datenpakete vom Proxy Device Driver direkt dem Plugin Manager übergeben werden. Dabei gilt es, eine Lösung zu finden, die folgendes erfüllt:

- Die bereits implementierten Kommunikationswege für Guided Traffic und Data Traffic müssen weiter bestehen und ihre Leistungsfähigkeit darf sich nicht verschlechtern
- Die Schnittstelle zum Plugin Manager muss die drei Konfigurationen des Kommunikationsweges abstrahieren

Als erstes wurde der Proxy Device Driver analysiert, um eine geeignete Stelle im Code zu finden für die Erweiterung. Als zweiter Schritt wurde eine Schnittstelle definiert für den Datenpaketaustausch zwischen dem Proxy Device Driver und dem Plugin Manager. Nach der Analyse und der Definition wurde die Schnittstelle im Proxy Device Driver implementiert.

4.2.2 Schnittstelle zum Plugin Manager

Die Schnittstelle zwischen dem Proxy Device Driver und dem Plugin Manager wurde mit vier Downcall und einer Upcall Funktion realisiert. Die Funktionen sind in `./src/pts/linux/ethernet/np_promethos.h` definiert. Für das Senden von Datenpaketen wurden zwei Funktionen definiert.

```
int promethos_send( struct sk_buff *skb );

int promethos_send_raw( struct sk_buff *skb,
                        unsigned short blade,
                        unsigned short port );
```

Mit der Funktion `promethos_send()` werden Datenpakete in Form eines Socket Buffers zum Senden an den Proxy Device Driver übergeben. Der Socket Buffer beinhaltet nur das Layer 3 IP Paket. Das IP Routing und das Erstellen des Layer 2 Headers wird durch den Netzwerkprozessor ausgeführt. In speziellen Situationen kann auch die Funktion `promethos_send_raw()` verwendet werden. Bei dieser Funktion wird kein IP Routing im Netzwerkprozessor durchgeführt. Somit muss der Socket Buffer den benötigten Layer 2 Header enthalten und beim Aufruf der Funktion angegeben werden, auf welcher Blade und auf welchem Port das Datenpaket gesendet werden soll.

```
int promethos_register(promethos_upcall_f func);
int promethos_unregister(promethos_upcall_f func);
```

Um Datenpakete zu empfangen, muss beim Proxy Device Driver eine Upcall Funktion registriert werden. Mit dem Funktionsaufruf `promethos_register()` kann eine Funktion vom Typ `promethos_upcall_f` registriert werden. Sobald die Funktion registriert ist, wird für jedes empfangene Datenpaket, das für den Plugin Manager bestimmt ist, die registrierte Funktion aufgerufen. Neben dem Verweis auf den Socket Buffer wird als erster Parameter die Plugin ID übergeben.

```
typedef int (*promethos_upcall_f) ( unsigned short pluginId,
                                    struct sk_buff *skb );
```

Ist eine Empfangsfunktion registriert, kann keine weitere registriert werden. Erst durch das Abmelden der Empfangsfunktion durch den Funktionsaufruf `promethos_unregister()` ist dies wieder möglich. Trifft ein Datenpaket für ein PromethOS Plugin ein und ist keine Empfangsfunktion registriert, wird das Datenpaket in den IP Stack weitergeleitet.

4.2.3 Pakete empfangen

Ethertype, Subtype	Inbound Outbound	Data Control	Header Information	Contents	Comment
Guided Traffic					
C100, 0C	Inbound	Control		Frame control, guided command, end delimiter	The subtype (0C) is part of the Frame Control
C100, 0C	Outbound	Control		Frame control, guided commands, end delimiter	For local or remote PowerNP Network Processor. The subtype (0C) is part of the Frame Control
Inbound Data Traffic					
D200, 02	Inbound	Data	SB, SP, Net	Layer 2 frame	For Layer 3 direct (Currently not supported)
D200, 04	Inbound	Data	SB, SP	Layer 2 frame	For Layer 2 frames
D200, 06	Inbound	IPC	SB, SP	IPC frame	For IPC
D200, 08	Inbound	Data	SB, SP, Bridge ID, Vlan ID	Layer 2 frame	For multicast. Currently not supported
Outbound Data Traffic					
D200, 01	Outbound	Data		Layer 2 frame	For regular (Layer 2) outbound data. Lookup is performed on ingress side.
D200, 03	Outbound	Data	IP DA, IC Field	Layer 3 frame	For Layer 3 outbound data. Layer 3 lookup is performed on the ingress side. Used for sndData_prot
D200, 05	Outbound	Data / IPC	TB, TP, LID, FHF, FHE, FCInfo	Raw frame	For unicast frames (to a single port) that do not require ingress side processing. Used for: sndData_prot_itf, sndData_raw_itf, sandIPC_unicast
D200, 07	Outbound	Data / IPC	MID Stake FHF, FHE, FCInfo	Raw frame	For multicast frames that do not require ingress side processing. Used for: sndIPC_multicast, sndData_prot_itf_multicast

Tabelle 3: Übersicht der Encapsulation Headers

Um die Datenpakete zu empfangen, registriert sich der Proxy Device Driver mit `dev_add_pack()` als Schicht-3-Protokoll beim Linux Kernel für Datenpakete mit der Ethernet Protokoll ID's `0xC100` und `0xD200`. Jedem Datenpaket, das vom oder zum Netzwerkprozessor gesendet wird, ist ein 32 Bytes langer Header vorangestellt. Dieser erweiterte Ethernet Header

Field name	Value	Length	Description
DA	0xFFFF FFFF FFFF	6B	Destination MAC address (of the CP) if connected by Ethernet.
SA	0x0000 0000 0000	6B	Source MAC address (of the PowerNP Network Processor) if connected by Ethernet.
Ethertype	0xD200	2B	The Ethertype for D2 encapsulated frames
Subtype	0x04	1B	The subtype of the D2 encapsulation.
Reserved		1B	
Frame header		10B	This is the frame header as defined in the Frame Header section. If the frame is redirected, on the ingress side, this is the UJ1 format. If the frame is redirected on the egress side, it can be U1, U2, U3, or M1. The UC/MC and FHF fields of the frame header are used to determine the format of the frame header.
Header type		3b	As defined in the UJ1 frame format
Component		5b	As defined in the UJ1 frame format
Reason code		8b	As defined in the UJ1 frame format
Reserved		2B	Padding to make the size a multiple of 4.
Frame size		2B	The length of the original (whole) frame (excluding D2 header). Valid only when fragmented=1.
Frame ID		2B	This is a counter value that is unique for the given Thread ID. Valid only when fragmented=1.
Fragmented		1b	1= frame is fragmented, 0=frame is a whole frame (not fragmented)
More fragments		1b	1= there are more fragments to the current frame. Valid only when fragmented=1.
Thread ID		6b	Identifies which GDH thread sent the frame. Used for fragment reassembly. Must always be valid.
Fragment sequence		8b	A sequential identifier of the fragment within the frame. Should be 0 for the first fragment of the frame, 1 for the second and so on. Valid only when fragmented=1.
Data		Variable	The Layer 2 frame

Tabelle 4: Encapsulation Header (Ethertype 0xD200, 04)

hat den Ethertype `0xC100` für Guided Traffic und `0xD200` für Data Traffic. Der neu hinzugekommene PromethOS Traffic wird ebenfalls mit einem Header vom Ethertype `0xD200` gesendet und empfangen. Eine Übersicht dieser Header ist in Tabelle 3 dargestellt. Detailliertere Informationen zu diesen Headers sind aus [6] und [7] zu entnehmen.
Den Datenpaketen, die vom Classifier an den Plugin Manager gesendet werden, wird ein Header mit Ethertype `0xD200` mit dem Subtypen `0x04` vorangestellt. Der Aufbau dieses Headers ist in Tabelle 4 dargestellt. Dieser Header wird verwendet, um Unicast Datenpaket in Layer 2 Format an den Control Point zu senden. Das heisst, dieser Typ wird ebenfalls verwendet, wenn ein Datenpaket eintrifft, das für eines der lokalen Linux Interfaces bestimmt ist. Diese Datenpakete sind für eine Applikation im Userspace bestimmt und somit an dem IP Stack weiterzuleitet. Als Entscheidungsgrundlage, welche `0xD200` Datenpakete mit Subtyp `0x04` Data Traffic und welche PromethOS Traffic sind, dient das Feld Component (siehe Tabelle 5). Weil der Classifier nur Datenpakete an den Control Point weiterleitet, die für ein PromethOS Plugin bestimmt sind, bedeutet der Wert 5 in diesem Feld, dass das Datenpaket für den Plugin Manager bestimmt ist.

Value	Picocode component performing the redirection
1	PPP ingress
2	Ethernet ingress
3	IPV4 ingress
4	MPLS ingress
5	Layer 4 (Classifier)
6	Layer 2 Bridging ingress
19	IPV4 egress
20	MPLS egress

Tabelle 5: Werte und Bedeutung des Feldes Component

Wird von Linux Kernel eine Datenpaket mit Typ `0xC100` empfangen, ruft der Kernel die Funktion `reth_ctrl_rcv()` des Proxy Device Drivers auf, trifft ein Datenpaket mit Typ `0xD200` ein, wird die Funktion `reth_data_rcv()` aufgerufen. Beide Funktionen mussten zuvor beim Linux Kernel mit `dev_add_pack()` registriert werden. Um den PromethOS Traffic an den Plugin Manager weiterzugeben, muss somit die Funktion `reth_data_rcv()` erweitert werden.

```
prm_rhp = (rh_d204_t2 *)skb->data;

if ((promethos_receive!=NULL) &&
    (prm_rhp->rh_htype == RHD204_COMPONENT_LAYER4))
{
  ...
  /*--- upcall Plugin Manager ---*/
  promethos_receive(pluginId, skb);
  ...
  return(0);
}
```

Falls der Plugin Manager eine Empfangsfunktion beim Proxy Device Driver registriert (`promethos_receive!=NULL`) hat und das Datenpaket vom Classifier stammt, wird der Socket Buffer durch Aufruf der Empfangsfunktion an den Plugin Manager übergeben.

4.2.4 Pakete senden

Für das Senden vom Plugin Manager aus, kann nur beschränkt auf den Funktionen des Proxy Device Drivers aufgebaut werden. Die bestehenden Funktionen gehen davon aus, dass das IP Routing bereits im IP Stack stattgefunden hat. Dadurch ist auch definiert, auf welchem Port des Netzwerkprozessors das Datenpaket gesendet werden muss. Für das Übertragen zum Netzwerkprozessor wird somit der Encapsulation Type `0xD200, 0x05` verwendet. Da der Plugin Manager aber keine IP Routing Funktionalität hat und diese Aufgabe vom Netzwerkprozessor übernommen werden soll, können die bestehenden Sendefunktionen nicht verwendet werden.
Für das Senden vom Plugin Manager aus wurden die zwei Funktionen `promethos_send()` und `promethos_send_raw()` im Proxy Device Drivers implementiert. Wird `promethos_send()` verwendet, wird das Datenpaket mit einem Encapsulation Type `0xD200, 0x03` gesendet. Somit wird nur ein Layer 3 Datenpaket dem Netzwerkprozessor übergeben. Dieser führt auf der Ingress Seite das notwendig IP Routing durch und bildet anschliessend den noch fehlenden Layer 2 Header. Im Encapsulation Header muss das Layer 4 Skip Flag gesetzt sein. Durch das Setzen dieses Flags wird der Classifier übersprungen und verhindert, dass ein vom Plugin Manager gesendetes Datenpaket erneut zum Plugin Manager weitergeleitet wird. Würde das Flag nicht gesetzt, bestünde die Möglichkeit, dass ein Datenpaket zwischen Netzwerkprozessor und Plugin Manager endlos hin und her gesendet würde.
In Spezialfällen kann es nützlich sein, die Routing-Funktionalität im Netzwerkprozessor zu umgehen. Dazu kann `promethos_send_raw()` verwendet werden. Bei dieser Funktion wird das Datenpaket mit einem Encapsulation Type `0xD200, 0x05` gesendet. Ein Setzen des Layer 4 Skip Flag ist nicht notwendig (und auch nicht möglich), da diese Datenpakete nur auf der Egress Seite bearbeitet werden und somit der Classifier nicht zum Einsatz kommt.
Für diesen Sendevorgang zum Netzwerkprozessor ist es entscheidend, ob der Plugin Manager auf dem embedded PowerPC oder einer externen CPU abläuft. Beim Einsatz einer externen CPU, die über eine Ethernet Verbindung mit dem Netzwerkprozessor kommuniziert, wird der Socket Buffer mit `hard_start_xmit()` dem BroadCom 5700 Netzwerkkartentreiber übergeben. Wird der Plugin Manager auf dem embedded PowerPC ausgeführt, muss der Socket Buffer in den D6 DRAM Speicher kopiert werden. Dafür kann die Funktion `reth_pciXmit()` verwendet werden.

4.3 NP Control Daemon

Für die Kontrolle und Konfiguration des Classifiers wurde der NP Control Daemon (NP CtrlD) und der NP Control Client (NPCtrl) implementiert, der auf dem Control Point des Netzwerkpro-

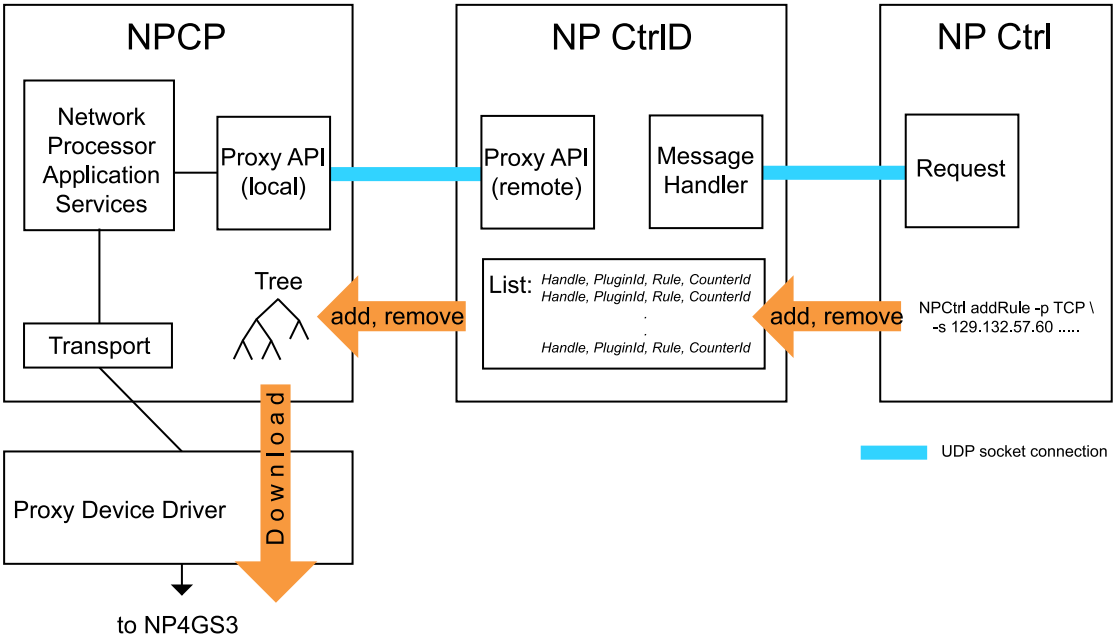


Abbildung 6: NP Control Daemon Architektur

zessors (NPCP) aufbaut und die Proxy API Schnittstelle verwendet. Die Anforderungen des NP Control Daemon lassen sich wie folgt umschreiben:

- Stellt eine einfache Schnittstelle zur Konfiguration des Classifiers mittels Plugin ID und zugehöriger Regel
- Abstrahiert die komplexen Funktionsaufrufe für die Anmeldung beim Network Processor Application Services (NPAS)
- Konfiguriert Zähler im Netzwerkprozessor, damit ausgewertet werden kann, wieviele Datenpakete auf eine Regel zugetroffen haben

In Abbildung 6 ist die Architektur des NP Control Daemon aufgezeigt. Der Daemon verwaltet alle konfigurierten Regeln in einer Liste. Diese Liste enthält die Informationen, welches Plugin welche Regel angemeldet hat und welcher Paketzähler zur Regel gehört. Weiter werden in der Liste Informationen zu den Regeln abgespeichert, die der Daemon benötigt, um die Regeln zu verwalten. Erhält der NPCtrD vom NPCtrl die Aufforderung, eine Regel hinzuzufügen oder zu löschen, wird die Liste aktualisiert und die entsprechenden Funktionen im NPCP aufgerufen. Die Regeln werden im NPCP und im Netzwerkprozessor in einer Baumstruktur verwaltet. Das Ändern einer Regel hat zur Folge, dasS die Baumstruktur angepasst und anschliessend der komplette Baum in den Netzwerkprozessor geladen wird. Ein Änderung in den Regeln hat immer einen Download zur Folge. Dieser Download wird vom NPCtrlD nach jedem Hinzufügen oder Löschen einer Regel ausgelöst.
Der NP Control Daemon bietet auch Platz für zukünftige Erweiterungen. Der Daemon ist somit für die PromethOS spezifische Steuerung des Netzwerkprozessors verantwortlich.

4.3.1 Verwendung des NPCP Proxy APIs

Um den bestehenden Control Point zu erweitern, besitzt der npcp eine Proxy API Schnittstelle [3]. Damit auch in späteren Versionen des Control Points die Erweiterungen genutzt werden können, ist die Verwendung dieser Schnittstelle notwendig. Der NP Control Daemon kann, sobald der NP Control Point initialisiert ist, gestartet werden. Danach läuft der NP Control Daemon im Hintergrund und terminiert erst, wenn der NP Control Point terminiert. Bei der Implementation musste festgestellt werden, dass die Proxy API Schnittstelle des Advanced Software Offering 1.3.4 eine «Copy-Past»-Fehler

hat. In der Datei `./src/wrapper/linux/proxy_remote_api.c` muss in der Funktion `np_cls_addStaticRule_tcp()`, die für das Hinzufügen einer TCP-Protokoll Regel verwendet wird, folgende Zeile ersetzt werden.

```
PROXY_MESSAGE_INIT("np_cls_addStaticRule_udp");
```

ersetzen durch:

```
PROXY_MESSAGE_INIT("np_cls_addStaticRule_tcp");
```

In der NPCP Proxy API Schnittstelle sind bereits sehr viele Funktionen des NP Control Points verfügbar. Es kann aber sein, dass eine Funktion, die man verwenden will, noch nicht implementiert ist. In [3] ist im Kapitel «Adding New APIs to the Proxy API Component» beschrieben, wie zusätzliche Funktionen zur Schnittstelle hinzugefügt werden können.

4.3.2 NPCtrlD Kommunikationsprotokoll

Um den Classifier des Netzwerkprozessors zu konfigurieren, wurde eine einfache Schnittstelle implementiert, die auf dem UDP Protokoll basiert. Für die Kommunikation zwischen dem NP Control Daemon und einem Client werden NP Control Messages ausgetauscht. Das Format dieser Messages ist in `./src/wrapper/linux/NPCtrlD/NPCtrlMsg.h` definiert. Eine Message besteht jeweils aus einem Struct vom Typ `np_daemon_message_s`.

```
struct np_daemon_message_s
{
   char type;
   union{
     struct add_rule_s add_rule;
     struct rcv_list_s rcv_list;
   } content;
};
```

Das genaue Format der einzelnen Messages, d. h. auf welche Art der Union «content» verwendet wird, hängt vom Typ der NP Control Message ab. Die in der Version v1.0 des NP Control Daemons verwendeten Typen von NP Control Messages sind in der Tabelle 6 aufgelistet.

Message Type	NPCtrlD	NPCtrl	Inhalt	Funktion
MSG_CMD_SUCCESS	Sender	Empfänger		Bestätigt eine erfolgreiche Ausführung
MSG_CMD_ERROR	Sender	Empfänger		Ausführung ist fehlgeschlagen
MSG_CMD_ADD_RULE	Empfänger	Sender	Regel	Fügt eine Regel für den Classifier hinzu. Das Feld `add_rule` wird für das Übermitteln der Regel verwendet.
MSG_CMD_REQ_LIST	Empfänger	Sender		Fordert eine Liste aller im Classifier verwendeten Regeln an. Gleichzeit werden auch die Counter der entsprechenden Regeln ausgelesen. Als Antwort auf diese Anfrage wir pro Regel eine Message vom Typ MSG_CMD_RCV_LIST gesendet.
MSG_CMD_RCV_LIST	Sender	Empfänger	Regel, Counter	Wird als Antwort auf MSG_CMD_REQ_LIST gesendet. Für die Übermittlung von Regel und Counter wird das Feld `rcv_list` verwendet.
MSG_CMD_EMPTY_LIST	Sender	Empfänger		Wird als Antwort auf MSG_CMD_REQ_LIST verwendet sofern keine Regel vorhanden ist.
MSG_CMD_FLUSH	Empfänger	Sender		Löscht alle Regeln im Classifier

Tabelle 6: NP Control Daemon Messages

Der NP Control Daemon empfängt alle UDP Datenpakete auf den in `#define MSG_DEFAULT_PORT` definierten UDP-Port. Es findet keine Anmeldung oder Authentifizierung statt. Es ist auch nicht vorgesehen, dass mehrere Applikationen auf diese Schnittstelle zugreifen, sondern man geht von einem NP Control Client aus.

4.3.3 NP Control Client

Der NP Control Client ist in der aktuellen Version als Userspace Applikation «NPCtrl» realisiert. Mit dem Aufruf von NPCtrl mit den entsprechenden Optionen können die Regeln des Classifiers konfiguriert oder angezeigt werden. In einer späteren Version könnte NP Control ersetzt werden, und z. B. der Plugin Manager oder ein Plugin Loader könnte das An- und Abmelden von Regeln übernehmen.
NP Control sendet die in Tabelle 6 NP Control Daemon Messages an den Daemon, wartet auf ein Antwort und terminiert wieder. Ein Aufruf von NP Control hat immer die Struktur von `NPCtrl <Funktion> [Optionen]`. Als Funktion sind in Version 1.0 `addRule`, `flush` und `list` implementiert. Mit `addRule` wird eine einzelne Regel hinzugefügt. Durch `flush` werden alle aktuell konfigurierten Regeln im Classifier gelöscht. Möchte man sich einen Überblick verschaffen, welche Regeln konfiguriert sind und wieviele Datenpakete bisher auf diese Regeln zugetroffen haben, so kann man mit `list` die Regelliste mit Zählerwerten anzeigen lassen.

4.4 Embedded PowerPC

Der Netzwerkprozessor NP4GS3 hat im EPC eine embedded PowerPC Subsystem mit einem 133 MHz PPC405D4V4 Prozessor Core. Ziel ist es, auf dem embedded PowerPC PromethOS zu installieren, d. h. eine Linux Installation mit dem im Abschnitt 4.2 beschriebenen Proxy Device Driver zu installieren. Ist dies gelungen, soll mittels des Plugin Manager Simulators der maximale Durchsatz für ein PromethOS Plugin, das auf dem embedded PowerPC ausgeführt wird, gemessen werden. Dazu waren folgende Punkte notwendig, auf die ich den folgenden Abschnitten detaillierter eingehen will.

- Verwendung von d6load abklären
- Erstellen eines Linux Kernel für den embedded PowerPC 405
- Erstellen eines Dateisystems für den embedded PowerPC 405
- Anpassen des Bootpicocode
- Anpassen des Proxy Device Drivers

4.4.1 Upload des Kernelimages

Der embedded PowerPC hat ausser dem Speicher auf dem Application Reference Board keine weiteren Speichermedien. Der Linux Kernel kann somit nicht wie üblich von einer Harddisk oder Diskette gebootet werden. Der embedded PowerPC kann aber anstatt von einer Harddisk vom externen D6 DRAM Speicher oder mittels dem PCI Bus gebootet werden. Dabei ist zu beachten, dass der PCI Bus des embedded PowerPCs nicht identisch ist mit dem PCI Bus, durch den das Application Reference Board am Hostrechner angeschlossen ist, sondern es ist ein eigener, unabhängiger PCI Bus. Dies ist ein Unterschied zur Reference Plattform, bei der das K2 Control Board mit dem PCI Bus des embedded PowerPCs verbunden ist. Im Advanced Software Offering gibt es ein Tool (d6load), um ein Kernelimage in den D6 DRAM Speicher zu laden. Von diesem d6load gibt es verschiedene Varianten. In der Variante für die Reference Plattform schreibt das d6load über den PCI Bus des embedded PowerPC in den D6 DRAM Speicher. Bei der Variante für das Application Reference Board ist dies nicht möglich. Das Kernelimage wird als Guided Traffic zum Netzwerkprozessor gesendet, wo ein Thread den Inhalt der eintreffenden Pakete in den D6 DRAM Speicher schreibt. Die beiden Varianten von d6load befinden sich im ASO unter:

- ./src/linux/em405/k2/utils/d6load.c für die Reference Plattform
- ./src/linux/em405/sonic/utils/d6load.c für das Application Reference Board

Damit der embedded PowerPC nun vom D6 DRAM Speicher und nicht via PCI Bus bootet, muss auf dem Application Reference Board der Configuration Jumper «Boot embedded PowerPC from D6» gesetzt sein. Wird nun der embedded PowerPC aus dem Reset Status geholt, bootet er das im D6 DRAM Speicher befindende Kernelimage.

4.4.2 Der Linuxkernel für den embedded PowerPC 405

Um das Kernelimage für den embedded PowerPC zu erstellen, wurde MontaVista Linux Professional Edition verwendet. Dies beinhaltet einerseits einen Cross-Compiler für den PowerPC 405 und andererseits einen angepassten Sourcetree für den embedded PowerPC405 des NP4GS3. Der Sourcetree besteht aus einem angepassten 2.4.17 Linux Kernel. Leider musste festgestellt werden, dass dieser nur auf der Reference Plattform funktionierte und nicht auf dem Application Reference Board. Der Verdacht kam auf, dass die unterschiedliche Ansteuerung der seriellen Schnittstelle auf den beiden Board Varianten die Ursache ist. Der Linux Kernel konnte somit keine Console auf der RS-232 Schnittstelle öffnen.
Durch die grosse Unterstützung von Roman Pletka vom IBM Zürich Research Laboratory ist es gelungen, den Kernel so anzupassen, dass er auch auf dem Application Reference Board funktioniert.
Damit der Bootloader in der Lage ist, Meldungen auf die Console auszugeben, musste die Datei `./arch/ppc/boot/common/ns16550.c` angepasst werden. Für die Ansteuerung der seriellen Schnittstelle nach dem Bootvorgang mussten mehrere Dateien geändert werden. Neben dem Gerätetreiber an sich musste auch der Kernel angepasst werden, damit der UART Exar ST16C2550 des Application Reference Boards über den Control Access Bus (CAB) angesteuert werden kann. Angepasst wurden die Dateien:

- ./arch/ppc/kernel/misc.S
- ./arch/ppc/platforms/rainier.c
- ./arch/ppc/config.in
- ./drivers/char/serial.c
- ./arch/ppc/kernel/head_4xx.S

Alle Änderungen werden mit dem Einspielen der beiden Patch Dateien `bootloader_serial_sonic.patch` und `serial_sonic_tlb_pinned.patch` eingebracht. Beim Konfigurieren des Kernels ist darauf zu achten, dass unter «Platform support» die Option «Serial console support for ARB/Sonic boards (experimental)» (CONFIG_SONIC_SERIAL) und unter «General Setup» die Option «Pinned Kernel TLBs (experimental)» gesetzt sind. Die gesamte Datei `.config`, die für das Kompilieren des Linux Kernels verwendet wurde, ist in Anhang C.4 abgedruckt.

4.4.3 Root Filesystem

Auf der Application Platform wird das Root Filesystem durch eine NFS Verbindung geholt. Dafür gibt es im Kernel einen NPnet Treiber, der während des Bootvorganges eine Netzwerkverbindung zwischen dem embedded PowerPC und dem K2 Control Board über den PCI Bus des embedded PowerPC aufbaut. Dies ist beim Application Reference Board nicht möglich, da die PCI Bus Verbindung fehlt. Deshalb wurde dem Kernelimage eine initial RAM Disk angefügt. Diese initial RAM Disk muss alle Dateien enthalten, die für das Starten von Linux notwendig sind. Um diese RAM Disk möglichst klein zu halten, wurde Busybox [15] verwendet. Neben den grundlegenden Dateien für das Betriebsystem wurden auch alle Dateien auf die RAM Disk kopiert, die für das Starten und Testen des Application Reference Boards notwendig sind. Dazu gehören folgende Dateien:

- /usr/bin/pci_rethdd.o: Proxy Device Driver
- /usr/bin/npcp: Control Point
- /etc/npcp/npref.elf: Picocode
- /etc/npcp.d/ifcfg: Interface Configuration
- /etc/npcp.d/npcp: Control Point Configuration
- /usr/bin/NPCtrlD: NP Control Daemon
- /usr/bin/NPCtrl: NP Control
- /usr/bin/pluginManagerSim.o: Plugin Manager Simulator

4.4.4 Bootpicocode

Als der Linux Kernel auf dem embedded PowerPC seinen Dienst tat, musste leider festgestellt werden, dass die Kommunikation des Control Point mit den Pico Engines nicht funktioniert. Anhand der Debug Informationen konnte nachvollzogen werden, dass der Control Point zwar in den Buffer im D6 DRAM Speicher seine Datenpakete schreiben konnte, er aber keine Antwort bekam. Dies machte er solange, bis der Buffer voll war und keine Datenpakete mehr Platz hatten.
Für die Kommunikation zwischen embedded PowerPC und den Pico Engines ist auf der Pico Engine Seite der General PowerPC Handler (GPH) Thread verantwortlich. Dieser Thread wird vom Bootpicocode konfiguriert und gestartet. Der Sourcecode für den Bootpicocode wird mit dem Advanced Software Offering mitgeliefert. Der Original-Bootpicocode von S3 wurde mit einer für das Sonic Board abgeänderten ASO-1.2.2 erstellt. Dieser Sourcecode stand uns von IBM zur Verfügung. Dabei ist wichtig zu wissen, dass der Bootcode für das Application Reference Board mit `make npref` kompiliert wird. Dadurch wird unter anderem beim Kompilieren das Define `CUSTBOOTCNFG=1` gesetzt. Das hat wiederum zur Folge, dass anstelle von `./inc/ctl/np_boot_config.inc` die Datei `./inc/ctl/custboot/np_boot_config.inc` eingebunden wird. In dieser Datei wird definiert, ob der GPH gestartet werden soll oder nicht.

```
; *************************
; GPH Configuration Values
; *************************
;
GPH_Flags     EQU 0b00000011  ; Bit 0 => No GPH Init Flag
                              ;          1 = Do not init GPH
                              ;          0 = Initialize GPH
                              ; Bit 1 => No GPH Picocode Load Flag
                              ;          1 = Do not load GPH code
                              ;          0 = Load from source
                              ; Bit 2 => PowerPC Reset Disable Flag
                              ;          1 = Bring PowerPC out of reset
                              ;          0 = Hold PowerPC in reset
GPH_Load_Info EQU 0x04006001  ; Lobyte = Byte offset in flash
                              ; Hibyte = No. of picocode instructions
                              ;          to load, in words (4-bytes)
```

Das `GPH_Flags` wurde auf `0b00000000` gesetzt und der Bootpicocode neu kompiliert. Um den Bootpicocode zu testen, wurde der Code mit dem NPScope über den RiscWatch Debugger in den Speicher des NP4GS3 geladen. Damit dies funktioniert, muss auf dem Application Reference Board der Konfigurations Jumper «Boot picocode from alternate source» gesetzt sein. Ist dieser Jumper gesetzt, wird kein Bootpicocode aus dem Flash geladen und kein Thread gestartet, wenn der NP4GS3 aus dem Reset Status geholt wird. Nun kann der Bootpicocode mit NPScope geladen werden. Der Code aus der Datei `npref.elf` wird in den Instruktionsspeicher geschrieben und anschliessend ein Thread gestartet. Das Board bootete korrekt und ein anschliessender Test hat ergeben, dass der Control Point nun korrekt auf dem embedded PowerPC abläuft.
Um den Bootpicocode ins Flash zu brennen, muss die Datei `npref.elf` in das Standard Intel MCS-86 Format `.hex` oder `.mcs` konvertiert werden. Dafür wurde die Konvertierungssoftware NPFlash verwendet, die beim IBM PowerNP Software Developer's Toolkit dabei ist. Die entstandene `npref.mcs` Datei wurde mit der Flash Programming Application ins Flash gebrannt. Mit den Kommandos

```
# ./fpa -target 0 -image ./npref.mcs
# ./fpa -target 1 -image ./npref.mcs
```

werden beide Sektoren des Flashs gebrannt, und mit dem Kommando

```
# ./fpa -boot 0
```

wird festgelegt, dass der NP4GS3 vom ersten Sektor booten soll.
Erstaunlicherweise konnte das Application Reference Board nicht korrekt booten. Derselbe Bootpicocode, der bei einem Download mit NPScope funktioniert, versagt, wenn er aus dem Flash geladen wird. Des weiteren konnte beobachtet werden, dass der Bootpicocode aus dem Flash funktioniert, wenn zuerst derselbe Bootpicocode mit NPScope geladen wird und anschliessend durch ein Softreset des Application Reference Boards der Bootpicocode aus dem Flash geladen wird. Nach einer Unterbrechung der Stromzufuhr des Application Reference Boards kann das Board aber nur mit dem Bootpicocode aus dem Flash nicht korrekt booten.
Bei der Fehlereingrenzung kamen wir auf zwei Schlüsse:

- Entweder führt NPScope auf dem Application Reference Board Code aus, der im Bootpicocode nicht enthalten ist. Diese Fehlerquelle wird aber als sehr unwahrscheinlich beurteilt.
- Oder es liegt ein Fehler vor bei der Konvertierung von der Datei `npref.elf` in die Datei `npref.mcs`. Für diese Möglichkeit spricht die Unsicherheit, ob die Werte in der Definitionsdatei von NPFlash korrekt gesetzt sind. In dieser Datei wird definiert, welche Codesegmente an welche Adresse in der Datei konvertiert werden `npref.mcs`. Dagegen spricht aber, dass ein falsch konvertierter Bootpicocode korrekt bootet, nachdem das Application Reference Board einmal korrekt mittels NPScope Codedownload gebootet hat.

Da in keiner Dokumentation Informationen über die genaue Funktion der Definitionsdatei von NPFlash vorzufinden war und somit auch nicht verifiziert werden konnte, ob die darin gesetzten Werte korrekt sind, wurde ein eher unkonventioneller Weg eingeschlagen.
Auf der CD-ROM, die zusammen mit dem Application Reference Board ausgeliefert wird, ist der Original Bootpicocode in der Datei `copernicus_flash.hex` gespeichert. Nachdem dieser Bootpicocode ins Flash gebrannt wurde, bootete das Board korrekt ohne Hilfe durch NPScope, natürlich mit dem Schönheitsfehler, dass der General PowerPC Handler Thread nicht lief. Des weiteren wurde analysiert, wo im Assemblercode überall das Define `GPH_Flags` verwendet wird. Es konnte festgestellt werden, dass es nur an einer Stelle vorkommt. Der Wert von `GPH_Flags` wird einmal ins Register r8 geladen, ansonsten wird er nicht verwendet. Mit der Datei `./obj/i386-redhat-linux/npdd/pico/link/npref/npref.lst` konnte genau nachvollzogen werden, welcher Maschinencode an welcher Adresse das Laden des Registers r8 mit `GPH_Flags` zu Folge hat.

```
                      =00000063    Get_GPH_Flags           EQU $
.. 00000063 B1740000 =00000000             ldr     r8, #GPH_Flags
.. 00000064 18700000                       ret
```

In der Datei `copernicus_flash.hex` wurde an der Stelle 0x63 nach dem Maschinencode `B1740000` gesucht und man wurde fündig. Dabei ist zu beachten, dass in der Datei `copernicus_flash.hex` nicht nach Instruktionen adressiert wird, sondern nach Bytes, man muss somit nicht an der Stelle 0x63 suchen, sondern an der Stelle 0x018C (1 Opcode = 4 Bytes). Dabei ist das Format der Hex Datei wie folgt zu lesen:

```
:100171 00 18700000 B1750020 18700000 B1740001 02
 ^^^^^^    ^^^^^^^^ ^^^^^^^^ ^^^^^^^^ ^^^^^^^^ ^^
 Address   Opcode_1 Opcode_2 Opcode_3 Opcode_4 Checksum
```

In der Original Datei `copernicus_flash.hex`, von der hier ein Auszug abgedruckt ist,

```
:1001710018700000B175002018700000B174000102
:1001810018700000B175000018700000B174000310
:1001910018700000B1750400B0756001187000009E
```

wurde der Opcode `B1740003` durch `B1740000` ersetzt und die Checksumme neu berechnet. Um die Checksumme zu berechnen, werden alle Bytes einer Zeile addiert (inkl. Checksumme). Das Resultat modulo 0xFF muss null ergeben.

```
:1001710018700000B175002018700000B174000102
:1001810018700000B175000018700000B174000013
:1001910018700000B1750400B0756001187000009E
```

Die abgeänderte Hex Datei wurde in den Flash Speicher gebrannt und ein anschliessender Test ergab, dass das Application Reference Board korrekt bootet und der General PowerPC Handler Thread startet. Damit sind die Voraussetzungen für das Funktionieren des Proxy Device Drivers und somit des Plugin Managers gegeben.

4.4.5 Anpassungen

An der Schnittstelle des Plugin Managers ändert sich nichts, die Prozessor Architektur und die damit unterschiedlichen Kommunikationsarten werden durch den Proxy Device Driver abstrahiert. Im Proxy Device Driver mussten die Funktionen `promethos_send()` und `promethos_send_raw()` angepasst werden. Diese Funktionen für das Senden von Datenpaketen vom Plugin Manager zum Netzwerkprozessor wurden neu geschrieben und sind noch nicht für den embedded PPC ausgelegt. Für das Empfangen von Datenpaketen wurde die bestehende Funktion des Proxy Device Drivers erweitert; eine Anpassung dieser Funktion ist nicht notwendig. Bei den sendenden Funktionen muss unterschieden werden, ob die Datenpakete via dem Buffer im D6 DRAM Speicher oder via der PCI-Ethernet Bridge an den Netzwerkprozessor übermittelt werden. Bei der embedded PowerPC Architektur wird via D6 DRAM Speicher kommuniziert und für das Schreiben der Datenpakete in den D6 DRAM Speicher die Funktion `reth_pciXmit()` verwendet. Der Funktionsname ist etwas verwirrend, denn der PCI Bus wird für das Senden der Datenpakete gar nicht verwendet. Der Name kommt daher, da dieselbe Funktion verwendet wird bei einer PCI-Attached Internal Control Point Configuration. Die Datenpakete werden da via dem PCI Bus des embedded PPC in den D6 DRAM Speicher geschrieben. Als Parameter werden Verweise auf den Socket Buffer und auf das sendende Linux Netzwerk Device übergeben. Da die Datenpakete aber nicht vom Linux Userspace und somit über eine Linux Netzwerk Device gesendet wurden, sondern von einem PromethOS Plugin stammen, kann man keine Linux Netzwerk Device zuordnen. Da dieser Pointer auf das Linux Netzwerk Device nur für die Ausgabe von Debugging Meldungen verwendet wird, wird ein Pointer auf einen Struct `net_device` mit dem Namen `dummy_dev` übergeben. Dieses Dummy Device hat keinen weiteren Einfluss.

```
#ifdef PPC405
  skb->dev = dummy_dev;
  reth_pciXmit(skb, &dummy_dev);
#else
  skb->dev = prm_pdev;
  prm_pdev->hard_start_xmit(skb, prm_pdev);
#endif
```

Bei einer Ethernet-Attached External Control Point Configuration werden die Datenpakete via der BroadCom Netzwerkkarte zum definierten DMU des Netzwerkprozessors gesendet. Der Pointer auf dieses Linux Netzwerk Interface wird während der Initialisierung des Proxy Device Drivers in `prm_pdev` gespeichert. Mit `prm_pdev->hard_start_xmit()` werden die Datenpakete dem Network Device Driver `copernicus.o` zum Senden übergeben.

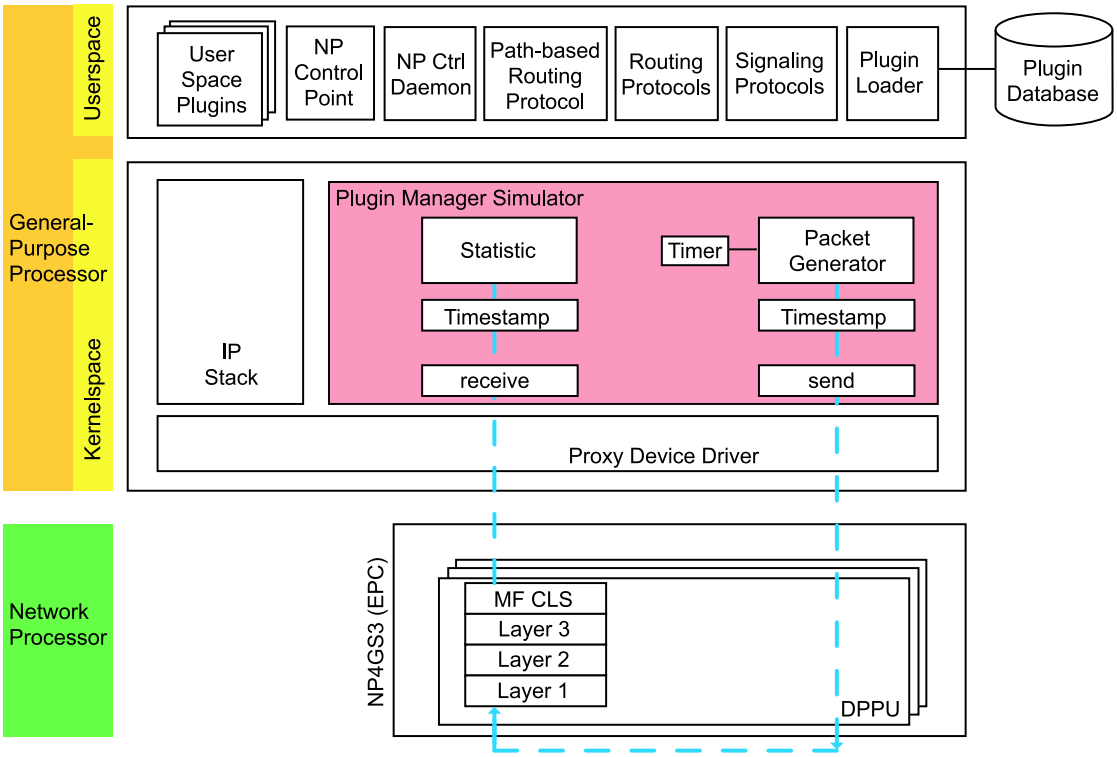


Abbildung 7: Plugin Manager Simulator als Traffic Generator

5 Evaluation

5.1 Plugin Manager Simulator

Für die Evaluation der Implementation wurde der Plugin Manager Simulator entwickelt. Es ist ein Linux Kernel-Modul, das den eigentlichen Plugin Manager ersetzt. Es benützt dieselbe Schnittstelle zum Proxy Device Driver wie der Plugin Manager. Der Simulator arbeitet aber nicht mit PromethOS Plugins zusammen, sondern der Simulator erzeugt oder bearbeitet die Datenpakete selbst. Seine Funktion beschränkt sich auf die Messung der Verbindung zwischen Plugin Manager und dem NP4GS3 beziehungsweise zwischen Plugin Manager und einem oder mehreren Traffic Generators im Netz.

5.2 Simulationsaufbau

5.2.1 Plugin Manager Simulator als Traffic Generator

Die Messungen können auf zwei verschiedene Arten durchgeführt werden. In der ersten Variante (in Abbildung 7 dargestellt) erzeugt der Plugin Manager Simulator die Datenpakete und sendet diese inklusive Ethernet Header auf einem fest vorgegebenen Port des NP4GS3. Dieser sendende Port ist mit einem gekreuzten Ethernetkabel mit einem weiteren Port des NP4GS3 verbunden. Die Datenpakete werden, nachdem sie beim zweiten Port eingetroffen sind, vom Picocode prozessiert und vom Classifier klassifiziert. Der Classifier wurde mit einer Regel so konfiguriert, damit alle diese Pakete auf diese Regel zutreffen. Somit werden alle Datenpakete an den Control Point weitergeleitet und gelangen schlussendlich wieder zum Plugin Manager Simulator zurück.

5.2.2 Mit externen Traffic Generators

In der zweiten Variante (in Abbildung 8 dargestellt) werden UDP Pakete von einem Traffic Generator erzeugt, wobei dafür eine Applikation im Userspace auf einem weiteren Linuxrechner verwendet wird. Da diese Computer am Hochschulnetzwerk in der Regel mit maximal 100 MBit/s

angeschlossen sind, mussten mehrere solche Traffic Generators eingesetzt werden, um die Leistungsfähigkeit des NP4GS3 ganz auszutesten. Die abgehenden UDP Datenpakete sind an einen NP4GS3 Port des PromethOS Hosts adressiert. Die eintreffenden Datenpakete werden wie beim ersten Testaufbau vom NP4GS3 prozessiert und klassifiziert. Dabei ist der Classifier wieder so konfiguriert, dass alle die UDP Pakete auf eine Regel zutreffen und somit an den Control Point und zum Plugin Manager Simulator gelangen. Bei den vom Plugin Manager Simulator empfangenen Datenpaketen wird die IP Ziel- und Quelladresse vertauscht, die Checksumme neu berechnet und sofort wieder gesendet.

```
int echo_receive(unsigned short pluginId, struct sk_buff *skb)
{
   unsigned int tmp;
   struct udphdr *uh;

   uh = (struct udphdr *)((u_int32_t *)skb->nh.iph + skb->nh.iph->ihl);

   tmp = skb->nh.iph->saddr;
   skb->nh.iph->saddr = skb->nh.iph->daddr;
   skb->nh.iph->daddr = tmp;
   uh->check = 0;

   update_checksum(skb);
   promethos_send(skb);

   return(0);
}
```

Das Konzept des Messverfahrens hat den Vorteil, dass es ebenfalls möglich wäre, diese Funktionalität in einem PromethOS Plugin zu implementieren. Damit könnte die Leistungsfähigkeit mit Einbezug des Plugin Managers gemessen werden. Da uns zur Zeit keine Infrastruktur zur Verfügung steht, um mit einem Durchsatz von 1 GBits/sec senden und empfangen zu können, müsste der Traffic Generator auf mehrere Hosts aufgeteilt werden. Eine Messung mit 10 Traffic Generators, die mit je 100 MBits/sec am Netzwerk anschlossen sind, ist kompliziert und birgt viele Fehlerquelle in sich. Das Testverfahren ist zwar im Plugin Manager implementiert, doch für die Messungen würde nur die Variante mit dem Plugin Manager Simulator als Traffic Generator verwendet.

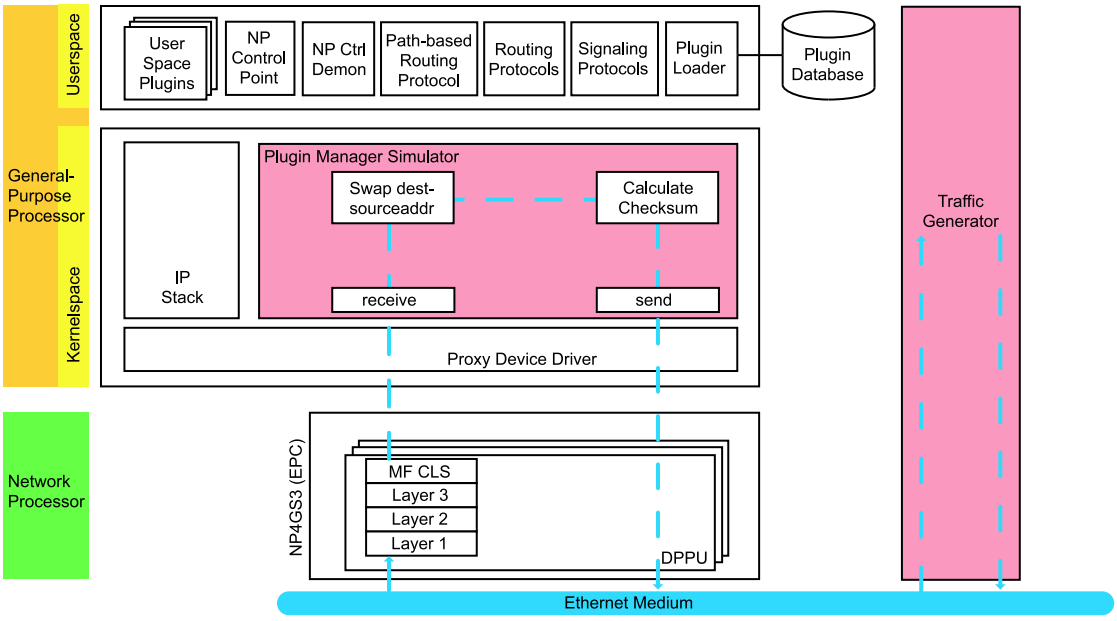


Abbildung 8: Plugin Manager Simulator mit externem Traffic Generator

5.2.3 Messverfahren

5.2.4 Latenzzeiten

Um die Latenzzeit der Datenpakete zu messen, wird jedes Datenpaket vor dem Senden mit einer Zeitmarke versehen. Das Resultat der im Linuxkernel zur Verfügung stehenden Funktion `do_gettimeofday()` wird im Datenpaket direkt nach den Layer 4 Header ins Datenpaket geschrieben. Mit dieser Funktion sind Zeitmessungen mit Mikrosekunden Intervallen möglich. Es wird darauf verzichtet, die Byte Reihenfolge mit `htonl()` richtig zu stellen, da die Pakete in beiden Messvarianten vom selben Host gesendet und empfangen werden und es zwischen dem Zeitmarke setzen und dem Senden möglichst keine zusätzlichen Funktionsaufrufe geben sollte.
Wird ein Datenpaket empfangen, so wird als erstes mit einem weiteren `do_gettimeofday()` die aktuelle Zeit gelesen und die Differenz zwischen dem aktuellen Zeitwert und dem im Datenpaket abgespeicherten Zeitwert errechnet. Für die Messung ist in erster Linie der Durchschnittswert der Latenzzeit interessant. Deshalb werden die einzelnen Latenzzeiten fortlaufend summiert und am Schluss der Messung durch die Anzahl eingetroffener Datenpakete dividiert.

5.2.5 Durchsatz

Für das Errechnen des Durchsatzes wird die Zeit gemessen vom Senden des ersten Datenpaketes bis zum Eintreffen des letzten Datenpaketes. Für die Berechnung des Durchsatzes wird die Anzahl empfangener Datenpakete mit der Datenpaketlänge multipliziert und durch das gemessene Zeitintervall dividiert. Der für das Senden der Datenpakete vom Linuxkernel zum NP4GS3 benötigte 36 Bytes lange D2-Header wird nicht miteingerechnet. Die Datenpaketlänge ist definiert als Anzahl Bytes, die am Gbit-Ethernetport des NP4GS3 pro Datenpaket gesendet werden.

5.2.6 Paketverluste

Für jedes Datenpaket wird im Linuxkernel ein Socket Buffer angelegt. Werden die Datenpakete schneller gesendet als das System verarbeiten kann, so beginnen sich die Socket Buffer im Linuxkernel zu stauen. Ist der für Socket Buffer zur Verfügung stehende Speicher erschöpft, kann für ein zu sendendes Datenpaket kein neuer Socket Buffer erstellt werden und das Senden des Datenpakets scheitert. Bei den Messungen wird nicht versucht, das Datenpaket ein zweites Mal zu senden, sondern dieser Vorgang geht als Paketverlust in die Messung ein und wird zusätzlich in einem Zähler mitgezählt. Bei Überlastung des Systems werden noch an weiteren Stellen im Linuxkernel oder in der Hardware Datenpakete verworfen. Der Plugin Manager Simulator zählt alle eintreffenden Datenpakete und kann somit aus der Differenz mit den gesendeten Datenpaketen das Total der Paketverluste während einer Messung errechnen.

5.2.7 Sendeverhalten

Jede Messung ist definiert durch die Parameter Anzahl zu sendende Datenpakete, Datenpaketlänge und Zeitintervall zwischen zwei Sendevorgängen. Das Sendeverhalten ist durch diese drei Parameter definiert für den Plugin Manager Simulator beziehungsweise für die externen Traffic Generators. Bei jeder Messung wird die durchschnittliche Latenzzeit, der Durchsatz und die Anzahl verlorener Datenpakete gemessen. Für einen Test wird diese Messung nun mit unterschiedlichen Werten für die Parameter Paketlänge und Zeitwert durchgeführt. Der Parameter Anzahl zu sendende Datenpakete ist für einen Test konstant. Er sollte hoch gewählt werden, damit einzelne Messfehler wenig Einfluss auf das Messresultat haben. Die Messungen wurden mit den fünf verschiedenen Datenpaketlängen 72, 350, 700, 1050 und 1460 Bytes durchführt. Es wurden pro Messung jeweils 100’000 Datenpakete gesendet.

5.3 Messwerte Ethernet-Attached External Control Point

Für Tests standen zwei Systeme zur Verfügung. Am Institut für Technische Informatik und Kommunikationsnetze an der ETH Zürich wurden die Messungen auf einem Application Reference Board mit einem NP4GS3 Version B durchgeführt. Das Board ist in einem Dell Precision 340

über einen PCI 32-bit/33MHz angeschlossen. Die Tests wurden auf diesem System sowohl mit der Konfiguration Ethernet-Attached External Control Point als auch mit der Konfiguration Embedded PowerPC Processor Internal Control Point (Messwerte siehe Abschnitt 5.4) durchgeführt. Bei der ersten Konfiguration wird der Plugin Manager Simulator auf dem Intel Pentium 4 des Hostrechners ausgeführt, bei der zweiten Konfiguration auf dem embedded PowerPC 405 des Netzwerkprozessors.
Zudem konnten die Messungen auf einem Application Reference Board mit einem NP4GS3 Version C durchgeführt werden. Als Host kam ein IBM xSerie 335 Server mit einer PCI-X 64-bit/100MHz Schnittstelle, zum Einsatz. Da der PCI-X 64-bit/100MHz Standard vom BroadCom 5700 Gigabit-Ethernet Controller nicht unterstützt wird (siehe Tabelle 2), wird auf dem schnellsten gemeinsamen Standard kommuniziert. Bei dieser Konfiguration ist dies der PCI-X 64-bit/66MHz Standard.

5.3.1 Messwerte PCI 32-bit/33MHz Bus mit Datenpaketlänge 72 Bytes

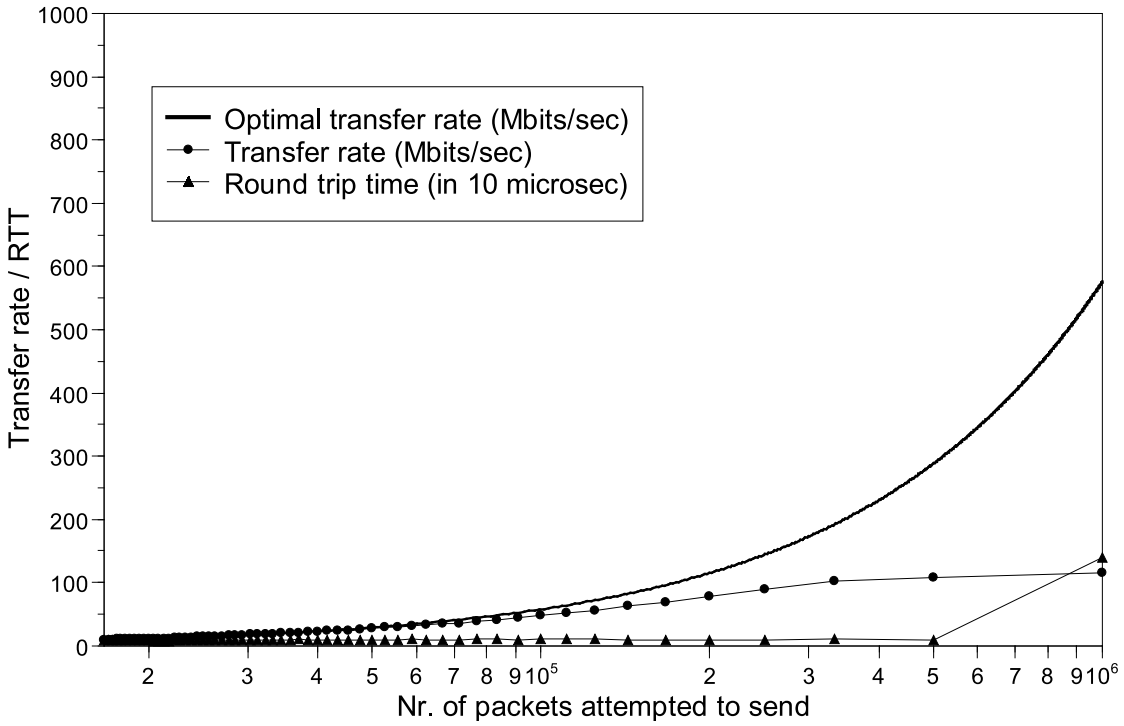


Abbildung 9: Transferrate und RTT (PCI 32-bit/33MHz Bus, Paketlänge 72 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: DELL Precision 340
             CPU: Intel(R) Pentium 4(TM) CPU 1.80GHz
          Memory: 512 MBytes
             Bus: PCI 32-bit/33MHz
              OS: Red Hat 7.3 (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 72 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	98678	200170	115.298	1389.8
500000	100000	100000	186093	107.190	94.8
333333	100000	100000	176202	101.492	104.2
250000	100000	100000	154570	89.033	93.8
200000	100000	100000	135213	77.883	97.1
166667	100000	100000	120488	69.401	96.0
142857	100000	100000	110792	63.816	97.4
125000	100000	100000	98115	56.514	101.3

111111	100000	100000	88581	51.022	103.4
100000	100000	100000	82382	47.452	99.2
90909	100000	100000	78574	45.258	97.4
83333	100000	100000	70847	40.808	100.1
76923	100000	100000	68435	39.419	99.2
71429	100000	100000	62367	35.923	95.9
66667	100000	100000	60275	34.719	93.8
62500	100000	100000	58691	33.806	96.0
58824	100000	100000	54168	31.201	99.4
55556	100000	100000	50771	29.244	98.2
52632	100000	100000	50731	29.221	95.8
50000	100000	100000	48192	27.759	92.2
47619	100000	100000	45521	26.220	88.7
45455	100000	100000	41515	23.912	88.9
43478	100000	100000	40618	23.396	88.7
41667	100000	100000	40611	23.392	89.4
40000	100000	100000	39164	22.558	96.1
38462	100000	100000	36986	21.304	96.1
37037	100000	100000	35332	20.351	99.1
35714	100000	100000	35188	20.268	97.0
34483	100000	100000	33625	19.368	95.3
33333	100000	100000	30631	17.644	98.2
32258	100000	100000	30469	17.550	97.5
31250	100000	100000	30469	17.550	96.0
30303	100000	100000	30177	17.382	88.4
29412	100000	100000	29097	16.760	90.1
28571	100000	100000	28203	16.245	89.6
27778	100000	100000	27096	15.608	90.2
27027	100000	100000	26873	15.479	89.3
26316	100000	100000	25393	14.626	82.2
25641	100000	100000	25386	14.623	96.7
25000	100000	100000	24725	14.241	88.0
24390	100000	100000	23954	13.798	89.0
23810	100000	100000	23624	13.607	90.6
23256	100000	100000	22792	13.128	88.3
22727	100000	100000	22396	12.900	91.2
22222	100000	100000	21860	12.591	88.2
21739	100000	100000	20390	11.745	74.5
21277	100000	100000	20312	11.700	72.9
20833	100000	100000	20310	11.699	72.2
20408	100000	100000	20310	11.699	70.5
20000	100000	100000	19918	11.473	94.8
19608	100000	100000	19524	11.246	91.9
19231	100000	100000	19116	11.011	93.3
18868	100000	100000	18801	10.829	89.9
18519	100000	100000	18279	10.529	89.3
18182	100000	100000	18043	10.393	91.3
17857	100000	100000	17770	10.235	92.4
17544	100000	100000	17378	10.010	95.1
17241	100000	100000	16927	9.750	98.6
16949	100000	100000	16926	9.750	96.9
16667	100000	100000	16507	9.508	90.7

5.3.2 Messwerte PCI 32-bit/33MHz Bus mit Datenpaketlänge 350 Bytes

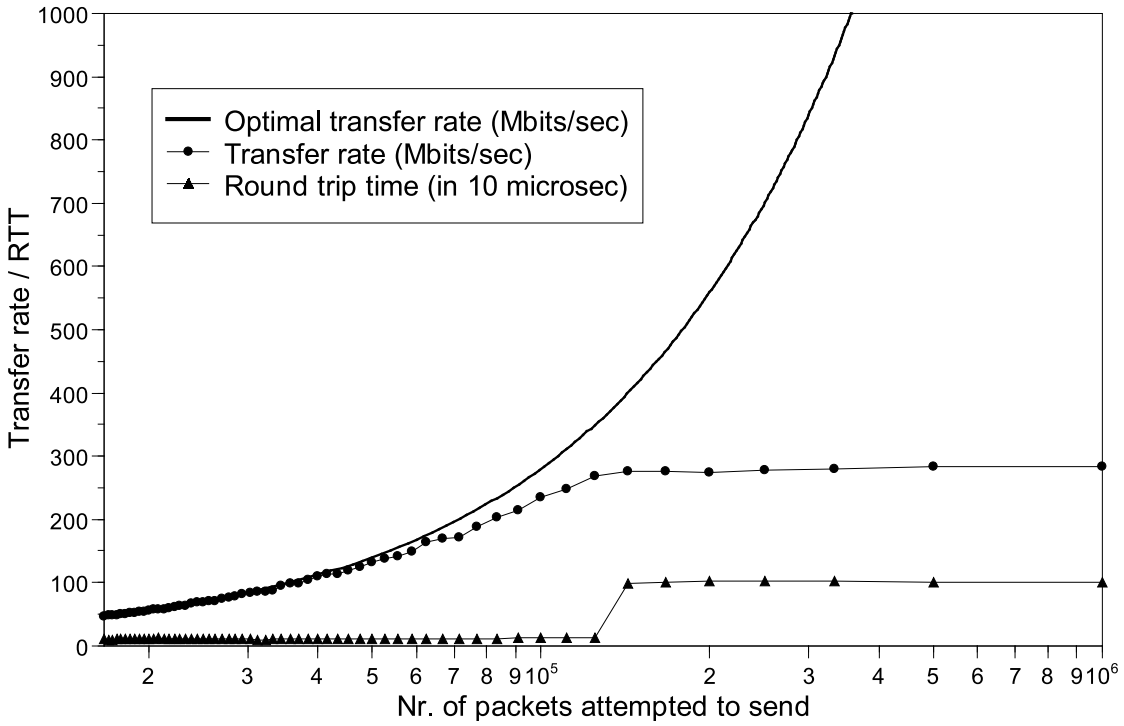


Abbildung 10: Transferrate und RTT (PCI 32-bit/33MHz Bus, Paketlänge 350 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: DELL Precision 340
             CPU: Intel(R) Pentium 4(TM) CPU 1.80GHz
          Memory: 512 MBytes
             Bus: PCI 32-bit/33MHz
              OS: Red Hat 7.3 (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 350 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	27019	101205	283.374	1010.0
500000	100000	36169	101160	283.247	1008.3
333333	100000	48760	99565	278.782	1020.9
250000	100000	62057	99474	278.527	1016.5
200000	100000	73043	97959	274.285	1027.1
166667	100000	86584	98821	276.698	1011.3
142857	100000	98654	98708	276.383	991.8
125000	100000	100000	96139	269.190	132.9

111111	100000	100000	88297	247.231	127.8
100000	100000	100000	84088	235.445	121.6
90909	100000	100000	76544	214.322	117.9
83333	100000	100000	72634	203.376	116.3
76923	100000	100000	67414	188.759	112.1
71429	100000	100000	61281	171.586	111.0
66667	100000	100000	60300	168.840	109.9
62500	100000	100000	58533	163.891	108.3
58824	100000	100000	53397	149.511	106.7
55556	100000	100000	50751	142.104	113.9
52632	100000	100000	49490	138.571	115.8
50000	100000	100000	47359	132.605	112.8
47619	100000	100000	44383	124.271	111.8
45455	100000	100000	42468	118.910	110.2
43478	100000	100000	40621	113.738	109.8
41667	100000	100000	40610	113.709	107.1
40000	100000	100000	39321	110.099	104.7
38462	100000	100000	37011	103.630	100.6
37037	100000	100000	35370	99.035	100.6
35714	100000	100000	35292	98.817	109.0
34483	100000	100000	33598	94.075	103.6
33333	100000	100000	31359	87.806	99.3
32258	100000	100000	30473	85.324	97.7
31250	100000	100000	30465	85.302	96.3
30303	100000	100000	30177	84.496	108.3
29412	100000	100000	29093	81.461	105.9
28571	100000	100000	28078	78.618	110.0
27778	100000	100000	27079	75.820	107.5
27027	100000	100000	26791	75.014	104.6
26316	100000	100000	25411	71.149	102.0
25641	100000	100000	25385	71.077	103.6
25000	100000	100000	24808	69.463	108.7
24390	100000	100000	24296	68.028	105.5
23810	100000	100000	23683	66.313	108.5
23256	100000	100000	22846	63.968	109.0
22727	100000	100000	22418	62.770	108.1
22222	100000	100000	21926	61.393	107.6
21739	100000	100000	21422	59.981	108.2
21277	100000	100000	20744	58.084	114.1
20833	100000	100000	20312	56.873	120.7
20408	100000	100000	20310	56.868	116.3
20000	100000	100000	19920	55.775	99.8
19608	100000	100000	19521	54.659	100.1
19231	100000	100000	19108	53.503	99.8
18868	100000	100000	18746	52.490	102.0
18519	100000	100000	18283	51.193	99.8
18182	100000	100000	18039	50.508	102.7
17857	100000	100000	17768	49.750	99.0
17544	100000	100000	17377	48.657	103.6
17241	100000	100000	16925	47.390	98.6
16949	100000	100000	16926	47.394	97.7
16667	100000	100000	16513	46.235	102.0

5.3.3 Messwerte PCI 32-bit/33MHz Bus mit Datenpaketlänge 700 Bytes

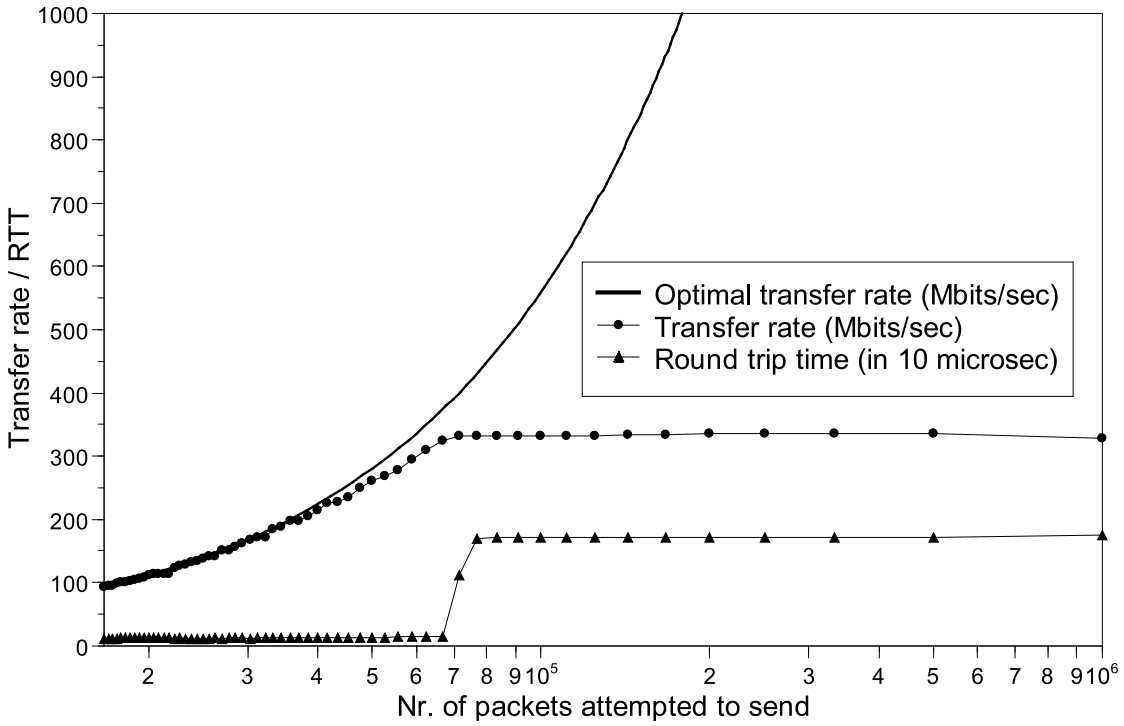


Abbildung 11: Transferrate und RTT (PCI 32-bit/33MHz Bus, Paketlänge 700 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: DELL Precision 340
             CPU: Intel(R) Pentium 4(TM) CPU 1.80GHz
          Memory: 512 MBytes
             Bus: PCI 32-bit/33MHz
              OS: Red Hat 7.3 (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 700 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	16393	58598	328.151	1749.4
500000	100000	16454	60073	336.410	1708.6
333333	100000	23595	60003	336.017	1709.3
250000	100000	30965	59932	335.620	1708.6
200000	100000	38299	59829	335.043	1708.7
166667	100000	45495	59770	334.712	1708.3
142857	100000	52392	59593	333.723	1710.3
125000	100000	59549	59413	332.714	1711.9

111111	100000	66660	59341	332.311	1711.1
100000	100000	73624	59324	332.216	1708.5
90909	100000	80462	59324	332.216	1705.6
83333	100000	86814	59123	331.090	1707.5
76923	100000	93463	59320	332.192	1694.8
71429	100000	100000	59393	332.598	1121.6
66667	100000	100000	57860	324.016	152.0
62500	100000	100000	55406	310.274	144.7
58824	100000	100000	52564	294.359	139.8
55556	100000	100000	49618	277.862	137.6
52632	100000	100000	48070	269.194	135.6
50000	100000	100000	46452	260.133	133.3
47619	100000	100000	44684	250.231	131.3
45455	100000	100000	41866	234.451	132.5
43478	100000	100000	40659	227.692	133.9
41667	100000	100000	40174	224.974	131.6
40000	100000	100000	38427	215.192	128.4
38462	100000	100000	36483	204.303	127.1
37037	100000	100000	35348	197.951	128.5
35714	100000	100000	35159	196.891	126.5
34483	100000	100000	33531	187.776	123.8
33333	100000	100000	32940	184.465	120.7
32258	100000	100000	30500	170.799	129.3
31250	100000	100000	30468	170.623	128.0
30303	100000	100000	30100	168.558	115.4
29412	100000	100000	28905	161.866	118.0
28571	100000	100000	27871	156.076	119.8
27778	100000	100000	27065	151.566	118.4
27027	100000	100000	26796	150.059	115.1
26316	100000	100000	25390	142.181	119.3
25641	100000	100000	25385	142.155	116.5
25000	100000	100000	24750	138.602	113.9
24390	100000	100000	24022	134.522	117.4
23810	100000	100000	23654	132.460	114.1
23256	100000	100000	22834	127.871	117.0
22727	100000	100000	22515	126.087	119.3
22222	100000	100000	21976	123.065	116.3
21739	100000	100000	20323	113.811	122.2
21277	100000	100000	20312	113.750	121.5
20833	100000	100000	20310	113.735	120.8
20408	100000	100000	20310	113.734	117.9
20000	100000	100000	19892	111.396	124.1
19608	100000	100000	19429	108.801	124.0
19231	100000	100000	19071	106.798	122.7
18868	100000	100000	18707	104.761	121.3
18519	100000	100000	18280	102.369	122.6
18182	100000	100000	18026	100.943	121.4
17857	100000	100000	17769	99.507	119.5
17544	100000	100000	17474	97.855	116.7
17241	100000	100000	17057	95.519	115.4
16949	100000	100000	16925	94.779	113.2
16667	100000	100000	16615	93.044	114.6

5.3.4 Messwerte PCI 32-bit/33MHz Bus mit Datenpaketlänge 1050 Bytes

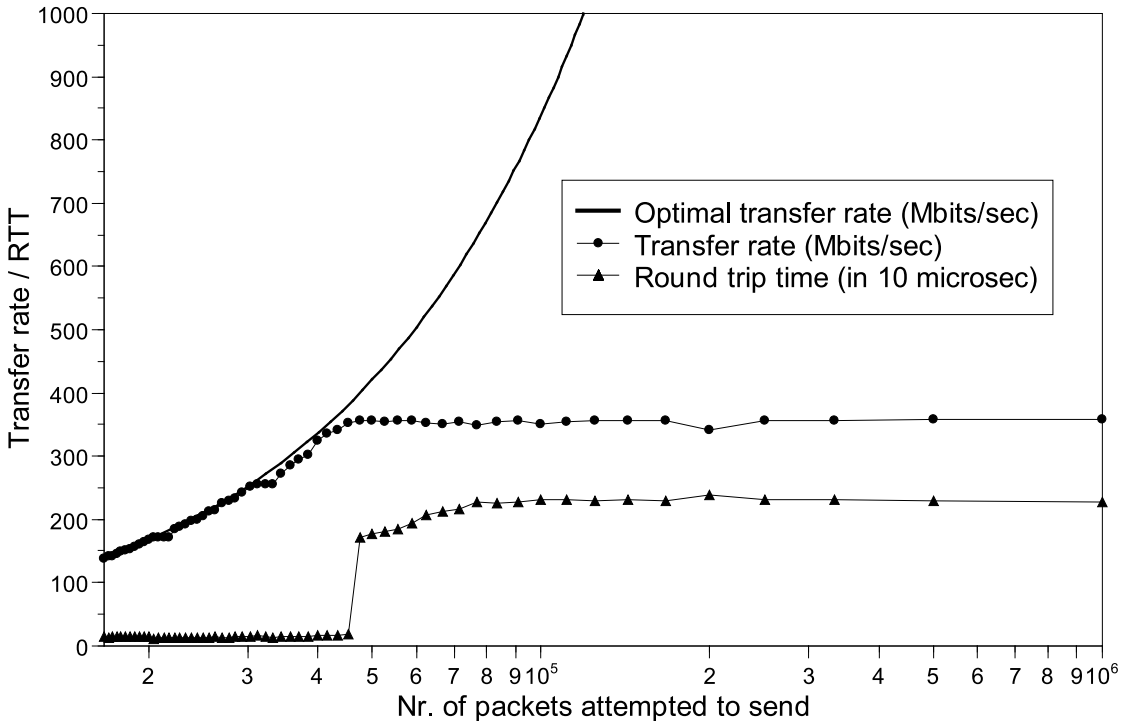


Abbildung 12: Transferrate und RTT (PCI 32-bit/33MHz Bus, Paketlänge 1050 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: DELL Precision 340
             CPU: Intel(R) Pentium 4(TM) CPU 1.80GHz
          Memory: 512 MBytes
             Bus: PCI 32-bit/33MHz
              OS: Red Hat 7.3 (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 1050 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	5603	42683	358.537	2265.1
500000	100000	10551	42712	358.779	2292.5
333333	100000	15535	42496	356.970	2307.4
250000	100000	20468	42387	356.049	2309.7
200000	100000	24242	40715	342.007	2386.9
166667	100000	30289	42322	355.508	2290.1
142857	100000	35099	42339	355.644	2304.8
125000	100000	40044	42364	355.857	2298.9

111111	100000	44614	42128	353.873	2308.0
100000	100000	48771	41725	350.493	2319.6
90909	100000	54247	42337	355.635	2269.0
83333	100000	58333	42092	353.575	2260.2
76923	100000	61882	41572	349.207	2281.3
71429	100000	67166	42126	353.860	2157.0
66667	100000	70436	41817	351.260	2133.4
62500	100000	75488	41873	351.733	2064.0
58824	100000	81065	42333	355.596	1940.3
55556	100000	84671	42320	355.485	1839.8
52632	100000	88226	42134	353.926	1808.3
50000	100000	93712	42475	356.788	1776.6
47619	100000	98732	42306	355.370	1721.9
45455	100000	100000	42001	352.805	173.9
43478	100000	100000	40646	341.428	158.3
41667	100000	100000	39924	335.358	157.4
40000	100000	100000	38639	324.567	159.3
38462	100000	100000	35970	302.149	150.2
37037	100000	100000	35015	294.127	150.8
35714	100000	100000	34067	286.159	151.5
34483	100000	100000	32376	271.958	143.2
33333	100000	100000	30513	256.311	131.9
32258	100000	100000	30470	255.949	137.0
31250	100000	100000	30454	255.815	159.3
30303	100000	100000	29987	251.894	148.5
29412	100000	100000	28805	241.965	144.3
28571	100000	100000	27817	233.667	136.4
27778	100000	100000	27305	229.364	134.3
27027	100000	100000	26842	225.472	135.3
26316	100000	100000	25443	213.719	136.3
25641	100000	100000	25349	212.929	133.5
25000	100000	100000	24303	204.142	131.6
24390	100000	100000	23699	199.076	129.8
23810	100000	100000	23536	197.706	128.6
23256	100000	100000	22790	191.439	133.8
22727	100000	100000	22459	188.655	129.2
22222	100000	100000	21857	183.596	131.2
21739	100000	100000	20330	170.769	122.1
21277	100000	100000	20312	170.619	121.4
20833	100000	100000	20310	170.603	120.6
20408	100000	100000	20310	170.601	117.4
20000	100000	100000	19919	167.323	138.3
19608	100000	100000	19506	163.854	140.8
19231	100000	100000	19057	160.081	140.9
18868	100000	100000	18661	156.748	140.3
18519	100000	100000	18279	153.548	143.7
18182	100000	100000	18027	151.430	140.2
17857	100000	100000	17766	149.231	139.0
17544	100000	100000	17380	145.989	136.4
17241	100000	100000	16927	142.187	137.9
16949	100000	100000	16924	142.165	135.4
16667	100000	100000	16502	138.620	138.3

5.3.5 Messwerte PCI 32-bit/33MHz Bus mit Datenpaketlänge 1460 Bytes

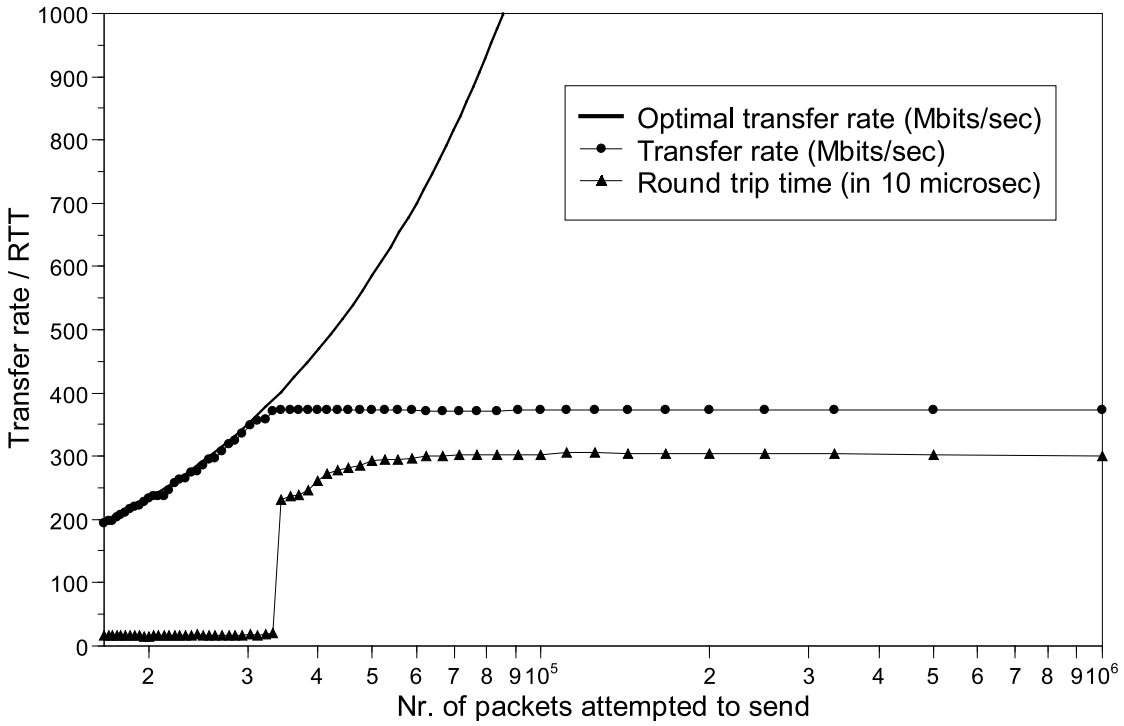


Abbildung 13: Transferrate und RTT (PCI 32-bit/33MHz Bus, Paketlänge 1460 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: DELL Precision 340
             CPU: Intel(R) Pentium 4(TM) CPU 1.80GHz
          Memory: 512 MBytes
             Bus: PCI 32-bit/33MHz
              OS: Red Hat 7.3 (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 1460 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	4074	31951	373.184	3011.1
500000	100000	7687	31964	373.334	3026.5
333333	100000	11256	31903	372.631	3036.5
250000	100000	14888	31893	372.511	3038.9
200000	100000	18443	31886	372.433	3039.3
166667	100000	22029	31878	372.331	3037.4
142857	100000	25579	31886	372.424	3034.0
125000	100000	28925	31886	372.425	3055.7

111111	100000	32312	31878	372.333	3058.9
100000	100000	35664	31891	372.481	3029.0
90909	100000	39201	31887	372.437	3026.6
83333	100000	42769	31851	372.019	3027.6
76923	100000	45330	31820	371.653	3028.4
71429	100000	49431	31788	371.290	3020.6
66667	100000	51903	31807	371.507	3008.5
62500	100000	54810	31835	371.834	2994.1
58824	100000	59106	31856	372.082	2971.5
55556	100000	61775	31868	372.215	2953.0
52632	100000	64134	31866	372.191	2953.2
50000	100000	68155	31900	372.587	2920.0
47619	100000	71998	31922	372.847	2861.0
45455	100000	75521	31906	372.659	2818.4
43478	100000	78108	31923	372.860	2770.0
41667	100000	80240	31927	372.903	2727.3
40000	100000	84153	31934	372.988	2608.7
38462	100000	87706	31940	373.061	2465.0
37037	100000	90467	31937	373.022	2383.8
35714	100000	94415	31950	373.180	2358.2
34483	100000	98714	31971	373.427	2302.5
33333	100000	100000	31701	370.270	195.6
32258	100000	100000	30669	358.213	174.7
31250	100000	100000	30433	355.456	173.4
30303	100000	100000	29896	349.185	175.0
29412	100000	100000	28734	335.612	173.3
28571	100000	100000	27856	325.363	170.7
27778	100000	100000	27377	319.768	166.3
27027	100000	100000	26380	308.120	164.3
26316	100000	100000	25387	296.516	160.5
25641	100000	100000	25158	293.844	170.9
25000	100000	100000	24463	285.724	172.9
24390	100000	100000	23686	276.650	174.0
23810	100000	100000	23386	273.144	169.1
23256	100000	100000	22653	264.587	170.5
22727	100000	100000	22432	262.007	164.0
22222	100000	100000	22021	257.209	163.8
21739	100000	100000	21081	246.224	165.3
21277	100000	100000	20329	237.443	169.2
20833	100000	100000	20312	237.239	167.9
20408	100000	100000	20309	237.215	163.9
20000	100000	100000	19872	232.107	152.3
19608	100000	100000	19408	226.682	154.5
19231	100000	100000	19028	222.244	155.9
18868	100000	100000	18777	219.317	164.1
18519	100000	100000	18454	215.544	163.3
18182	100000	100000	17984	210.055	166.6
17857	100000	100000	17630	205.923	165.3
17544	100000	100000	17342	202.559	163.5
17241	100000	100000	16927	197.709	160.4
16949	100000	100000	16924	197.675	157.3
16667	100000	100000	16514	192.886	155.7

5.3.6 Messwerte PCI 64-bit/66MHz Bus mit Datenpaketlänge 72 Bytes

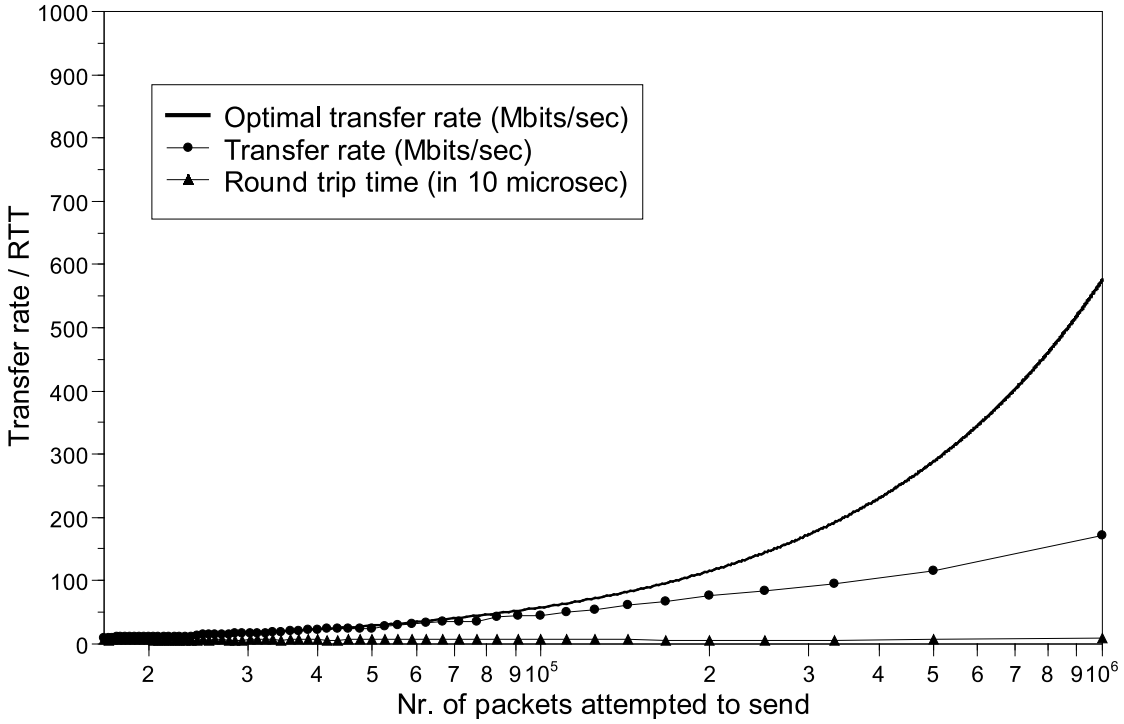


Abbildung 14: Transferrate und RTT (PCI 64-bit/66MHz Bus, Paketlänge 72 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 72 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	100000	297985	171.639	81.2
500000	100000	100000	198710	114.457	65.5
333333	100000	100000	163689	94.285	55.7
250000	100000	100000	145448	83.778	60.6
200000	100000	100000	131019	75.467	59.2
166667	100000	100000	114693	66.063	59.0
142857	100000	100000	106424	61.300	64.2
125000	100000	100000	92670	53.378	68.0

111111	100000	100000	88070	50.728	68.1
100000	100000	100000	78452	45.189	70.9
90909	100000	100000	76925	44.309	70.3
83333	100000	100000	75421	43.443	68.8
76923	100000	100000	61944	35.679	68.8
71429	100000	100000	59978	34.547	68.4
66667	100000	100000	59504	34.274	67.2
62500	100000	100000	57810	33.298	66.6
58824	100000	100000	55500	31.968	63.8
55556	100000	100000	49676	28.613	62.7
52632	100000	100000	48446	27.905	61.9
50000	100000	100000	41790	24.071	63.9
47619	100000	100000	40186	23.147	62.6
45455	100000	100000	40184	23.146	62.0
43478	100000	100000	40172	23.139	61.4
41667	100000	100000	40146	23.124	60.4
40000	100000	100000	39406	22.698	67.7
38462	100000	100000	37193	21.423	64.5
37037	100000	100000	35339	20.355	63.6
35714	100000	100000	33716	19.421	63.3
34483	100000	100000	33354	19.212	61.4
33333	100000	100000	31714	18.267	66.8
32258	100000	100000	30112	17.345	67.7
31250	100000	100000	30102	17.339	66.6
30303	100000	100000	29967	17.261	63.4
29412	100000	100000	28441	16.382	59.8
28571	100000	100000	27015	15.561	60.1
27778	100000	100000	26464	15.243	57.7
27027	100000	100000	26459	15.240	67.2
26316	100000	100000	25032	14.419	61.8
25641	100000	100000	24978	14.387	59.7
25000	100000	100000	23688	13.644	62.6
24390	100000	100000	22453	12.933	57.0
23810	100000	100000	20134	11.597	48.7
23256	100000	100000	20117	11.587	48.7
22727	100000	100000	20117	11.587	48.7
22222	100000	100000	20114	11.586	48.8
21739	100000	100000	20113	11.585	48.8
21277	100000	100000	20113	11.585	48.8
20833	100000	100000	20114	11.586	49.0
20408	100000	100000	20112	11.585	49.5
20000	100000	100000	19949	11.490	68.3
19608	100000	100000	19455	11.206	66.2
19231	100000	100000	18952	10.916	65.3
18868	100000	100000	18641	10.737	64.7
18519	100000	100000	18258	10.517	64.2
18182	100000	100000	17890	10.305	66.4
17857	100000	100000	17591	10.132	65.0
17544	100000	100000	17235	9.928	63.6
17241	100000	100000	16800	9.677	62.1
16949	100000	100000	16774	9.662	60.4
16667	100000	100000	16140	9.297	63.2

5.3.7 Messwerte PCI 64-bit/66MHz Bus mit Datenpaketlänge 350 Bytes

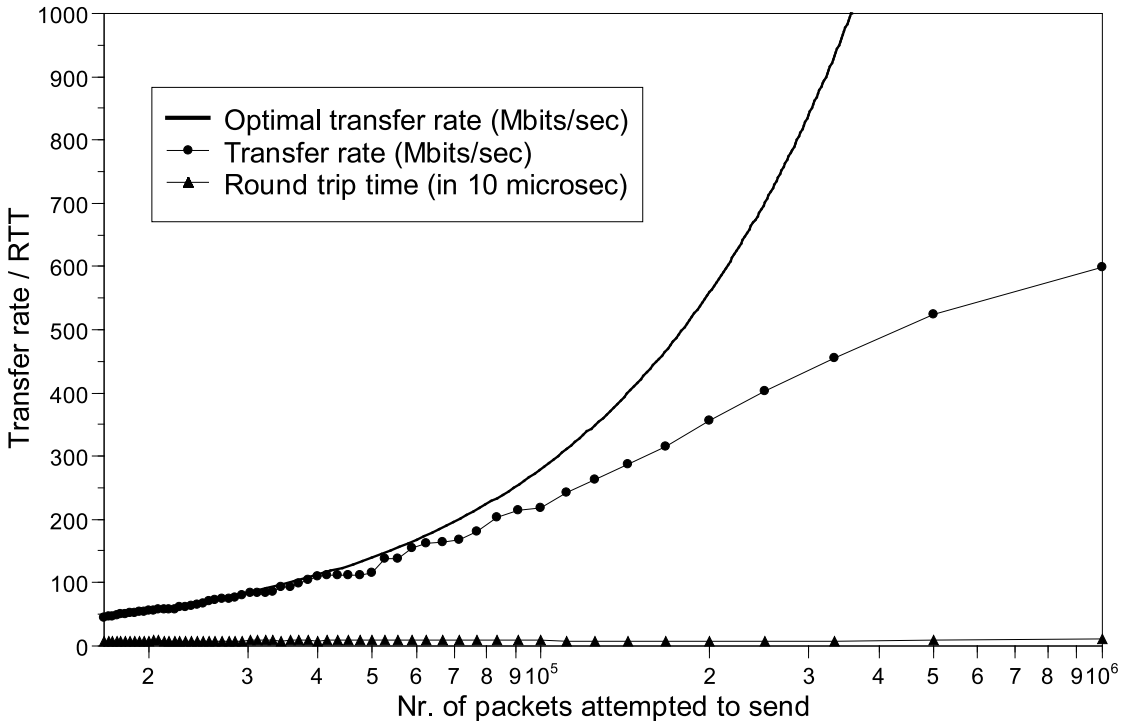


Abbildung 15: Transferrate und RTT (PCI 64-bit/66MHz Bus, Paketlänge 350 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 350 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	100000	214044	599.324	113.4
500000	100000	100000	186911	523.351	87.5
333333	100000	100000	162519	455.052	71.4
250000	100000	100000	144056	403.357	69.0
200000	100000	100000	127389	356.689	67.1
166667	100000	100000	112845	315.965	69.9
142857	100000	100000	102648	287.415	71.3
125000	100000	100000	93897	262.912	72.1

111111	100000	100000	86500	242.200	76.3
100000	100000	100000	77630	217.364	82.1
90909	100000	100000	76845	215.167	81.1
83333	100000	100000	72586	203.241	81.6
76923	100000	100000	64765	181.341	83.8
71429	100000	100000	59783	167.392	83.6
66667	100000	100000	58519	163.852	83.2
62500	100000	100000	57700	161.561	82.2
58824	100000	100000	55222	154.621	83.3
55556	100000	100000	49520	138.657	82.2
52632	100000	100000	49275	137.970	82.8
50000	100000	100000	41048	114.935	87.3
47619	100000	100000	40176	112.494	86.9
45455	100000	100000	40167	112.468	86.3
43478	100000	100000	40149	112.418	85.6
41667	100000	100000	40085	112.239	84.2
40000	100000	100000	39356	110.196	79.6
38462	100000	100000	37201	104.163	81.3
37037	100000	100000	35233	98.653	81.6
35714	100000	100000	33404	93.530	82.0
34483	100000	100000	33308	93.262	80.2
33333	100000	100000	30501	85.402	84.0
32258	100000	100000	30102	84.286	82.9
31250	100000	100000	29914	83.760	82.0
30303	100000	100000	29572	82.802	80.9
29412	100000	100000	28609	80.104	78.2
28571	100000	100000	27446	76.850	75.5
27778	100000	100000	26455	74.075	77.2
27027	100000	100000	26454	74.072	73.2
26316	100000	100000	25701	71.962	78.1
25641	100000	100000	25083	70.232	73.9
25000	100000	100000	23963	67.096	78.5
24390	100000	100000	23148	64.814	76.7
23810	100000	100000	22717	63.607	75.8
23256	100000	100000	22146	62.008	77.5
22727	100000	100000	21871	61.239	79.4
22222	100000	100000	20278	56.779	77.4
21739	100000	100000	20276	56.772	77.0
21277	100000	100000	20266	56.745	77.6
20833	100000	100000	20221	56.619	82.1
20408	100000	100000	20158	56.443	88.3
20000	100000	100000	19953	55.868	75.7
19608	100000	100000	19455	54.475	76.7
19231	100000	100000	18961	53.091	76.2
18868	100000	100000	18640	52.192	75.1
18519	100000	100000	18263	51.137	74.9
18182	100000	100000	17891	50.094	74.7
17857	100000	100000	17592	49.258	73.5
17544	100000	100000	17236	48.260	73.2
17241	100000	100000	16794	47.022	73.3
16949	100000	100000	16774	46.967	71.5
16667	100000	100000	16145	45.205	71.2

5.3.8 Messwerte PCI 64-bit/66MHz Bus mit Datenpaketlänge 700 Bytes

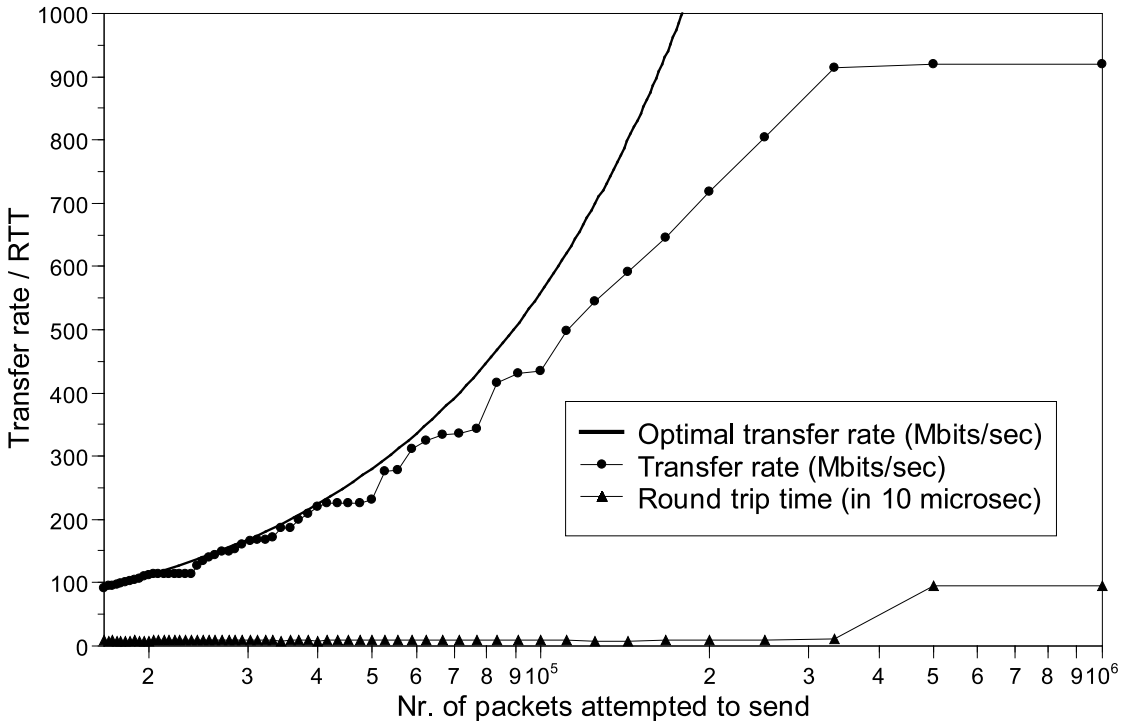


Abbildung 16: Transferrate und RTT (PCI 64-bit/66MHz Bus, Paketlänge 700 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 700 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	77214	164145	919.213	946.0
500000	100000	84738	164177	919.393	954.5
333333	100000	100000	163270	914.312	104.4
250000	100000	100000	143479	803.484	87.3
200000	100000	100000	128362	718.826	85.0
166667	100000	100000	115421	646.356	82.2
142857	100000	100000	105773	592.328	79.5
125000	100000	100000	97390	545.385	79.8

111111	100000	100000	89017	498.497	80.6
100000	100000	100000	77644	434.807	81.7
90909	100000	100000	76948	430.908	80.5
83333	100000	100000	74250	415.798	85.5
76923	100000	100000	61185	342.634	83.7
71429	100000	100000	59994	335.964	83.3
66667	100000	100000	59560	333.535	82.6
62500	100000	100000	57796	323.659	81.5
58824	100000	100000	55456	310.553	82.5
55556	100000	100000	49610	277.817	81.4
52632	100000	100000	49276	275.947	85.3
50000	100000	100000	41335	231.475	86.7
47619	100000	100000	40176	224.986	86.3
45455	100000	100000	40174	224.974	85.7
43478	100000	100000	40167	224.937	84.9
41667	100000	100000	40144	224.808	83.6
40000	100000	100000	39412	220.707	79.3
38462	100000	100000	37242	208.554	80.5
37037	100000	100000	35490	198.745	81.9
35714	100000	100000	33423	187.170	81.3
34483	100000	100000	33306	186.513	79.6
33333	100000	100000	30497	170.782	83.7
32258	100000	100000	30101	168.564	83.0
31250	100000	100000	29960	167.778	82.1
30303	100000	100000	29598	165.749	81.3
29412	100000	100000	28523	159.729	81.5
28571	100000	100000	27346	153.139	83.7
27778	100000	100000	26449	148.113	82.0
27027	100000	100000	26448	148.107	80.6
26316	100000	100000	25465	142.606	83.1
25641	100000	100000	25083	140.463	81.3
25000	100000	100000	23987	134.328	87.6
24390	100000	100000	22457	125.757	90.0
23810	100000	100000	20120	112.670	97.3
23256	100000	100000	20109	112.608	97.3
22727	100000	100000	20108	112.607	97.3
22222	100000	100000	20108	112.604	97.2
21739	100000	100000	20108	112.606	97.1
21277	100000	100000	20107	112.601	96.9
20833	100000	100000	20108	112.602	96.4
20408	100000	100000	20107	112.601	94.9
20000	100000	100000	19951	111.723	76.5
19608	100000	100000	19455	108.951	78.8
19231	100000	100000	18976	106.263	79.2
18868	100000	100000	18659	104.491	80.3
18519	100000	100000	18309	102.529	79.6
18182	100000	100000	17903	100.258	78.6
17857	100000	100000	17595	98.534	78.6
17544	100000	100000	17237	96.527	78.7
17241	100000	100000	16798	94.069	80.4
16949	100000	100000	16774	93.932	78.5
16667	100000	100000	16135	90.355	81.2

5.3.9 Messwerte PCI 64-bit/66MHz Bus mit Datenpaketlänge 1050 Bytes

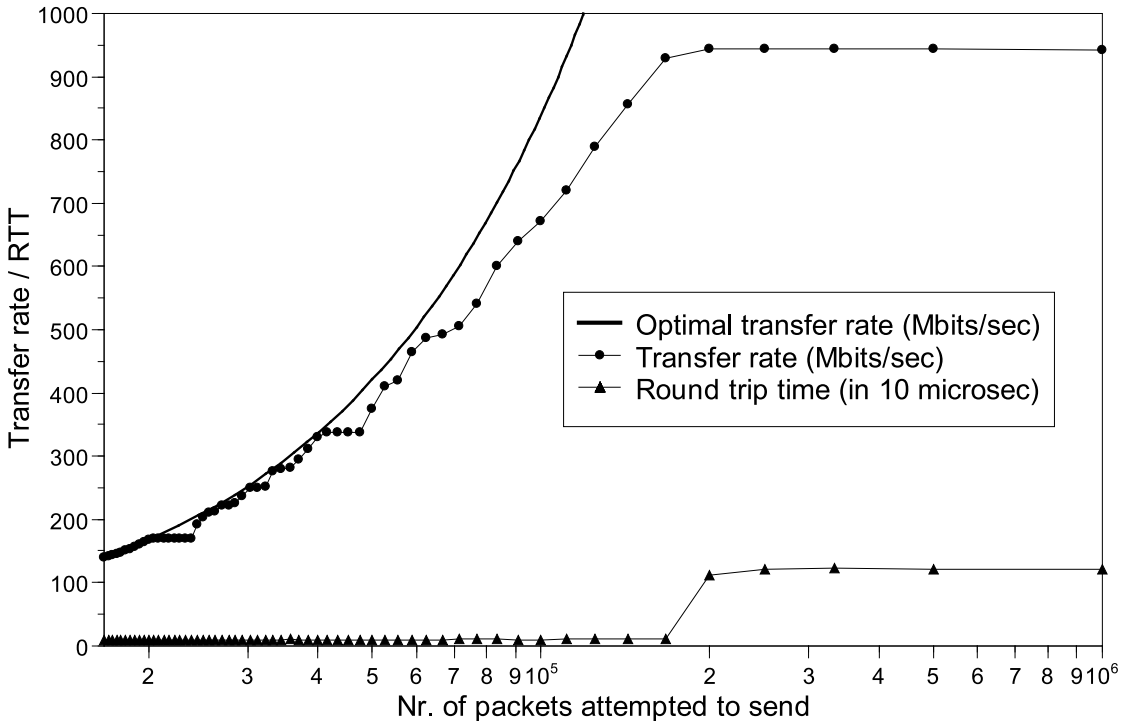


Abbildung 17: Transferrate und RTT (PCI 64-bit/66MHz Bus, Paketlänge 1050 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 1050 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
1000000	100000	31597	112228	942.712	1201.1
500000	100000	40436	112330	943.570	1214.8
333333	100000	58083	112403	944.188	1237.7
250000	100000	74735	112431	944.418	1209.8
200000	100000	90363	112465	944.702	1124.6
166667	100000	100000	110595	928.994	108.0
142857	100000	100000	102022	856.989	102.9
125000	100000	100000	93983	789.458	101.1

111111	100000	100000	85743	720.239	99.0
100000	100000	100000	79919	671.320	98.2
90909	100000	100000	76237	640.394	97.2
83333	100000	100000	71415	599.884	100.8
76923	100000	100000	64374	540.739	99.9
71429	100000	100000	60214	505.794	100.0
66667	100000	100000	58631	492.504	98.7
62500	100000	100000	58002	487.213	97.0
58824	100000	100000	55244	464.047	93.6
55556	100000	100000	49964	419.695	92.4
52632	100000	100000	48869	410.501	94.1
50000	100000	100000	44577	374.445	92.5
47619	100000	100000	40214	337.799	88.6
45455	100000	100000	40175	337.470	88.0
43478	100000	100000	40160	337.343	88.0
41667	100000	100000	40107	336.899	90.7
40000	100000	100000	39371	330.715	97.3
38462	100000	100000	37147	312.032	98.4
37037	100000	100000	34967	293.722	96.9
35714	100000	100000	33450	280.979	99.5
34483	100000	100000	33316	279.858	98.7
33333	100000	100000	32891	276.287	94.7
32258	100000	100000	29933	251.434	98.0
31250	100000	100000	29777	250.130	97.0
30303	100000	100000	29768	250.050	94.6
29412	100000	100000	28094	235.990	93.6
28571	100000	100000	26774	224.898	94.0
27778	100000	100000	26399	221.752	93.0
27027	100000	100000	26389	221.665	91.6
26316	100000	100000	25367	213.085	91.8
25641	100000	100000	24959	209.655	95.5
25000	100000	100000	24076	202.240	95.8
24390	100000	100000	22749	191.092	94.9
23810	100000	100000	20129	169.084	96.9
23256	100000	100000	20110	168.927	96.9
22727	100000	100000	20108	168.911	96.9
22222	100000	100000	20108	168.905	96.8
21739	100000	100000	20107	168.899	96.7
21277	100000	100000	20108	168.905	96.5
20833	100000	100000	20108	168.905	96.2
20408	100000	100000	20107	168.895	94.7
20000	100000	100000	19946	167.548	92.6
19608	100000	100000	19455	163.418	92.0
19231	100000	100000	18964	159.301	92.2
18868	100000	100000	18632	156.512	92.6
18519	100000	100000	18261	153.391	92.5
18182	100000	100000	17881	150.201	91.3
17857	100000	100000	17593	147.784	92.3
17544	100000	100000	17249	144.894	95.0
17241	100000	100000	17151	144.072	92.7
16949	100000	100000	16884	141.826	94.2
16667	100000	100000	16638	139.760	93.6

5.3.10 Messwerte PCI 64-bit/66MHz Bus mit Datenpaketlänge 1460 Bytes

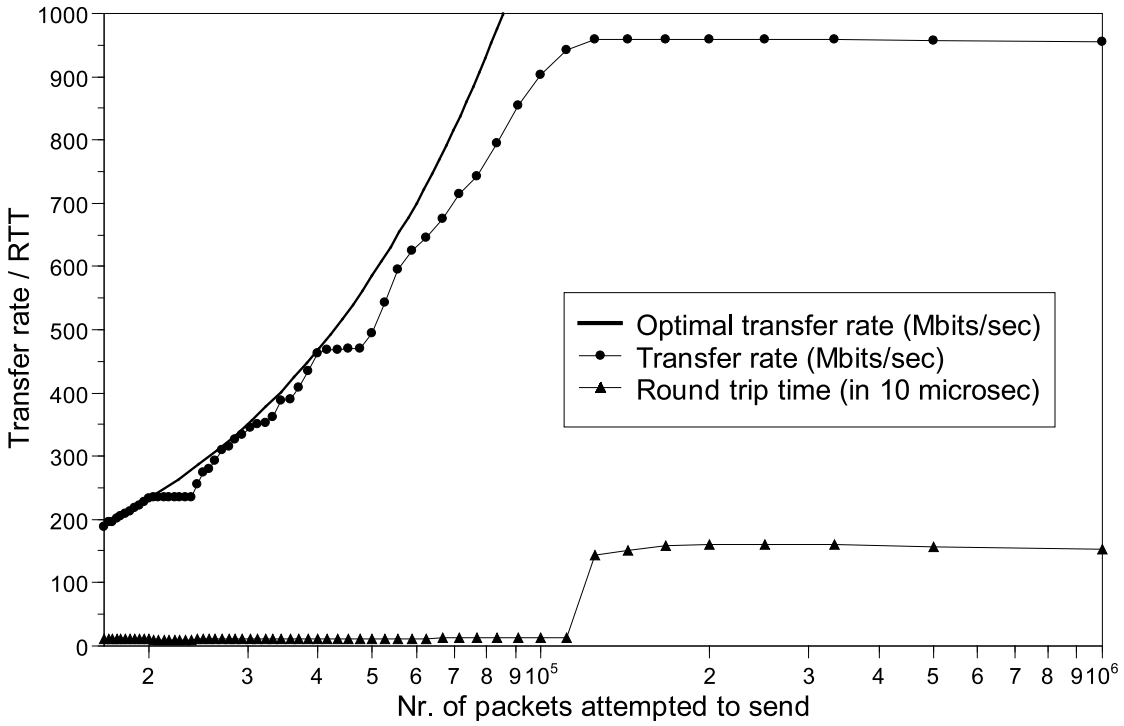


Abbildung 18: Transferrate und RTT (PCI 64-bit/66MHz Bus, Paketlänge 1460 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 1460 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	18998	81846	955.966	1531.4
500000	100000	28217	81971	957.426	1571.8
333333	100000	38795	82045	958.280	1597.4
250000	100000	50430	82084	958.743	1610.4
200000	100000	61426	82105	958.990	1608.1
166667	100000	72039	82132	959.297	1588.5
142857	100000	83636	82143	959.434	1515.9
125000	100000	91008	82150	959.506	1436.0

111111	100000	100000	80650	941.995	130.8
100000	100000	100000	77304	902.914	125.0
90909	100000	100000	73178	854.720	122.6
83333	100000	100000	67980	794.011	121.2
76923	100000	100000	63639	743.304	118.9
71429	100000	100000	61263	715.546	119.2
66667	100000	100000	57784	674.918	118.5
62500	100000	100000	55192	644.640	116.6
58824	100000	100000	53475	624.589	116.3
55556	100000	100000	50886	594.344	116.7
52632	100000	100000	46480	542.885	115.6
50000	100000	100000	42303	494.098	112.3
47619	100000	100000	40302	470.727	111.1
45455	100000	100000	40169	469.172	110.3
43478	100000	100000	40153	468.987	109.6
41667	100000	100000	40091	468.257	108.3
40000	100000	100000	39540	461.830	109.8
38462	100000	100000	37143	433.828	107.8
37037	100000	100000	34934	408.027	108.0
35714	100000	100000	33395	390.054	107.9
34483	100000	100000	33262	388.499	108.7
33333	100000	100000	30986	361.913	103.3
32258	100000	100000	30107	351.652	104.5
31250	100000	100000	30003	350.433	114.2
30303	100000	100000	29593	345.647	112.3
29412	100000	100000	28611	334.176	109.5
28571	100000	100000	27893	325.791	107.1
27778	100000	100000	27020	315.595	107.2
27027	100000	100000	26463	309.090	104.6
26316	100000	100000	25137	293.599	108.9
25641	100000	100000	23942	279.644	105.0
25000	100000	100000	23471	274.142	103.2
24390	100000	100000	21848	255.190	102.2
23810	100000	100000	20152	235.379	97.2
23256	100000	100000	20135	235.172	97.1
22727	100000	100000	20129	235.108	97.0
22222	100000	100000	20111	234.894	96.9
21739	100000	100000	20109	234.876	96.9
21277	100000	100000	20111	234.893	96.8
20833	100000	100000	20110	234.880	96.7
20408	100000	100000	20110	234.879	96.2
20000	100000	100000	19944	232.947	113.0
19608	100000	100000	19454	227.221	111.3
19231	100000	100000	18947	221.301	110.7
18868	100000	100000	18627	217.559	109.5
18519	100000	100000	18255	213.220	108.0
18182	100000	100000	17887	208.918	110.3
17857	100000	100000	17588	205.429	106.8
17544	100000	100000	17232	201.266	111.8
17241	100000	100000	16791	196.117	110.5
16949	100000	100000	16777	195.952	108.5
16667	100000	100000	16134	188.444	106.8

5.4 Messwerte Embedded PowerPC Processor Internal Control Point

5.4.1 Messwerte mit Datenpaketlänge 72 Bytes

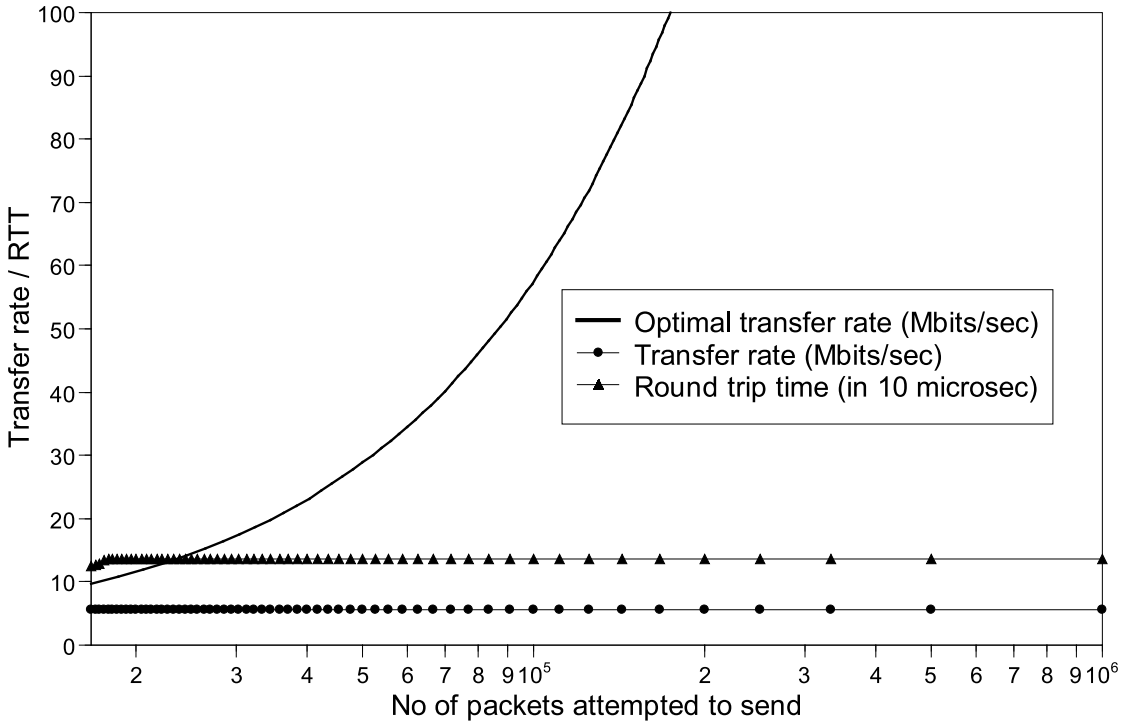


Abbildung 19: Transferrate und RTT (embedded PowerPC, Paketlänge 72 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: Application Referece Board
             CPU: embedded PowerPC 405 133 MHz
          Memory: 64 MBytes
             Bus: D6 DRAM Memory
              OS: Monta Vista Linux (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 72 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	100000	9807	5.649	135.9
500000	100000	100000	9807	5.649	135.9
333333	100000	100000	9807	5.649	135.9
250000	100000	100000	9807	5.649	135.9
200000	100000	100000	9807	5.649	135.9
166667	100000	100000	9807	5.649	135.9

142857	100000	100000	9807	5.649	135.9
125000	100000	100000	9807	5.649	135.9
111111	100000	100000	9807	5.649	135.9
100000	100000	100000	9807	5.649	135.9
90909	100000	100000	9807	5.649	135.9
83333	100000	100000	9807	5.649	135.9
76923	100000	100000	9807	5.649	135.9
71429	100000	100000	9807	5.649	135.9
66667	100000	100000	9807	5.649	135.9
62500	100000	100000	9807	5.649	135.9
58824	100000	100000	9807	5.649	135.9
55556	100000	100000	9807	5.649	135.9
52632	100000	100000	9807	5.649	135.9
50000	100000	100000	9807	5.649	135.9
47619	100000	100000	9807	5.649	135.9
45455	100000	100000	9807	5.649	135.9
43478	100000	100000	9807	5.649	135.9
41667	100000	100000	9807	5.649	135.9
40000	100000	100000	9807	5.649	135.9
38462	100000	100000	9807	5.649	135.9
37037	100000	100000	9807	5.649	135.9
35714	100000	100000	9807	5.649	135.8
34483	100000	100000	9807	5.649	135.8
33333	100000	100000	9807	5.649	135.8
32258	100000	100000	9806	5.648	135.8
31250	100000	100000	9807	5.649	135.8
30303	100000	100000	9807	5.649	135.8
29412	100000	100000	9807	5.649	135.8
28571	100000	100000	9807	5.649	135.8
27778	100000	100000	9807	5.649	135.8
27027	100000	100000	9807	5.649	135.8
26316	100000	100000	9807	5.649	135.9
25641	100000	100000	9807	5.649	135.8
25000	100000	100000	9807	5.649	135.8
24390	100000	100000	9807	5.649	135.8
23810	100000	100000	9807	5.649	135.8
23256	100000	100000	9807	5.649	135.8
22727	100000	100000	9807	5.649	135.8
22222	100000	100000	9807	5.649	135.8
21739	100000	100000	9807	5.649	135.8
21277	100000	100000	9806	5.648	135.8
20833	100000	100000	9807	5.649	135.8
20408	100000	100000	9807	5.649	135.8
20000	100000	100000	9807	5.649	135.8
19608	100000	100000	9807	5.649	135.8
19231	100000	100000	9807	5.649	135.8
18868	100000	100000	9807	5.649	135.8
18519	100000	100000	9807	5.649	135.8
18182	100000	100000	9807	5.649	135.8
17857	100000	100000	9807	5.649	135.8
17544	100000	100000	9782	5.634	134.1
17241	100000	100000	9696	5.585	128.2
16949	100000	100000	9674	5.572	126.7
16667	100000	100000	9640	5.553	124.3

5.4.2 Messwerte mit Datenpaketlänge 350 Bytes

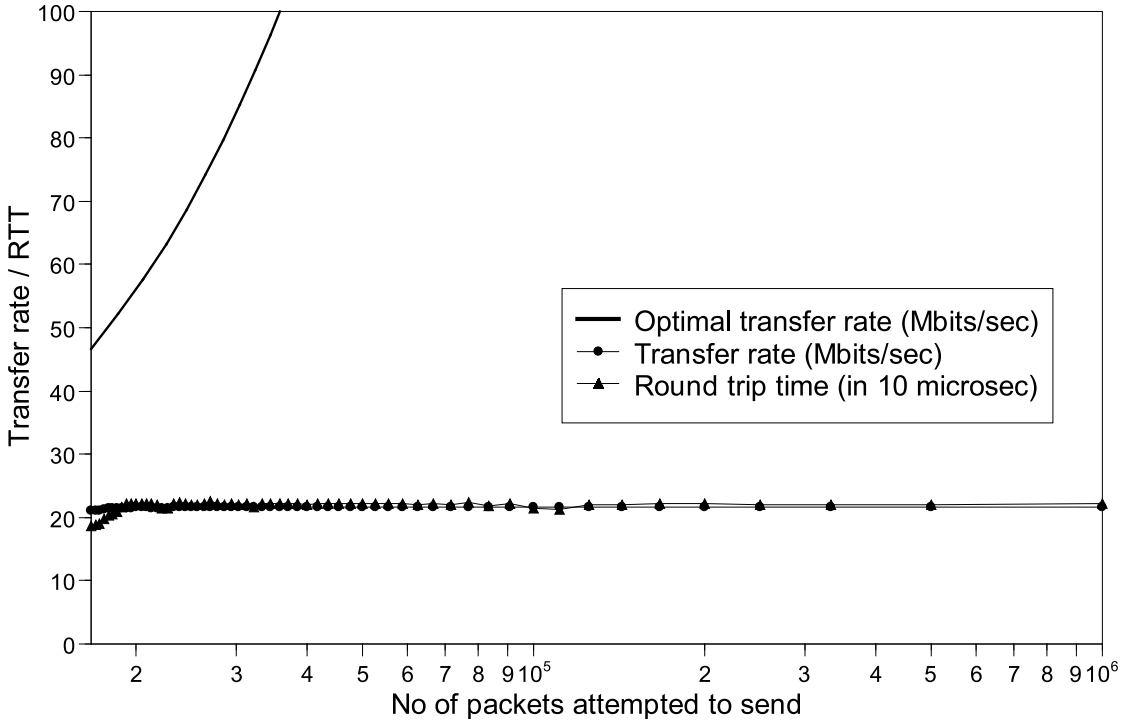


Abbildung 20: Transferrate und RTT (embedded PowerPC, Paketlänge 350 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: Application Referece Board
             CPU: embedded PowerPC 405 133 MHz
          Memory: 64 MBytes
             Bus: D6 DRAM Memory
              OS: Monta Vista Linux (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 350 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	100000	7711	21.590	222.1
500000	100000	100000	7711	21.590	220.7
333333	100000	100000	7711	21.591	220.8
250000	100000	100000	7711	21.590	220.7
200000	100000	100000	7711	21.591	221.7
166667	100000	100000	7711	21.590	221.3
142857	100000	100000	7711	21.592	220.8
125000	100000	100000	7711	21.592	220.2

111111	100000	100000	7688	21.525	212.4
100000	100000	100000	7690	21.532	214.4
90909	100000	100000	7711	21.591	221.4
83333	100000	100000	7699	21.556	217.1
76923	100000	100000	7701	21.561	223.6
71429	100000	100000	7711	21.590	220.5
66667	100000	100000	7710	21.588	222.3
62500	100000	100000	7711	21.590	220.4
58824	100000	100000	7711	21.590	221.0
55556	100000	100000	7711	21.591	221.2
52632	100000	100000	7710	21.588	221.2
50000	100000	100000	7711	21.591	221.1
47619	100000	100000	7711	21.590	221.5
45455	100000	100000	7710	21.587	221.3
43478	100000	100000	7711	21.591	221.2
41667	100000	100000	7711	21.590	222.0
40000	100000	100000	7711	21.590	220.6
38462	100000	100000	7711	21.590	221.5
37037	100000	100000	7711	21.590	221.3
35714	100000	100000	7710	21.588	220.9
34483	100000	100000	7711	21.589	221.6
33333	100000	100000	7711	21.591	221.2
32258	100000	100000	7698	21.553	216.6
31250	100000	100000	7712	21.593	220.9
30303	100000	100000	7711	21.590	220.8
29412	100000	100000	7710	21.588	221.1
28571	100000	100000	7710	21.588	219.4
27778	100000	100000	7700	21.561	222.1
27027	100000	100000	7700	21.559	224.6
26316	100000	100000	7706	21.577	222.3
25641	100000	100000	7706	21.576	220.2
25000	100000	100000	7704	21.572	219.9
24390	100000	100000	7703	21.568	221.3
23810	100000	100000	7701	21.564	223.3
23256	100000	100000	7697	21.551	222.6
22727	100000	100000	7639	21.388	214.4
22222	100000	100000	7643	21.401	215.2
21739	100000	100000	7647	21.413	219.4
21277	100000	100000	7687	21.524	221.2
20833	100000	100000	7691	21.535	221.8
20408	100000	100000	7690	21.533	222.2
20000	100000	100000	7690	21.533	222.0
19608	100000	100000	7686	21.522	221.4
19231	100000	100000	7680	21.503	221.4
18868	100000	100000	7673	21.484	215.6
18519	100000	100000	7669	21.474	208.0
18182	100000	100000	7662	21.454	205.9
17857	100000	100000	7649	21.417	202.6
17544	100000	100000	7616	21.325	197.2
17241	100000	100000	7538	21.107	189.5
16949	100000	100000	7492	20.978	188.3
16667	100000	100000	7524	21.067	186.6

5.4.3 Messwerte mit Datenpaketlänge 700 Bytes

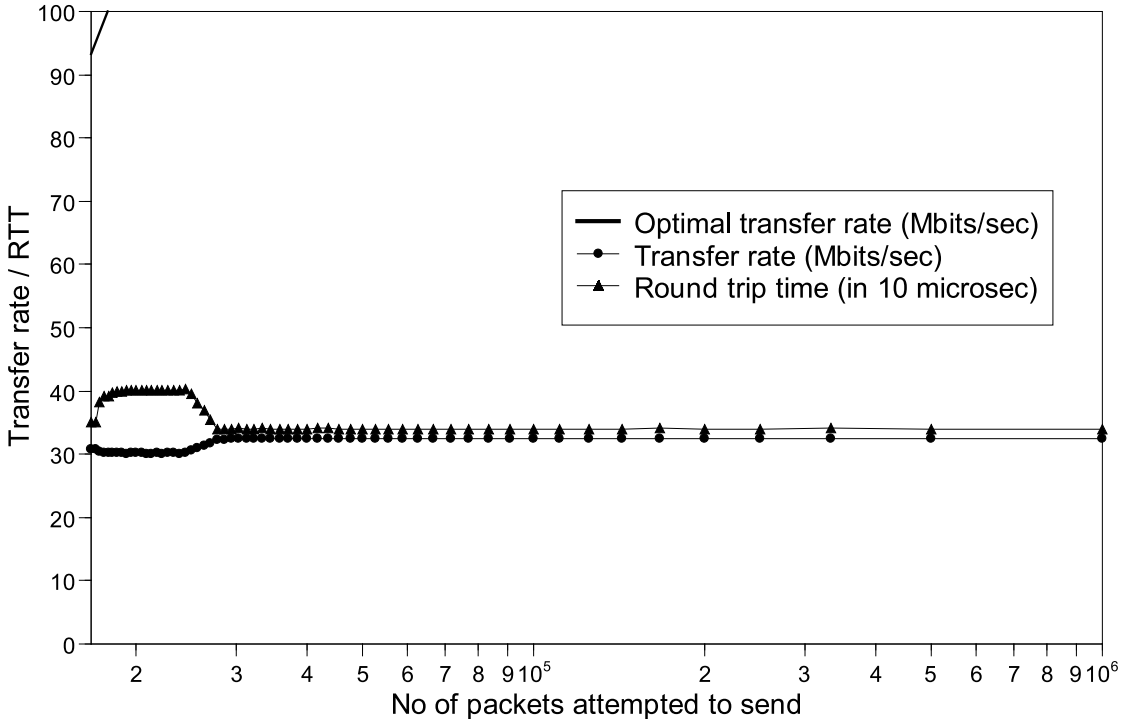


Abbildung 21: Transferrate und RTT (embedded PowerPC, Paketlänge 700 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: Application Referece Board
             CPU: embedded PowerPC 405 133 MHz
          Memory: 64 MBytes
             Bus: D6 DRAM Memory
              OS: Monta Vista Linux (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 700 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	100000	5802	32.489	339.7
500000	100000	100000	5800	32.481	339.9
333333	100000	100000	5798	32.469	340.3
250000	100000	100000	5801	32.486	339.7
200000	100000	100000	5800	32.481	339.8
166667	100000	100000	5797	32.463	340.4
142857	100000	100000	5803	32.499	339.2
125000	100000	100000	5806	32.512	338.9

111111	100000	100000	5800	32.478	339.9
100000	100000	100000	5798	32.469	340.2
90909	100000	100000	5802	32.488	339.6
83333	100000	100000	5801	32.488	339.6
76923	100000	100000	5800	32.482	339.6
71429	100000	100000	5799	32.475	339.9
66667	100000	100000	5803	32.498	339.1
62500	100000	100000	5805	32.509	338.9
58824	100000	100000	5804	32.505	339.2
55556	100000	100000	5800	32.481	339.8
52632	100000	100000	5800	32.481	339.9
50000	100000	100000	5802	32.491	339.6
47619	100000	100000	5800	32.482	339.8
45455	100000	100000	5802	32.489	339.5
43478	100000	100000	5796	32.456	340.6
41667	100000	100000	5795	32.452	340.6
40000	100000	100000	5798	32.469	340.3
38462	100000	100000	5802	32.491	339.7
37037	100000	100000	5800	32.482	339.8
35714	100000	100000	5799	32.474	339.9
34483	100000	100000	5799	32.474	340.0
33333	100000	100000	5797	32.466	340.4
32258	100000	100000	5802	32.494	339.5
31250	100000	100000	5800	32.480	339.7
30303	100000	100000	5796	32.460	340.4
29412	100000	100000	5780	32.370	339.4
28571	100000	100000	5761	32.261	339.6
27778	100000	100000	5749	32.195	339.7
27027	100000	100000	5670	31.751	354.2
26316	100000	100000	5591	31.307	370.0
25641	100000	100000	5536	31.000	380.0
25000	100000	100000	5445	30.495	395.6
24390	100000	100000	5398	30.230	402.1
23810	100000	100000	5377	30.110	401.6
23256	100000	100000	5382	30.137	400.7
22727	100000	100000	5380	30.130	400.8
22222	100000	100000	5376	30.105	401.6
21739	100000	100000	5378	30.116	401.4
21277	100000	100000	5377	30.110	401.7
20833	100000	100000	5376	30.105	401.7
20408	100000	100000	5381	30.132	400.9
20000	100000	100000	5382	30.142	400.6
19608	100000	100000	5383	30.145	400.4
19231	100000	100000	5373	30.089	400.9
18868	100000	100000	5386	30.161	399.9
18519	100000	100000	5386	30.162	399.2
18182	100000	100000	5395	30.210	397.4
17857	100000	100000	5409	30.291	391.6
17544	100000	100000	5396	30.217	391.3
17241	100000	100000	5430	30.407	382.0
16949	100000	100000	5507	30.841	350.4
16667	100000	100000	5499	30.792	350.2

5.4.4 Messwerte mit Datenpaketlänge 1050 Bytes

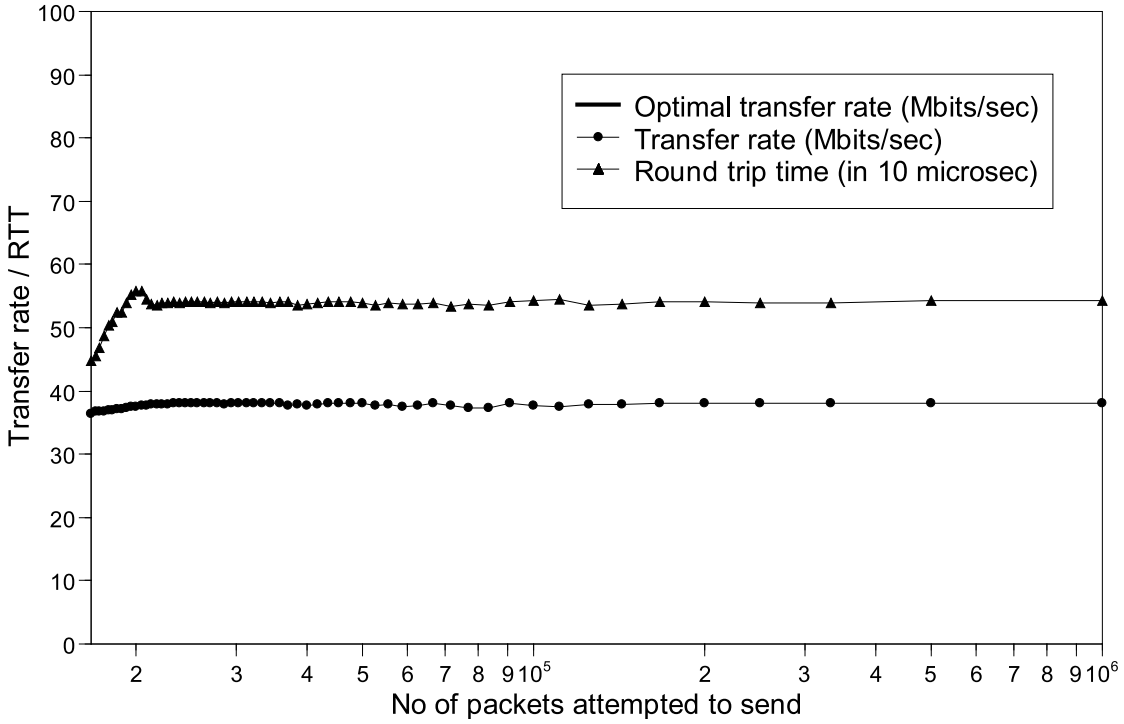


Abbildung 22: Transferrate und RTT (embedded PowerPC, Paketlänge 1050 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: Application Referece Board
             CPU: embedded PowerPC 405 133 MHz
          Memory: 64 MBytes
             Bus: D6 DRAM Memory
              OS: Monta Vista Linux (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 1050 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	98858	4534	38.086	543.0
500000	100000	98857	4535	38.094	543.6
333333	100000	98921	4530	38.055	539.9
250000	100000	99014	4531	38.063	539.6
200000	100000	98894	4533	38.076	540.3
166667	100000	98942	4531	38.064	540.6
142857	100000	99110	4502	37.814	536.9
125000	100000	99111	4509	37.876	535.4

111111	100000	99974	4467	37.523	545.1
100000	100000	99664	4484	37.663	543.6
90909	100000	98959	4531	38.059	540.8
83333	100000	99971	4450	37.382	535.5
76923	100000	99955	4445	37.341	537.7
71429	100000	99691	4478	37.614	533.9
66667	100000	99035	4531	38.061	539.4
62500	100000	99875	4489	37.711	537.5
58824	100000	99997	4464	37.495	537.6
55556	100000	99275	4506	37.848	539.6
52632	100000	99918	4475	37.589	535.3
50000	100000	98966	4531	38.058	539.5
47619	100000	98835	4533	38.078	541.5
45455	100000	98940	4534	38.083	541.5
43478	100000	98889	4532	38.072	541.1
41667	100000	99132	4510	37.883	538.4
40000	100000	99512	4480	37.633	538.0
38462	100000	99334	4502	37.816	535.2
37037	100000	99931	4485	37.672	541.4
35714	100000	98925	4532	38.072	541.5
34483	100000	98996	4532	38.066	540.0
33333	100000	98814	4535	38.090	541.4
32258	100000	98816	4534	38.083	541.6
31250	100000	98792	4532	38.071	541.4
30303	100000	98870	4533	38.074	540.5
29412	100000	98763	4532	38.070	541.8
28571	100000	99135	4508	37.869	539.3
27778	100000	98813	4532	38.068	540.7
27027	100000	98915	4531	38.061	539.5
26316	100000	98871	4533	38.076	541.9
25641	100000	98988	4531	38.057	540.8
25000	100000	98898	4531	38.064	540.5
24390	100000	98875	4533	38.078	540.2
23810	100000	98959	4533	38.073	540.1
23256	100000	99090	4531	38.059	540.8
22727	100000	99448	4518	37.948	538.7
22222	100000	99842	4511	37.894	538.8
21739	100000	100000	4504	37.830	534.9
21277	100000	100000	4497	37.772	538.2
20833	100000	100000	4488	37.697	545.3
20408	100000	100000	4476	37.596	556.9
20000	100000	100000	4468	37.529	558.4
19608	100000	100000	4458	37.446	552.4
19231	100000	100000	4442	37.309	539.2
18868	100000	100000	4428	37.192	524.7
18519	100000	100000	4414	37.075	523.3
18182	100000	100000	4391	36.883	510.0
17857	100000	100000	4388	36.857	503.6
17544	100000	100000	4376	36.760	487.4
17241	100000	100000	4371	36.715	468.0
16949	100000	100000	4364	36.660	454.4
16667	100000	100000	4330	36.370	447.4

5.4.5 Messwerte mit Datenpaketlänge 1460 Bytes

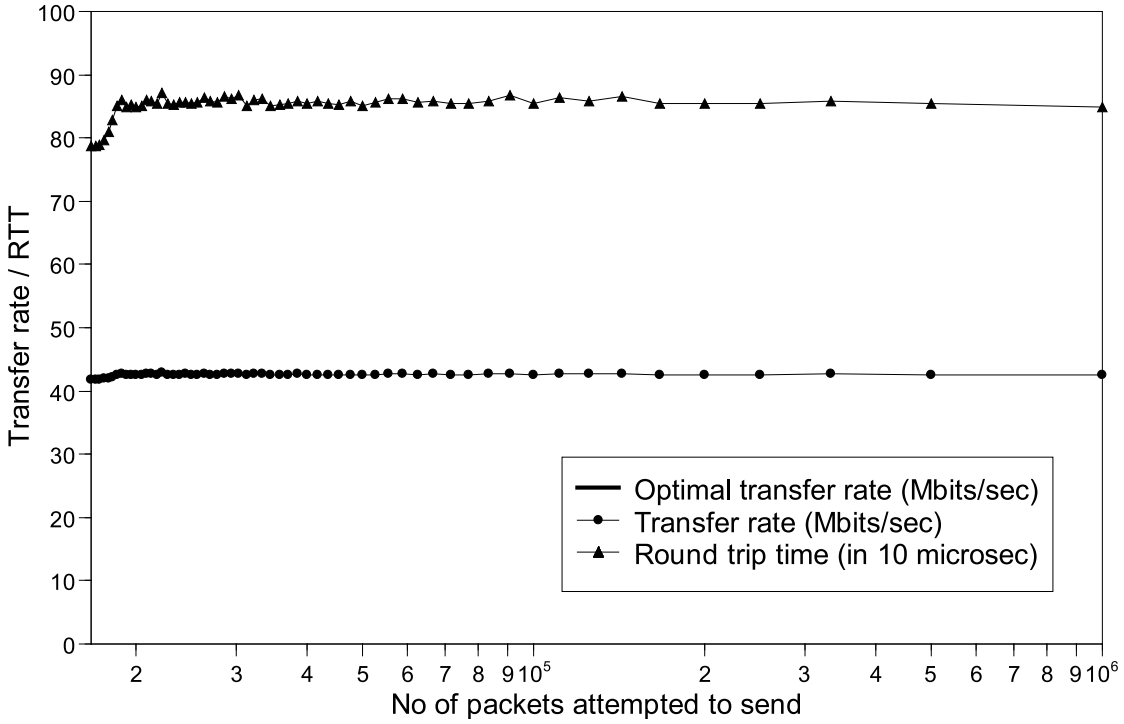


Abbildung 23: Transferrate und RTT (embedded PowerPC, Paketlänge 1460 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: Application Referece Board
             CPU: embedded PowerPC 405 133 MHz
          Memory: 64 MBytes
             Bus: D6 DRAM Memory
              OS: Monta Vista Linux (2.4.17 Kernel)

Networkprocessor: NP4GS3B

    Packetlength: 1460 Bytes
```

No of Pkt attempted to send (pps)	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro-second)
1000000	100000	95508	3638	42.497	849.8
500000	100000	95808	3646	42.583	854.7
333333	100000	96182	3652	42.657	859.0
250000	100000	95742	3644	42.561	853.9
200000	100000	95734	3643	42.550	853.8
166667	100000	95851	3645	42.576	853.7
142857	100000	96465	3659	42.739	865.3
125000	100000	96062	3651	42.645	859.1

111111	100000	96312	3658	42.720	863.1
100000	100000	95864	3646	42.588	854.9
90909	100000	96608	3663	42.784	868.0
83333	100000	95954	3650	42.630	857.8
76923	100000	95720	3646	42.585	855.1
71429	100000	95814	3647	42.599	855.5
66667	100000	96174	3653	42.662	859.2
62500	100000	96010	3648	42.614	857.0
58824	100000	96210	3654	42.677	861.4
55556	100000	96346	3656	42.706	861.6
52632	100000	95886	3648	42.604	857.1
50000	100000	95555	3638	42.492	851.6
47619	100000	95893	3647	42.597	858.7
45455	100000	95837	3644	42.563	853.1
43478	100000	95778	3645	42.571	855.0
41667	100000	95998	3648	42.612	858.4
40000	100000	95831	3645	42.573	855.5
38462	100000	96075	3651	42.646	858.9
37037	100000	95783	3645	42.579	854.6
35714	100000	95784	3643	42.555	852.7
34483	100000	95541	3640	42.515	850.2
33333	100000	96208	3654	42.674	861.9
32258	100000	96238	3654	42.681	860.3
31250	100000	95536	3641	42.523	851.0
30303	100000	96527	3660	42.749	867.3
29412	100000	96349	3656	42.702	862.6
28571	100000	96373	3659	42.734	865.1
27778	100000	95920	3649	42.615	857.4
27027	100000	95989	3648	42.608	857.5
26316	100000	96405	3657	42.708	863.7
25641	100000	95866	3647	42.595	856.0
25000	100000	95756	3644	42.558	853.9
24390	100000	95901	3649	42.616	857.2
23810	100000	95897	3648	42.608	856.7
23256	100000	95769	3645	42.569	853.2
22727	100000	95792	3644	42.567	855.3
22222	100000	96832	3667	42.831	871.1
21739	100000	95858	3647	42.591	855.3
21277	100000	96082	3652	42.661	858.9
20833	100000	95947	3651	42.639	859.4
20408	100000	96099	3641	42.525	851.8
20000	100000	95997	3637	42.482	849.1
19608	100000	96382	3644	42.556	852.9
19231	100000	96467	3636	42.471	849.5
18868	100000	96171	3655	42.691	861.0
18519	100000	96317	3640	42.512	851.5
18182	100000	95434	3603	42.087	827.8
17857	100000	95917	3589	41.923	810.1
17544	100000	96616	3585	41.877	796.2
17241	100000	96379	3577	41.779	788.4
16949	100000	96669	3572	41.723	787.7
16667	100000	96876	3574	41.741	786.6

5.5 Beurteilung der Resultate

5.5.1 Ethernet-Attached External Control Point

Die Messungen mit einem Ethernet-Attached External Control Point auf dem Dell Precision 340 zeigen einen maximalen Durchsatz von 373.3 Mbit pro Sekunde. Dieser Maximalwert ist durch die Begrenzung des PCI Busses zu erklären (wurde bereits in Abschnitt 3.5 erläutert). Der maximale Durchsatz von 959.5 MBit pro Sekunde wird nur erreicht, wenn ein genug schneller PCI Bus zum Einsatz kommt. Beim Test auf dem xSerie 335 Server ist nicht mehr der PCI Bus die begrenzende Komponente, sondern die Data Mover Units und das Gigabit-Ethernet an sich. Interessant ist auch die Feststellung, dass der IBM xSerie 335 Server generell die besseren Messwerte hat als der Dell Precision. Die Latenzzeiten sind kürzer und der maximale Durchsatz ist höher. Es ist anzunehmen, dass diese besseren Resultate auf den schnelleren PCI Bus, aber auch auf die grössere Performance der CPU zurückzuführen ist.
Bei einem zweiten Testdurchgang auf dem xSerie 335 Server wurde das Application Reference Board mit einem zusätzlichen Cross Traffic von 1 GBits/sec belastet. D.h. auf den dritten und noch freien Gigabit-Ethernet Port wurden zusätzliche IP Datenpakete gesendet, die von Netzwerkprozessor weitergeleitet werden mussten. Die Datenpakete waren so konzipiert, dass sie durch das Layer-3 Forwarding auf demselben Port wieder gesendet wurden. Beim Vergleich

System	Precision 340	xSerie 335 Server	embedded PPC
Hersteller	Dell	IBM	IBM
CPU	Pentium 4 1.8 GHz	Xenon 2.4 GHz	PPC 405 133 MHz
Bus	PCI 32-bit/33MHz	PCI-X 64bit/100MHz	
Maximaler Durchsatz (Paketlänge = 1460)	373.3 Mbit/sec	959.5 Mbit/sec	42.7 Mbit/sec
Maximaler Durchsatz (Paketlänge = 1050)	358.8 Mbit/sec	944.7 Mbit/sec	38.1 Mbit/sec
Maximaler Durchsatz (Paketlänge = 700)	336.4 Mbit/sec	919.4 Mbit/sec	32.5 Mbit/sec
Maximaler Durchsatz (Paketlänge = 350)	283.4 Mbit/sec	599.3 Mbit/sec	21.6 Mbit/sec
Maximaler Durchsatz (Paketlänge = 72)	115.3 Mbit/sec	171.6 Mbit/sec	5.7 Mbit/sec
Maximaler Durchsatz (Paketlänge = 1460)	31964 pps	82150 pps	3663 pps
Maximaler Durchsatz (Paketlänge = 1050)	42712 pps	112465 pps	4534 pps
Maximaler Durchsatz (Paketlänge = 700)	60073 pps	164177 pps	5805 pps
Maximaler Durchsatz (Paketlänge = 350)	101205 pps	214004 pps	7712 pps
Maximaler Durchsatz (Paketlänge = 72)	200170 pps	297985 pps	9807 pps
Minimale RTT (Paketlänge = 1460)	152.3 microsec	96.2 microsec	786.6 microsec
Minimale RTT (Paketlänge = 1050)	117.4 microsec	88.0 microsec	447.4 microsec
Minimale RTT (Paketlänge = 700)	113.2 microsec	76.5 microsec	338.9 microsec
Minimale RTT (Paketlänge = 350)	96.3 microsec	67.1 microsec	186.6 microsec
Minimale RTT (Paketlänge = 72)	70.5 microsec	48.7 microsec	124.3 microsec

Tabelle 7: Gesamtübersicht der Messresultate

der Testresultate mit und ohne Cross Traffic konnten keine Abweichungen festgestellt werden. Daraus lässt sich der Schluss ziehen, dass der EPC mit einem PromethOS Traffic von 959.5 MBis/sec noch nicht ausgelastet ist.

5.5.2 Embedded PowerPC Processor Internal Control Point

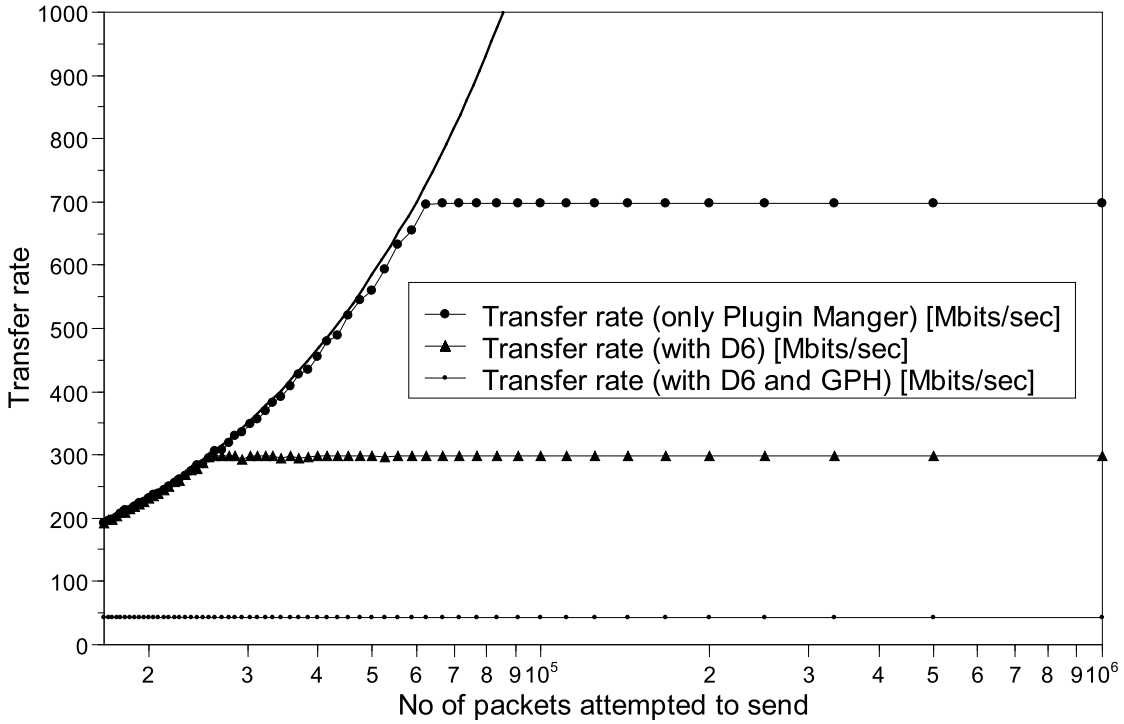


Abbildung 24: Transferrate (embedded PowerPC, Paketlänge 1460 Bytes)

Die Messwerte mit dem embedded PowerPC sind relativ ernüchternd ausgefallen. Ein maximaler Durchsatz von 42.7 Mbit/sec ist für den Einsatz von PromethOS in dieser Architektur kaum brauchbar. Um die tiefen Messwerte besser zu verstehen und eventuelle auch ein Begründung für die tiefen Werte zu finden, wurden weitere Messungen mit dem embedded PowerPC durchgeführt.
Um den Sinn der nachfolgend erklärten Messungen zu verstehen, muss man sich vor Augen führen, wie die Schnittstelle zwischen dem embedded PowerPC und dem GPH konzipiert ist. Will der embedded PowerPC ein Datenpaket senden, so kopiert er den Inhalt des Socket Buffers in einem definierten Format an eine bestimmte Stelle im Speicher (D6 DRAM). Über einen Interrupt signalisiert der PowerPC dem GPH, dass ein Datenpaket im Speicher bereitgestellt wurde und an welcher Speicheradresse es beginnt. Mit derselben Technik werden auch die Datenpakete empfangen. Ein detailliertere Beschreibung der Schnittstelle ist in [11, Mailbox Communications Between PowerPC Subsystem and EPC] nachzulesen.
Das Ziel dieser Messungen war herauszufinden, welcher Vorgang das System wie stark belastet. Dazu wurde das in Abschnitt 5.2.3 erläuterte Messverfahren in zwei zusätzlichen Varianten durchgeführt. Es wurde ebenfalls der Plugin Manager Simulator verwendet. In der ersten Variante der Messung werden die Datenpakete von Plugin Manager Simulator erstellt und ein entsprechender Socket Buffer in den Speicher geschrieben. Anstatt dass anschliessend das Datenpaket mit der Funktion `promethos_send_raw()` gesendet wird, ruft der Plugin Manager Simulator seine eigene Empfangsfunktion auf. Mit dieser Variante wird gemessen, wie viele Socket Buffers (gemessen in Bytes) pro Sekunde erstellt werden können.
In einem zweiten Verfahren wird der Socket Buffer in den gemeinsamen Speicherbereich zwischen embedded PowerPC und GPH kopiert und der Socket Buffer wird wieder freigegeben. Danach wird sofort wieder ein Socket Buffer angefordert und das Datenpaket vom gemeinsamen Speicherbereich in den neu erstellten Socket Buffer kopiert. Diese Messung hat zum Ziel,

zu evaluieren, wie aufwendig das Kopieren von Socket Buffers in den gemeinsamen Speicherbereich ist. Im Unterschied zum Original Messverfahren wird hier der GPH übersprungen.
In der Abbildung 24 sind die Resultate der Originalmessung aus Abschnitt 5.4.5 sowie die Resultate beider abgeänderten Varianten ersichtlich. Alle drei Messungen wurden mit einer Datenpaketlänge von 1460 Bytes durchgeführt. Die erste zusätzliche Messung hat ergeben, dass der embedded PowerPC nicht mehr als 697.39 MBits/sec an Socket Buffers erstellen kann. Das bedeutet, wie gut die Schnittstelle zwischen embedded PowerPC und GPH auch noch optimiert werden kann, eine höhere Transferrate als 697.39 Mbit/sec ist nicht zu erreichen. Man muss davon ausgehen, dass der embedded PowerPC mit seiner doch etwas tiefen Taktfrequenz nicht zu mehr in der Lage ist und sämtliche zur Verfügung stehenden CPU Zeit für den Overhead des Linux Betriebsystems und für das Erstellen der Socket Buffer verwendet.
Die zweite Messung, bei der die Schnittstelle mit dem gemeinsamen Speicherbereich zur Hälfte simuliert wird, hat eine maximale Transferrate von 298.04 MBit/sec ergeben. Zusätzlich zur ersten Messung wird hier der Socket Buffer zweimal kopiert. Vergleicht man die beiden Resultate, muss festgestellt werden, dass das Kopieren des Socket Buffers eine sehr teure Operation ist und die CPU stark belastet.
Als dritte Kurve ist die Originalmessung abgebildet. Mit 42.7 Mbit/sec ist sie noch um einiges tiefer als die zweite Variante ausgefallen. Der Unterschied zu den 298.04 MBit/sec lässt sich durch den Aufwand des GPH und durch den Overhead der Interrupt Behandlung erklären. Man muss sich auch die Frage stellen, wie viele Interrupts pro Sekunde ausgelöste werden können und ob diese Anzahl maximaler Interrupts pro Zeiteinheit einen «Flaschenhals» für die Kommunikation zwischen dem embedded PowerPC und dem GPH darstellt.

6 Pico Engine Performance Messung

6.1 Messverfahren

Um einen Anhaltspunkt für die Leistungsfähigkeit von Pico Plugins zu bekommen, wurden diese zusätzlichen Messungen durchgeführt. Das Messverfahren ist mit dem in Abschnitt 5.2.3 beschriebenen Verfahren identisch. Als Traffic Generator wird ebenfalls der Plugin Manager Simulator verwendet. Im Gegensatz zu den bisherigen Messungen, wo nur die ersten 64 Bytes des Datenpaketes in den EPC geladen werden, wird diesmal das gesamte Datenpaket in den ECP kopiert und wieder zurück in den Ingress Data Store geschrieben. Dies geschieht jeweils in Blöcken zu 64 Bytes. Um die Rechenleistung des EPCs zu testen, wird zusätzlich eine Iteration durchgeführt, die aus einer Vergleichsoperation und einer Subtraktion besteht. Diese Messverfahren soll über zwei Werte Aufschluss geben:

- Wie hoch ist der Durchsatz, wenn der EPC ganze Datenpakete verarbeiten muss?
- Ab wievielen Iterationen pro Byte ist ein Performance Einbruch des Durchsatzes feststellbar?

Um diese Messung durchzuführen wurde, der Picocode des Classifiers erweitert. Diese Erweiterung (siehe Auszug aus l4.asm) stellt das Pico Plugin im Messverfahren dar. Der Variable Wert bei der Messung ist die Anzahl Iteration (gleich zusätzliche Instruktionen), die pro Byte eines Datenpaketes ausgeführt werden.

```
        .
        .
        .
        ldr   r26,  DATAPOOL[0x10]  ; get IP Packetlength
        add   r26,#18
        slr   r26,#6

; ------ copy packet from ingress data store to EPC (and reverse) ----------
l4_promethos_loop       EQU $
        cmp r26, #0
        be l4_promethos_next
        xor   w28,  w28             ; 00 = 4 quadwords
        mov NQWA,#4                 ; set destination for rdmorei
        rdmorei w28,  #0            ; copy 64 bytes from Ingress EDS to EPC
        wait COP_ds

        ldr   w28,  CCTA            ; Get Original Current Buffer
        str   DSA,  w28             ; Set Destination
        mov   LMA,  #16             ; From Datapool to DataStore
        xor   w28,  w28             ; 00 = 4 quadwords
        wrids w28,  #0              ; update frame with DATAPOOL changes
        wait COP_ds

        sub r26,#1
        b l4_promethos_loop
l4_promethos_next       EQU $

; ------ Iteration ---------------------------------------------------------
        ldr r28, DATAPOOL[0x3E]     ; load number of iterations
l4_promethos_loop2      EQU $
        cmp r28, #0                 ; unitl (r28 == 0)
        be l4_promethos_next2
        sub r28,#1                  ; count down
        b l4_promethos_loop2

l4_promethos_next2      EQU $
        .
        .
        .
```

(Auszug aus ./aso-134/src/npdd/pico/src/data/l4.asm)

6.2 Messwerte

Die Messungen wurden auf einem System mit einem PCI-X 64-bit/100MHz Bus durchgeführt. Der PCI Bus stellt somit keine Einschränk für den Maximalwert dar. In den folgenden Auswertungsdiagrammen wird auf der X-Achse die Anzahl zusätzlicher Iterationen pro Byte abgebildet. Durch diese Iterationen werden zusätzliche Operationen auf dem Netzwerkprozessor ausgeführt, die den Aufwand für ein Pico Plugin darstellen. Der Graph «cost per packet» wurde berechnet und ist proportional zum Reziprokwert der «Transfer rate». Er stellt dar, wie stark die Kosten pro zusätzlicher Iteration steigen und ist kein Absolutwert für die Kosten pro Paket.

$$\frac{1}{transfer\_rate} * const \sim cost\_per\_packet \tag{1}$$

6.2.1 Messwerte mit Datenpaketlänge 72 Bytes

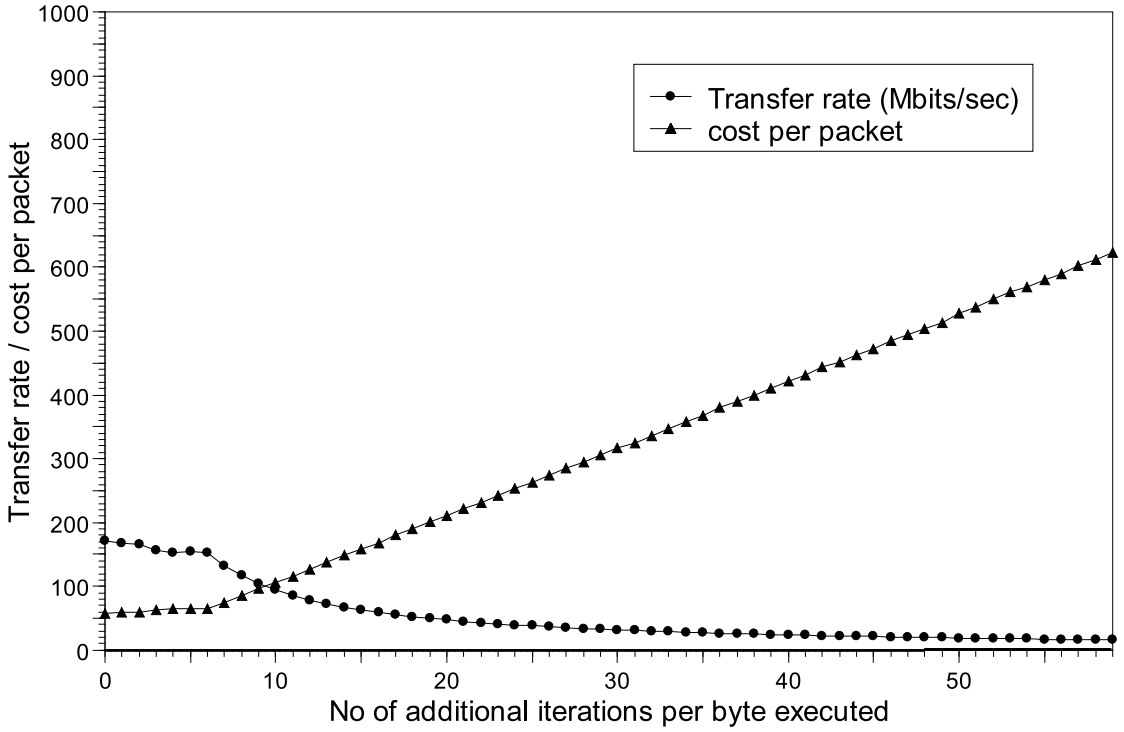


Abbildung 25: Pico Engine performance test (Paketlänge 72 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 72 Bytes
```

No of a. Iterations per packet executeted	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
0	100000	100000	297271	171.228	83.8
1	100000	100000	290940	167.582	109.3
2	100000	100000	289142	166.546	145.2
3	100000	100000	272572	157.002	181.3
4	100000	100000	265415	152.879	184.0
5	100000	100000	269402	155.175	199.2
6	100000	100000	265834	153.121	1504.9
7	100000	72357	230765	132.921	4288.3

8	100000	57519	203196	117.041	4869.6
9	100000	47813	180669	104.065	5317.9
10	100000	40645	164042	94.488	5820.2
11	100000	35118	149340	86.020	6381.1
12	100000	31122	136819	78.808	6703.4
13	100000	27965	126597	72.920	7915.1
14	100000	25367	116272	66.973	7649.2
15	100000	23149	109692	63.183	9320.0
16	100000	21443	103270	59.483	9476.8
17	100000	19806	96558	55.617	9770.7
18	100000	18599	91533	52.723	9636.9
19	100000	17561	86653	49.912	11014.0
20	100000	16644	82385	47.454	11487.8
21	100000	15716	78492	45.212	11596.5
22	100000	14970	75207	43.319	11505.0
23	100000	14125	71647	41.269	12365.6
24	100000	13634	68665	39.551	12670.4
25	100000	13102	66110	38.079	13292.1
26	100000	12393	63476	36.562	13791.2
27	100000	11940	60858	35.054	13384.4
28	100000	11438	58998	33.983	14679.6
29	100000	10963	56912	32.781	15950.6
30	100000	10636	54723	31.520	15616.2
31	100000	10295	53361	30.736	17417.6
32	100000	9946	51673	29.763	18231.6
33	100000	9601	50130	28.875	16309.0
34	100000	9343	48487	27.929	17034.2
35	100000	9064	47230	27.204	18899.3
36	100000	8778	45735	26.343	20092.5
37	100000	8581	44625	25.704	20055.8
38	100000	8364	43407	25.003	20361.2
39	100000	8229	42254	24.338	19899.6
40	100000	7976	41218	23.742	20013.9
41	100000	7770	40366	23.251	20077.1
42	100000	7575	39137	22.543	21422.8
43	100000	7425	38384	22.109	22377.7
44	100000	7235	37547	21.627	21572.4
45	100000	7072	36722	21.152	22589.2
46	100000	6936	35823	20.634	21490.7
47	100000	6852	35095	20.215	20988.0
48	100000	6744	34413	19.822	23619.9
49	100000	6582	33791	19.463	24394.9
50	100000	6417	32926	18.965	27375.0
51	100000	6329	32299	18.604	25968.2
52	100000	6171	31537	18.165	31511.9
53	100000	6046	30918	17.809	26916.7
54	100000	5974	30527	17.583	29331.6
55	100000	5939	29908	17.227	29706.5
56	100000	5850	29476	16.978	29368.0
57	100000	5712	28828	16.605	31369.0
58	100000	5658	28340	16.324	29784.2
59	100000	5557	27824	16.027	32046.6

6.2.2 Messwerte mit Datenpaketlänge 350 Bytes

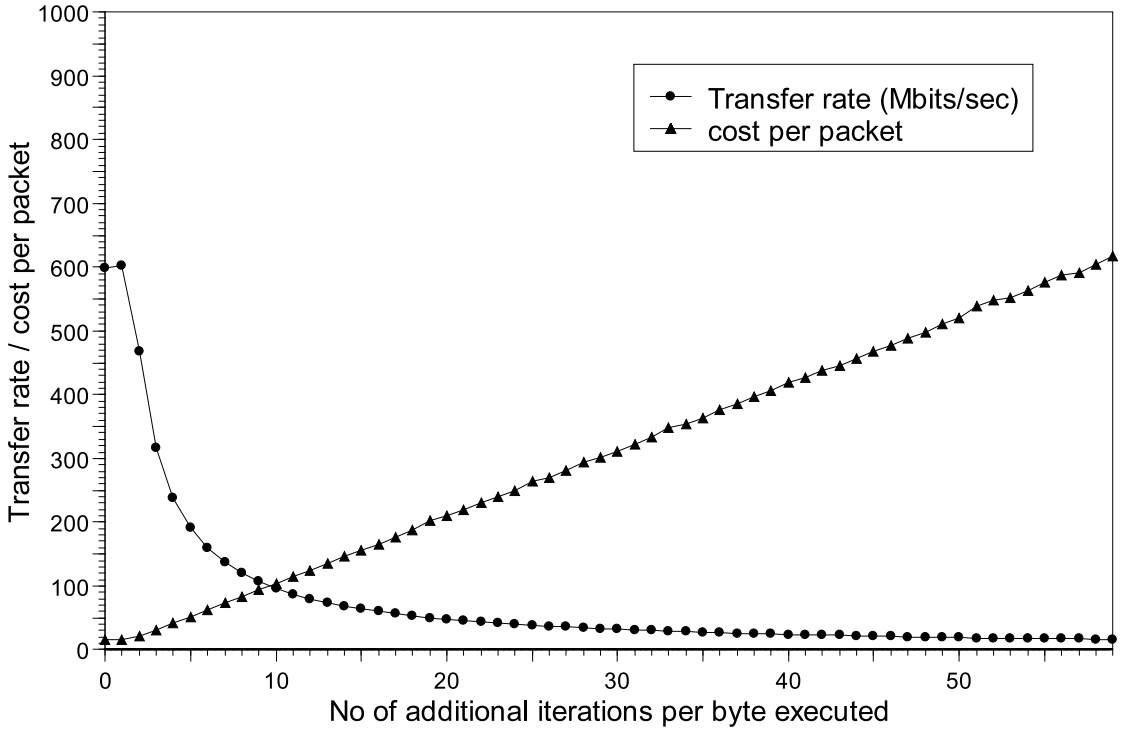


Abbildung 26: Pico Engine performance test (Paketlänge 350 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 350 Bytes
```

No of a. Iterations per packet executeted	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
0	100000	100000	213847	598.770	127.3
1	100000	100000	214833	601.533	202.3
2	100000	58346	167414	468.760	1988.0
3	100000	31536	113330	317.323	2945.1
4	100000	20846	84803	237.447	3939.8
5	100000	15547	68367	191.428	5125.0
6	100000	11980	56896	159.309	6198.5
7	100000	8471	49003	137.208	5922.1

8	100000	7243	42784	119.795	7651.2
9	100000	6300	38089	106.649	9423.2
10	100000	5599	34302	96.047	11403.8
11	100000	5053	31051	86.943	10442.3
12	100000	4604	28554	79.952	12887.2
13	100000	4239	26481	74.146	13581.8
14	100000	3921	24506	68.617	15927.1
15	100000	3669	22909	64.145	19301.0
16	100000	3415	21514	60.238	19758.7
17	100000	3246	20291	56.815	19441.0
18	100000	3053	19106	53.497	22058.6
19	100000	2890	17636	49.381	20825.5
20	100000	2746	16981	47.548	22278.5
21	100000	2623	16300	45.639	24480.4
22	100000	2507	15545	43.526	24123.9
23	100000	2404	14881	41.667	25501.9
24	100000	2309	14290	40.012	29836.7
25	100000	2193	13561	37.971	31755.3
26	100000	2157	13219	37.013	33101.4
27	100000	2074	12741	35.675	33728.6
28	100000	1998	12171	34.077	29828.3
29	100000	1970	11851	33.184	26450.5
30	100000	1921	11496	32.189	32162.2
31	100000	1838	11073	31.005	33786.6
32	100000	1781	10726	30.032	30662.8
33	100000	1732	10279	28.782	28436.3
34	100000	1685	10075	28.211	39904.3
35	100000	1671	9813	27.476	42375.2
36	100000	1611	9487	26.563	43752.3
37	100000	1589	9277	25.974	45540.7
38	100000	1535	9008	25.223	45442.9
39	100000	1526	8783	24.592	47448.9
40	100000	1468	8505	23.814	46023.9
41	100000	1443	8364	23.418	43584.7
42	100000	1413	8167	22.868	48818.9
43	100000	1397	8002	22.406	48895.8
44	100000	1364	7835	21.938	51250.5
45	100000	1345	7625	21.349	46733.1
46	100000	1314	7480	20.945	47139.0
47	100000	1307	7302	20.446	50142.2
48	100000	1288	7170	20.077	49005.5
49	100000	1253	7004	19.611	50975.2
50	100000	1227	6868	19.231	55647.1
51	100000	1202	6634	18.575	54613.0
52	100000	1190	6525	18.270	59800.8
53	100000	1177	6470	18.115	59361.8
54	100000	1160	6351	17.783	62743.3
55	100000	1129	6201	17.363	61735.5
56	100000	1119	6079	17.020	63732.0
57	100000	1143	6041	16.915	56976.9
58	100000	1087	5911	16.550	66436.2
59	100000	1079	5789	16.209	70487.8

6.2.3 Messwerte mit Datenpaketlänge 700 Bytes

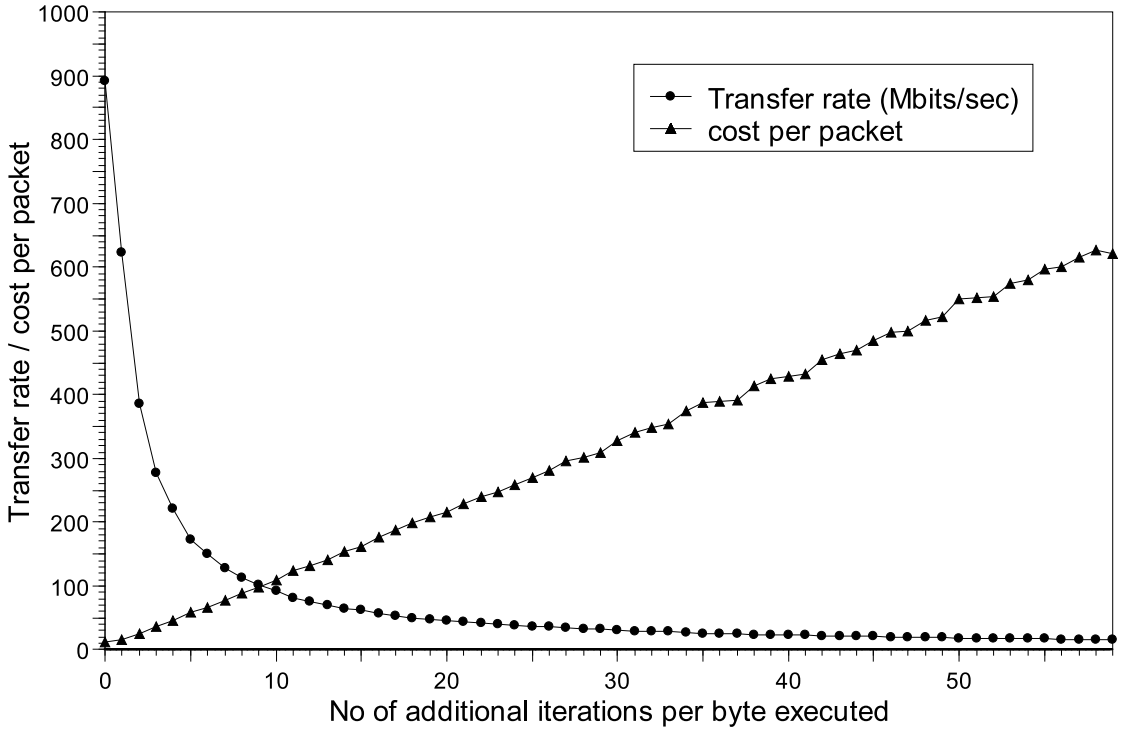


Abbildung 27: Pico Engine performance test (Paketlänge 700 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 700 Bytes
```

No of a. Iterations per packet executeted	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
0	100000	48626	159200	891.519	
1	100000	24218	111158	622.485	2615.3
2	100000	11695	68928	385.999	85704.3
3	100000	7705	49478	277.075	
4	100000	5954	39454	220.944	5363.5
5	100000	4494	30781	172.371	
6	100000	3871	26835	150.276	7736.5
7	100000	3272	22968	128.621	8689.2

8	100000	2894	20282	113.578	9897.1
9	100000	2558	18170	101.754	10368.9
10	100000	2357	16476	92.265	12136.4
11	100000	2030	14369	80.465	
12	100000	1943	13491	75.547	14343.9
13	100000	1803	12659	70.888	14558.3
14	100000	1725	11634	65.149	18647.6
15	100000	1589	11031	61.771	16929.9
16	100000	1495	10173	56.968	19848.8
17	100000	1386	9558	53.523	19230.2
18	100000	1308	8939	50.060	20523.1
19	100000	1295	8610	48.218	23751.9
20	100000	1248	8285	46.396	22823.7
21	100000	1172	7835	43.875	24928.2
22	100000	1111	7446	41.696	26382.6
23	100000	1089	7191	40.271	26595.0
24	100000	1025	6882	38.539	28553.4
25	100000	1022	6605	36.990	30878.0
26	100000	960	6371	35.676	28844.4
27	100000	927	6034	33.790	32140.2
28	100000	949	5930	33.208	31292.3
29	100000	907	5777	32.349	34041.0
30	100000	882	5461	30.584	34912.0
31	100000	820	5236	29.323	36586.5
32	100000	844	5136	28.760	39242.3
33	100000	811	5046	28.259	36447.5
34	100000	779	4779	26.760	40921.3
35	100000	751	4608	25.804	40091.7
36	100000	740	4580	25.650	42099.4
37	100000	748	4570	25.592	41961.3
38	100000	768	4318	24.182	45638.6
39	100000	693	4207	23.556	45878.2
40	100000	681	4162	23.305	48051.7
41	100000	688	4124	23.097	46112.3
42	100000	657	3919	21.947	48536.6
43	100000	640	3851	21.567	47073.5
44	100000	672	3807	21.320	52376.0
45	100000	640	3682	20.622	51833.8
46	100000	647	3587	20.086	52692.8
47	100000	613	3571	19.996	55262.2
48	100000	575	3458	19.365	52392.9
49	100000	603	3416	19.130	57693.9
50	100000	567	3247	18.183	56344.0
51	100000	567	3241	18.148	57700.9
52	100000	572	3224	18.052	62238.5
53	100000	583	3114	17.437	62729.2
54	100000	561	3078	17.237	61724.7
55	100000	536	2994	16.765	63481.5
56	100000	524	2978	16.677	65247.0
57	100000	544	2902	16.253	70013.3
58	100000	519	2846	15.937	65090.1
59	100000	538	2873	16.090	67562.3

6.2.4 Messwerte mit Datenpaketlänge 1050 Bytes

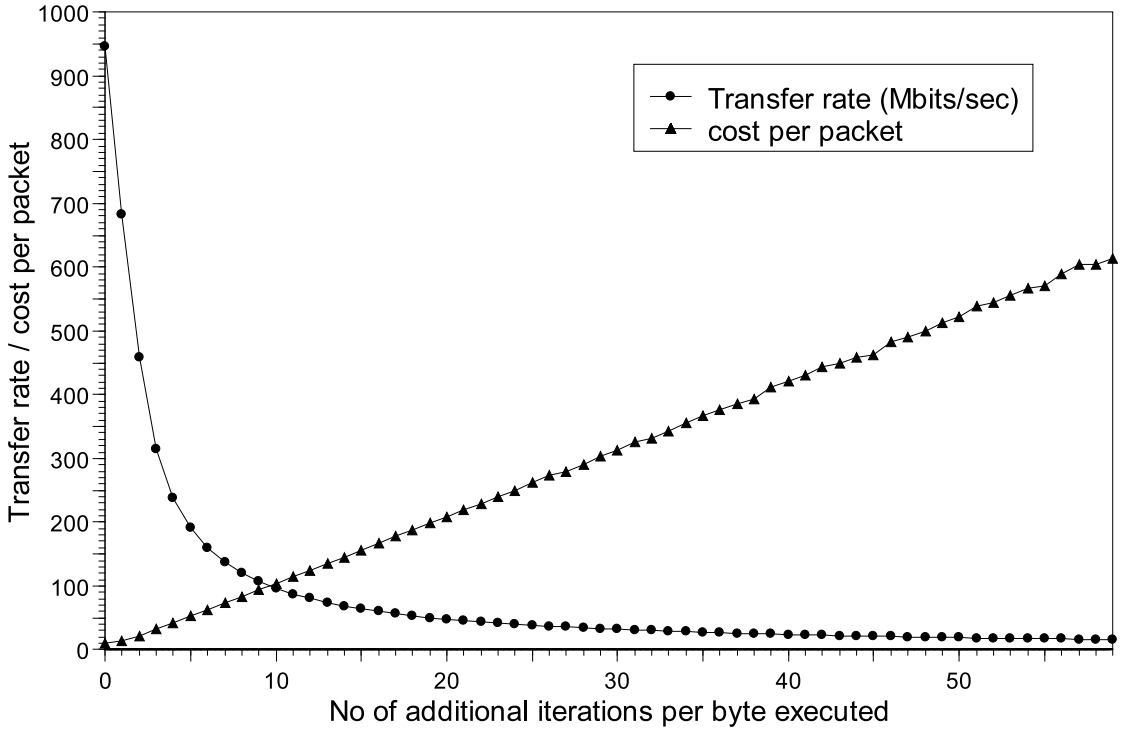


Abbildung 28: Pico Engine performance test (Paketlänge 1050 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 1050 Bytes
```

No of a. Iterations per packet executeted	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
0	100000	25314	112542	945.354	1609.3
1	100000	15210	81313	683.029	3236.3
2	100000	8871	54694	459.433	3894.8
3	100000	5590	37404	314.190	5093.4
4	100000	4007	28248	237.286	6227.3
5	100000	3182	22773	191.292	7566.4
6	100000	2625	19028	159.837	8717.7
7	100000	2243	16312	137.017	9859.0

8	100000	1946	14294	120.071	11685.4
9	100000	1739	12689	106.589	12377.5
10	100000	1554	11473	96.374	13438.9
11	100000	1436	10324	86.722	15165.1
12	100000	1360	9595	80.595	16116.9
13	100000	1236	8739	73.404	17802.4
14	100000	1130	8177	68.687	18277.7
15	100000	1073	7651	64.272	19353.6
16	100000	1012	7154	60.095	21146.9
17	100000	952	6685	56.153	22450.2
18	100000	898	6325	53.130	23147.5
19	100000	861	5995	50.356	24257.0
20	100000	820	5709	47.959	25838.1
21	100000	804	5421	45.538	27630.9
22	100000	758	5192	43.613	28022.7
23	100000	732	4966	41.710	29066.7
24	100000	702	4775	40.106	29964.9
25	100000	701	4536	38.099	31655.5
26	100000	647	4351	36.549	33276.5
27	100000	644	4251	35.710	33136.9
28	100000	620	4098	34.425	34949.0
29	100000	598	3914	32.879	35321.2
30	100000	604	3804	31.951	37913.8
31	100000	572	3651	30.671	38519.7
32	100000	558	3600	30.236	39195.4
33	100000	537	3471	29.153	39660.6
34	100000	519	3343	28.084	41321.5
35	100000	520	3252	27.317	43844.8
36	100000	526	3168	26.613	44683.9
37	100000	492	3089	25.945	44843.0
38	100000	485	3024	25.403	45593.0
39	100000	466	2892	24.291	48467.9
40	100000	482	2829	23.761	49434.1
41	100000	446	2771	23.275	50700.3
42	100000	465	2687	22.571	53776.3
43	100000	437	2648	22.240	50906.3
44	100000	441	2599	21.835	53319.9
45	100000	447	2574	21.624	53790.6
46	100000	399	2467	20.723	54136.1
47	100000	416	2432	20.426	54873.0
48	100000	407	2379	19.982	58200.4
49	100000	402	2323	19.511	59174.0
50	100000	392	2277	19.127	60412.0
51	100000	384	2211	18.576	60876.6
52	100000	370	2190	18.398	60018.0
53	100000	380	2143	18.000	62951.8
54	100000	378	2102	17.661	65678.3
55	100000	383	2087	17.530	64644.2
56	100000	376	2017	16.943	66254.7
57	100000	373	1972	16.567	68853.9
58	100000	348	1971	16.558	65888.9
59	100000	347	1941	16.305	68944.0

6.2.5 Messwerte mit Datenpaketlänge 1460 Bytes

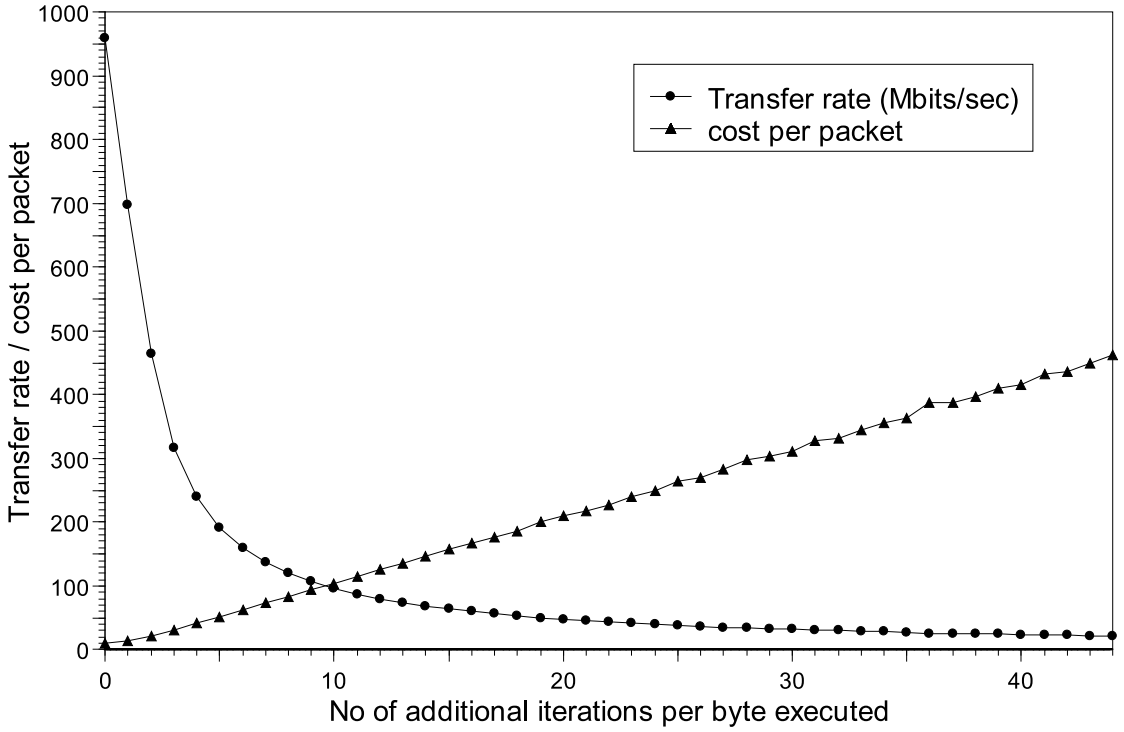


Abbildung 29: Pico Engine performance test (Paketlänge 1460 Bytes)

```
Measurement Report:
-------------------

NPCP / Plugin Manager Simulator running on

          System: IBM xSeries 335 server
             CPU: Intel(R) XEON(TM) CPU 2.40GHz
          Memory: 1024 MBytes
             Bus: PCI-X 64-bit/100MHz
              OS: Red Hat 8.0 (2.4.18 Kernel)

Networkprocessor: NP4GS3C

    Packetlength: 1460 Bytes
```

No of a. Iterations per packet executeted	Total of Pkt sent per probe	Total of Pkt recv. per probe	Transfer rate (pps)	Transfer rate (Mbit/sec)	averg. RTT (micro- second)
0	100000	16106	82163	959.664	1858.7
1	100000	10257	59680	697.061	3374.9
2	100000	6077	39687	463.548	4276.1
3	100000	3894	27178	317.440	5441.7
4	100000	2867	20470	239.086	6683.9
5	100000	2252	16421	191.792	7832.1
6	100000	1873	13728	160.345	9154.5
7	100000	1616	11730	137.010	10169.7

8	100000	1401	10257	119.801	11183.7
9	100000	1246	9127	106.602	12695.2
10	100000	1114	8243	96.280	13466.3
11	100000	1029	7461	87.145	15599.0
12	100000	937	6813	79.575	16553.1
13	100000	887	6342	74.074	17048.1
14	100000	812	5805	67.803	18995.0
15	100000	769	5454	63.700	19790.6
16	100000	731	5138	60.015	21010.4
17	100000	695	4868	56.854	21415.7
18	100000	659	4602	53.747	23054.3
19	100000	599	4263	49.791	25005.4
20	100000	589	4062	47.444	24896.0
21	100000	581	3930	45.903	27101.3
22	100000	539	3759	43.905	26793.3
23	100000	522	3562	41.606	27930.5
24	100000	503	3426	40.017	30055.7
25	100000	484	3240	37.842	31606.9
26	100000	470	3181	37.153	30720.7
27	100000	467	3024	35.320	33375.3
28	100000	456	2878	33.613	34655.9
29	100000	428	2818	32.919	35304.5
30	100000	434	2752	32.147	35220.5
31	100000	419	2618	30.574	37984.1
32	100000	385	2586	30.205	35994.5
33	100000	403	2489	29.069	39626.4
34	100000	386	2403	28.070	41101.5
35	100000	372	2360	27.568	40559.9
36	100000	344	2208	25.787	42102.9
37	100000	359	2213	25.844	43363.1
38	100000	366	2157	25.199	44293.5
39	100000	332	2085	24.356	45174.5
40	100000	348	2058	24.033	47771.2
41	100000	335	1977	23.089	48722.5
42	100000	334	1962	22.913	49981.9
43	100000	318	1909	22.301	48692.2
44	100000	321	1851	21.623	54020.0

6.3 Beurteilung der Resultate

Die Messresultate zeigen auf, dass es möglich ist, mit Linespeed die Datenpackete in den EPC zu kopieren und wieder zurück in den Ingress EDS Data Store zu schreiben. Es ist auch ersichtlich, dass sich durch diesen zusätzlichen Kopiervorgang die Round Trip Time vergrössert.
Das Verhältnis von Instruktionen, die pro Paket ausgeführt werden, zu Instruktionen, die pro Byte eine Paketes ausgeführt werden, hat auf den maximalen Durchsatz einen grossen Einfluss. Dies ist in Abbildung 25 bei «cost per packet» gut sichtbar. Während bei einer kleineren Anzahl zusätzlicher Iterationen die Kosten fast konstant sind, steigen ab 7 zusätzlichen Iterationen die Kosten linear an. Dies kann damit erklärt werden, dass bei kleinen Datenpaketen und wenig zusätzlichen Iterationen, der Aufwand pro Datenpaket gegenüber den Aufwand pro Byte dominiert. Werden mehr als 7 Iterationen pro Byte durchgeführt, nimmt der Aufwand, der abhängig ist von der Datenpaketlänge, überhand.
Fazit: Operationen die pro Byte eines Datenpaketes ausgeführt werden, sind sehr teuer und haben einen starken Einfluss auf den maximalen Durchsatz des Netzwerkprozessors. Ein Pico Plugin, das ein Datenpaket seriell Byte für Byte bearbeitet, würde den Netzwerkprozessor viel stärker belasten, als eine Pico Plugin, dass eine konstante Anzahl Operationen pro Datenpaket ausführt. In den weiterführenden Arbeiten müsste man sich überlegen, ob man Pico Plugins mit einer Ausführungszeit von O(n) zulassen will, oder ob man sich auf die Klasse von Pico Plugins mit Ausführungszeit O(1) beschränken will.

7 Zusammenfassung und Ausblick

7.1 Was wurde erreicht

In dieser Arbeit wurde der IBM PowerNP NP4GS3 Netzwerkprozessor verwendet. Es war relativ schnell gelungen, den Netzwerkprozessor in der «Ethernet-Attached External Control Point» Konfiguration einzusetzen. Durch die grosse Unterstützung von Roman Pletka vom IBM Zurich Research Laboratory ist es möglich geworden, auf dem embedded PowerPC Subsystem das Monta Vista Linux zu installieren. Damit war es möglich, den Netzwerkprozessor in der «Embedded PowerPC Processor Internal Control Point» Konfiguration einzusetzen und zu evaluieren.
Die Diplomarbeit hat aufgezeigt, wie der IBM PowerNP NP4GS3 für PromethOS eingesetzt werden kann. Die im Abschnitt 3 aufgezeigte Architektur lagert die Funktionalität der Paketklassifikation und das IP Forwarding auf den Netzwerkprozessor aus. Des weiteren wird bereits im Netzwerkprozessor entschieden, welches Datenpaket für welches PromethOS Plugin ist. Dadurch und durch die Auslagerungen von Funktionalität auf den Netzwerkprozessor soll der Host entlastet werden und ein Performancegewinn realisiert werden. Die Messungen in Abschnitt 5 haben bestätigt, dass die Implementation dieser Architektur in der Lage ist, die gesamte Bandbreite von Gigabit-Ethernet auszunützen. Die Evaluation hat aber auch aufgezeigt, dass die hohe Performance nur erreicht werden kann, wenn das Application Reference Board über einen schnellen PCI Bus angesteuert wird, der mehr als 2 Gbit/sec (Half Duplex) Durchsatz hat.
Die Messungen auf dem embedded PowerPC Subsystem haben aufgezeigt, dass der PowerPC mit einer Taktfrequenz von 133 MHz zu langsam ist, um PromethOS Traffic mit Linespeed zu verarbeiten. Zudem ist die Schnittstelle zwischen dem PowerPC und dem Netzwerkprozessor mit 42 Mbit/sec zu langsam, um den PowerPC für PromethOS in dieser Architektur einzusetzen.
In Abschnitt 3.6 wurde ein Architekturansatz aufgezeigt, wie es möglich wäre, mit Pico Plugins Netzwerkprozessoren für PromethOS noch besser einzusetzen. Die Messungen in Abschnitt 6 haben aufgezeigt, dass der Netzwerkprozessor durchaus in der Lage ist, durch den Einsatz von Pico Plugins die Performance von PromethOS noch zu steigern.

7.2 Weiterführende Arbeiten

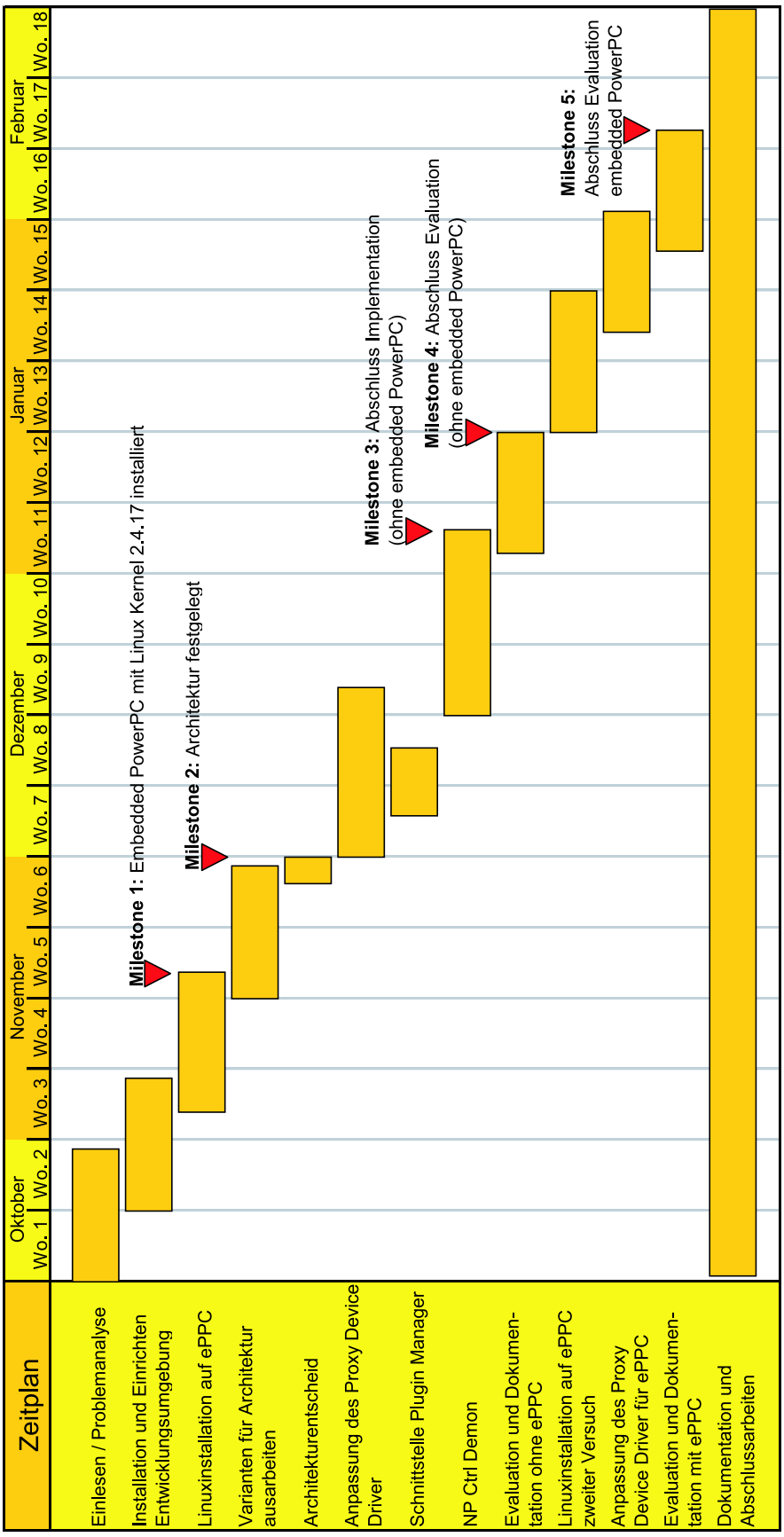
Es gibt zwei verschiedene Gebiete, die aufgrund dieser Diplomarbeit weiterverfolgt werden können. Der in Abschnitt 3.6 erläuterte Ansatz von Pico Plugins müsste weiterverfolgt werden, um weitere Performancegewinne durch den Netzwerkprozessor zu erhalten. Die Arbeiten für die Konzeption und Entwicklung von Pico Plugins wären sehr umfangreich. Man könnte sie wie folgt strukturieren:

- Es ist wünschenswert, dass die Pico Plugins von mehreren Netzwerkprozessor Architekturen unterstützt werden. Dazu muss abgeklärt werden, ob sich die Architektur auch auf anderen Netzwerkprozessoren implementieren liesse. Eventuell müssten auch kleine Änderungen im Architekturansatz in Kauf genommen werden.
- Pico Plugin Sprache: In welcher Sprache sollen Pico Plugins programmiert werden, damit die Möglichkeiten der Netzwerkprozessoren ausgeschöpft werden können? Die Pico Plugin Sprache muss zudem eine Programmverifikation unterstützen, um ein «korrektes» Ablaufen der Pico Plugins zu garantieren.
- Unabhängig von den beiden zuvor erwähnten Arbeiten muss abgeklärt werden, wie der Picocode auf dem NP4GS3 zur Laufzeit geändert werden kann, damit es möglich wird, Picocode Module zur Laufzeit zu laden und entladen. Neben dem Entwerfen einer Architektur für Picocode Module muss der bestehende Picocode angepasst werden und die Funktionalität der Picocode Verwaltung implementiert werden.
- Sind die Pico Plugin Sprache und die Architektur für Picocode Module vorhanden, kann ein Pico Plugin Compiler entwickelt werden. Damit wird es möglich, Pico Plugins zur Laufzeit zu kompilieren und in den Netzwerkprozessor zu laden.

Will man weiterhin am Einsatz des embedded PowerPC Subsystems festhalten, muss die Schnittstelle zwischen dem PowerPC und dem Netzwerkprozessor optimiert werden. Bei dieser Arbeit muss man die Schwerpunkte auf folgende Aspekte legen:

- Wie kann man das Kopieren von Socket Buffers im D6 DRAM Speicher verhindern? Es soll nach einer Architektur gesucht werden, bei welcher der GPH Thread direkt in die Socket Buffers schreibt und ein späteres Kopieren überflüssig wird.
- Gibt es Möglichkeiten, um die Interprozesskommunikation zwischen dem Proxy Device Driver (PowerPC) und dem GPH (Netzwerkprozessor) zu optimieren, um einen höheren Durchsatz der Schnittstelle zu erreichen? Bei diesem Punkt muss untersucht werden, ob der Interprozesskommunikation via Interrupts Grenzen gesetzt sind. Ebenfalls soll der Overhead, der durch die Interrupts verursacht wird, quantifiziert werden.
- Wird ein Ansatz für eine Optimierung gefunden, sollen diese Änderungen in den Proxy Device Driver und in den Picocode des GPH einfliessen.

A Zeitplan


Zeitplan
Oktober
Wo. 1
Wo. 2
Wo. 3
November
Wo. 4
Wo. 5
Wo. 6
Dezember
Wo. 7
Wo. 8
Wo. 9
Wo. 10
Wo. 11
Wo. 12
Januar
Wo. 13
Wo. 14
Wo. 15
Wo. 16
Februar
Wo. 17
Wo. 18
Einlesen / Problemanalyse
Installation und Einrichten Entwicklungsumgebung
Linuxinstallation auf ePPC
Varianten für Architektur ausarbeiten
Architekturentscheid
Anpassung des Proxy Device Driver
Schnittstelle Plugin Manager
NP Ctrl Demon
Evaluation und Dokumentation ohne ePPC
Linuxinstallation auf ePPC zweiter Versuch
Anpassung des Proxy Device Driver für ePPC
Evaluation und Dokumentation mit ePPC
Dokumentation und Abschlussarbeiten
Milestone 1: Embedded PowerPC mit Linux Kernel 2.4.17 installiert
Milestone 2: Architektur festgelegt
Milestone 3: Abschluss Implementation (ohne embedded PowerPC)
Milestone 4: Abschluss Evaluation (ohne embedded PowerPC)
Milestone 5: Abschluss Evaluation embedded PowerPC

B Aufgabenstellung

Institut für
Technische Informatik und
Kommunikationsnetze

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Wintersemester 2002/2003

Diplomarbeit

für

Pascal Erni

Betreuer: Lukas Ruf
Stellvertreter: Matthias Bossardt

Ausgabe: 21.10.2002
Abgabe: 21.02.2003

Einsatz und Programmierung des IBM NP4GS3 Netzwerkprozessors für Aktive Netzwerkknoten unter Linux

1 Einführung

Aktive Netzwerkknoten (Active Network Nodes (ANN)), die dem Plugin Modell [1] folgen, bieten die Möglichkeit an, ihre Funktionalität zur Laufzeit durch Installation von Service Code zu erweitern. Das Ziel sollte es sein, Code von Service Providern auf einem ANN installieren zu können, ohne dass die Funktion des ANNs beeinträchtigt werden kann. Auf einem ANN führt ein Node Operating System (NodeOS) die Funktion der Code-Verwaltung durch.
Am Institut für Technische Informatik und Kommunikationsnetze (TIK) der ETH Zürich (ETHZ) wurde ein Node Operating System, PromethOS [4], [6], unter Linux [5] entwickelt, welches die Installation von Code und deren Ausführung im Linux Kernel (sog. Kernel-Module[1]) gestattet. PromethOS stellt ein PromethOS Execution Environment (PromethOS-EE) zur Verfügung, in dessen Instanzen PromethOS Plugins installiert und zur Ausführung gebracht werden können.
Seit relativ kurzer Zeit werden von Prozessorherstellern sogenannte Netzwerkprozessoren hergestellt. Netzwerkprozessoren verfügen in den meisten Fällen über einen sogenannten Core, welcher für das interne Management des Netzwerkprozessors zuständig, und sogenannte micro- oder picoEngines, welche für Netzwerkoperationen wie Byte-Vergleichs- oder Tabellen-Abfrageoperationen optimiert werden. Diese Engines werden vom Core gesteuert. Typische Vertreter dieser Klasse der Netzwerkprozessoren werden von IBM (NP4GS3) [2] und Intel (IXP1200/2400/2800) [3] hergestellt.
Im Bereich des Active Networking (AN) werden Neztwerkprozessoren erst seit kurzer Zeit eingesetzt. Eine Möglichkeit Netzwerkprozessoren zu verwenden, ist die eher statische Konfiguration des Cores als Bridge zwischen dem Hostprozessor und den Engines.
Am TIK wird ein Ansatz verfolgt, welcher es gestatten soll, die Cores ebenfalls als dynamisch programmierbare Ausführungseinheiten in aktiven Knoten zu verwenden. Somit sollen die Netzwerkprozessorbasierten Interface Cards in einem PromethOS-basierten ANN eingesetzt werden können um den Host zu entlasten.

[1] Kernel-Module, die durch PromethOS verwaltet werden, werden PromethOS Plugins genannt.

2 Aufgaben: PromethOS-NP

Im Rahmen dieser Diplomarbeit soll die Einsatzmöglichkeit Netzwerkprozessor-basierter Interface Cards zur Unterstützung eines PromethOS-basierten ANN untersucht werden. Eine NP4GS3-basierte Karte von Silicon & Software Systems (S3group) kommt zum Einsatz.

3 Vorgehen

- Richten Sie sich eine Entwicklungsumgebung (GNU Tools) unter Linux ein.
- Machen Sie sich vertraut mit den Unterlagen (Dokumentation und Source Code) zu Linux 2.4.17, PromethOS und der NP4-Karte.
- Erstellen Sie einen Zeitplan, in welchem Sie die von Ihnen zu erreichenden Meilensteine Ihrer Arbeit identifizieren.
- Nehmen Sie die NP4-Karte in Betrieb.
- Entwickeln Sie ein Modell für den Einsatz von Netzwerkprozessoren in aktiven Netzwerkknoten.
- Definieren Sie die Kriterien, nach denen Sie die Einsetzbarkeit von Netzwerkprozessoren in ANNs beurteilen.
- Entwickeln Sie eine Node-Architektur, die PromethOS um Netzwerkprozessor Interface Cards erweitert.
- Definieren Sie die Interfaces.
- Implementieren Sie Ihre Architektur.
- Verifizieren, evaluieren und demonstrieren Sie das Erreichte durch eine Beispielapplikation.
- Dokumentieren Sie die Resultate ausführlich.

Auf eine klare und ausführliche Dokumentation wird besonders Wert gelegt. Es wird empfohlen, diese laufend nachzuführen und insbesondere die entwickelten Konzepte und untersuchten Varianten vor dem definitiven Variantenentscheid ausführlich schriftlich festzuhalten.

4 Organisatarorische Hinweise

- Die von IBM zur Verfügung gestellten Unterlagen unterliegen den Bedingungen eines **Non-Disclosure-Agreement (NDA)**. Sie dürfen nicht an institutsfremde Personen oder Firmen weitergegeben werden.
- Am Ende der zweiten Woche ist ein Zeitplan für den Ablauf der Arbeit vorzulegen und mit dem Betreuer abzustimmen.
- Mit dem Betreuer sind regelmässige, zumindest wöchentliche Sitzungen zu vereinbaren. In diesen Sitzungen sollen die Studenten mündlich über den Fortgang der Arbeit und die Einhaltung des Zeitplanes berichten und anstehende Probleme diskutieren.
- Am Ende des ersten Monates muss eine Vorabversion des Inhaltsverzeichnis zur Dokumentation dem Betreuer abgegeben und mit diesem besprochen werden.
- Nach der Hälfte der Arbeitsdauer soll ein kurzer mündlicher Zwischenbericht abgegeben werden, der über den Stand der Arbeit Auskunft gibt. Dieser Zwischenbericht besteht aus einer viertelstündigen, mündlichen Darlegung der bisherigen Schritte und des weiteren Vorgehens gegenüber Professor Plattner.
- Am Schluss der Arbeit muss eine Präsentation von **20 Minuten** im Fachgruppen- oder Institutsrahmen gegeben werden. Anschliessend an die Schlusspräsentation soll die Arbeit Interessierten praktisch vorgeführt werden.
- Die Arbeit muss regelmässig auf dem CVS-Server cvs.promethos.org gesichert werden. Es ist darauf zu achten, dass die richtige CVS-Branch verwendet wird.

- Der Topsy [7] oder der Linux Coding Style muss eingehalten werden.
- Bereits vorhandene Software kann übernommen und gegebenenfalls angepasst werden.
- Die Dokumentation ist mit dem Satzsystem LaTeX zu erstellen.
- Es ist ein mit Bindespiralen gebundener Schlussbericht (am TIK vorhanden) über die geleisteten Arbeit abzuliefern (4 Exemplare). Dieser Bericht besteht aus einem Abstract, einer Einleitung, einer Analyse von verwandten und verwendeten Arbeiten, sowie einer vollständigen Beschreibung der Konfiguration von den eingesetzten Programmen. Der Bericht ist in Deutsch oder Englisch zu halten und beinhaltet sowohl eine deutsche wie auch eine englische Zusammenfassung (sog. Executive Summary), die Aufgabenstellung und den Zeitplan.
- Die Arbeit muss auf CDROM archiviert abgegeben werden. Stellen Sie sicher, dass alle Programme sowie die Dokumentation sowohl in der lauffähigen, resp. druckbaren Version als auch im Quellformat vorhanden, lesbar und verwendbar sind.
- Diese Arbeit steht unter der GNU General Public License (GNU GPL). Die von IBM zur Verfügung gestellten Informationen unterliegen den Lizenzbestimmungen von IBM.
- Diese Arbeit wird als Diplomarbeit an der ETH Zürich durchgeführt. Es gelten die Bestimmungen hinsichtlich Kopier- und Verwendungsrechte der ETH Zürich.

Literatur

[1] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router plugins: A software architecture for next-generation routers, 2000.
[2] IBM Corp. Datasheet IBM NP4GS3. http://www.ibm.com, 2000.
[3] Intel Corp. Intel IXP1200 network processor – datasheet. http://www.intel.com, 2000.
[4] R. Keller, L. Ruf, A. Guindehi, and B. Plattner. PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing. In *Proceedings of the Fourth Annual International Working Conference on Active Networks IWAN*, number 2546 in Lecture Notes in Computer Science, Zurich, Switzerland, December 2002. Springer Verlag.
[5] Linux Homepage. http://www.linux.org, 2002.
[6] Lukas Ruf. The PromethOS Homepage. http://www.promethos.org, 2001.
[7] The Topsy Core Team. Topsy coding style. http://www.topsy.net/Standards, 2002.

Zürich, den 21.10.2002

C Konfiguration

C.1 NP Control Point Konfigurationsdatei (Simulation Mode)

```
############################################################
#
#  NP Control Point configuration file
#  (external CP model)
#
# /etc/npcp.d/npcp
#

TIMER=100
TRACE_OFF=TS

# Path to the operational picocode
# It must be set to the proper picocode location
PICOCODE=/opt/aso-134/obj/i386-redhat-linux/npdd/pico/link/np3b.elf
SIMCODE=/opt/aso-134/obj/i386-redhat-linux/npdd/pico/link/np3b.elf

BLADE=1
NP_VERSION_LABEL=NP4GS3_B
DBOARDA=GIGABIT
DBOARDB=GIGABIT
DMUA=GMII
DMUB=GMII
DMUC=GMII
DMUD=GMBI
```

C.2 Interface Konfigurationsdatei (Hardware Mode)

```
#################################
#
#  interface configuration file
#  (hardware mode)
#
# /etc/npcp.d/ifcfg.arb
#


#-------------------------------
# PTS interface configuration
#-------------------------------
PTSNAME=npct10
PTSPORT=0x5555
PTSIPADDR=3.3.3.3
PTSIPMASK=255.255.255.0

#-------------------------------
# NP Interface configurations
#-------------------------------

# RETH 1: connected to RETH 2 (crossed cable)
#-----------------------------------------
DEVICE=reth1
PORT=1.1.2
L4OPT=YES
ITFTYPE=ethernet
IPADDR=192.168.91.254
IPMASK=255.255.255.0

# RETH 2: connected to RETH 1 (crossed cable)
#-----------------------------------------
DEVICE=reth2
PORT=1.2.1
L4OPT=YES
ITFTYPE=ethernet
IPADDR=192.168.92.254
IPMASK=255.255.255.0

# RETH 3: connected to ETZ main switch
#-----------------------------------------
DEVICE=reth3
PORT=1.2.2
L4OPT=YES
ITFTYPE=ethernet
IPADDR=129.132.57.90
IPMASK=255.255.255.192
```

C.3 Interface Konfigurationsdatei (Simulation Mode)

```
##################################
#
#  interface configuration file
#  (simulation mode)
#
# /etc/npcp.d/ifcfg.sim
#


#--------------------------------
# PTS interface configuration
#--------------------------------
PTSNAME=npct10
PTSPORT=0x5555
PTSIPADDR=3.3.3.3
PTSIPMASK=255.255.255.0

#--------------------------------
# NP Interface configurations
#--------------------------------

DEVICE=reth1
PORT=1
L4OPT=YES
ITFTYPE=ethernet
IPADDR=192.168.91.254
IPMASK=255.255.255.0

DEVICE=reth2
PORT=2
L4OPT=YES
ITFTYPE=ethernet
IPADDR=192.168.92.254
IPMASK=255.255.255.0

DEVICE=reth3
PORT=3
L4OPT=YES
ITFTYPE=ethernet
IPADDR=192.168.93.254
IPMASK=255.255.255.0
```

C.4 Linux Kernel Konfigurationsdatei für den embedded PowerPC 405

Die Datei `.config` wurden mit `make menuconfig` erzeugt. Es sind nur die gesetzten Optionen aufgelistet.

```
#
# Automatically generated make config: don't edit
#
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_HAVE_DEC_LOCK=y
CONFIG_GENERIC_ISA_DMA=y

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y
CONFIG_ADVANCED_OPTIONS=y

#
# Loadable module support
#
CONFIG_MODULES=y
CONFIG_KMOD=y

#
# Platform support
#
CONFIG_PPC=y
CONFIG_PPC32=y
CONFIG_4xx=y
CONFIG_RAINIER=y
CONFIG_NP405GS=y
CONFIG_BIOS_FIXUP=y
CONFIG_VXBOOT=y
CONFIG_IBM405_ERR77=y
CONFIG_IBM_OCP=y
CONFIG_405XX_DMA=y
CONFIG_UART0_TTYS0=y
CONFIG_IBM405_ERR51=y
CONFIG_NOT_COHERENT_CACHE=y
CONFIG_SONIC_SERIAL=y

#
# General setup
#
CONFIG_PIN_TLB=y
CONFIG_NET=y
CONFIG_SYSCTL=y
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SEMMNI=128
CONFIG_SYSVIPC_SEMMSL=250
CONFIG_KCORE_ELF=y
CONFIG_BINFMT_ELF=y
CONFIG_KERNEL_ELF=y

#
# Parallel port support
#
CONFIG_CMDLINE_BOOL=y
```

```
CONFIG_CMDLINE="root=/dev/ram console=ttyS0 init=/bin/busybox"

#
# Block devices
#
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_SIZE=16384
CONFIG_BLK_DEV_INITRD=y

#
# Networking options
#
CONFIG_PACKET=y
CONFIG_NETLINK_DEV=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
CONFIG_IP_PNP=y
CONFIG_ARPD=y
CONFIG_SYN_COOKIES=y

#
# Network device support
#
CONFIG_NETDEVICES=y

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y

#
# Character devices
#
CONFIG_SERIAL=y
CONFIG_SERIAL_CONSOLE=y
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256

#
# File systems
#
CONFIG_EXT3_FS=y
CONFIG_JBD=y
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_TMPFS=y
CONFIG_PROC_FS=y
CONFIG_DEVPTS_FS=y
CONFIG_EXT2_FS=y

#
# Network File Systems
#
CONFIG_NFS_FS=y
CONFIG_ROOT_NFS=y
```

```
CONFIG_SUNRPC=y
CONFIG_LOCKD=y

#
# Partition Types
#
CONFIG_MSDOS_PARTITION=y
CONFIG_NLS=y

#
# Native Language Support
#
CONFIG_NLS_DEFAULT="iso8859-1"
```

C.5 Embedded PowerPC Subsytem Shellskript

Das Shellskript wurde verwendet, um die Initial Ram Disk zu aktuallisieren, den Linux Kernel zu kompilieren und das erstellte Image in den D6 DRAM Speicher zu kopieren.

```
#!/bin/sh
########################################################################
#
# IBM NP4GS3, embedded PowerPC Subsystem
#
# Shellscript to build initrd, make linux kernel and upload image.
#
# /opt/aso-134/exe/i386-redhat-linux/make_image
#



NPDD_SCRIPT=/opt/aso-134/exe/i386-redhat-linux/npdd
KERNEL_SRC_PATH=/opt/hardhat/devkit/lsp/ibm-np4gs3-ppc_405/linux
ASO_PPC_EXE_PATH=/opt/aso-134/exe/ppc405-hardhat-linux
INITRD_IMAGE_NAME=ramdisk.image
INITRD_IMAGE_PATH=/home/initrd
INITRD_MOUNT_PATH=/home/initrd/mnt

# Source function library.
. /etc/rc.d/init.d/functions


# restart NP and rethdd

$NPDD_SCRIPT stop
$NPDD_SCRIPT start


# building initial ram disk

if [ ! -d $INITRD_MOUNT_PATH/usr ]; then
action "Decompressing initial ram disk" gunzip $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME.gz
action "Mounting initail ram disk" mount -t ext2 -o loop $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME $INITRD_MOUNT_PATH
fi

action "Copying binaries to initial ram disk" cp -f $ASO_PPC_EXE_PATH/* $INITRD_MOUNT_PATH/usr/bin/
action "Unmounting initial ram disk" umount $INITRD_MOUNT_PATH
action "Runnig gzip to pack inital ram disk"  gzip -9 $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME
action "Copying to linux kernel sourcetree" cp -f $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME.gz \
                                            $KERNEL_SRC_PATH/arch/ppc/boot/images/
action "Decompressing initial ram disk" gunzip $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME.gz
action "Mounting initail ram disk" mount -t ext2 -o loop $INITRD_IMAGE_PATH/$INITRD_IMAGE_NAME $INITRD_MOUNT_PATH


# making kernel image

cd $KERNEL_SRC_PATH
action "Making linux kernel image" make zImage.initrd
cd -


# upload image

action "upload kernel image to D6 DRAM" ./d6load $KERNEL_SRC_PATH/arch/ppc/boot/images/zvmlinux.initrd.vxboot
```

D Hinzugefügte und geänderter Dateien im ASO 1.3.4

./aso-134/src/	
Makefile	Einbindung der neuen Makefiles

./aso-134/src/pts/linux/ethernet/	
rethdd_module.c	Erweiterung des Proxy Device Drivers
rethdd_structs.h	Wurde um die Definition des D203 Headers erweitert. Dieser Header wird für das Senden von IP Datenpaketen (ohne Ethernet Header) vom Plugin Manager zum Netzwerkprozessor
np_promethos.h	neu: Definition der Schnittstelle zwischen Plugin Manager und Proxy Device Driver

./aso-134/src/npdd/cls/	
np_cls_ip_api.cpp	Wurde so angepasst, damit die Plugin ID anstelle von NextHopIP in den Tree geschrieben wird
np_cls_ip_api.h	Hinzufügen des Action Type Attributes «PromethOS»
np_cls_npdata.h	Hinzufügen des Action Type Attributes «PromethOS»

./aso-134/src/npdd/pico/src/data/	
l4.asm	Hinzufügen des Action Type Attributes «PromethOS»

./aso-134/src/npdd/pico/inc/data/	
l4.inc	Hinzufügen des Action Type Attributes «PromethOS»

./aso-134/src/wrapper/linux/	
proxy_remote_api.c	Fehlerbehebung und Ersetzung der bestehenden Debug Makros
proxy_remote.h	Ersetzung der bestehenden Debug Makros

./aso-134/src/wrapper/linux/NPCtrlD/	
NPCtrlD.c	neu: NP Control Daemon
NPCtrlD.h	neu: NP Control Daemon
NPCtrlMsg.h	neu: NP Control Messages
Makefile	neu

./aso-134/src/wrapper/linux/NPCtrlD/	
NPCtrl.c	neu: NP Control Client
NPCtrl.h	neu: NP Control Client
Makefile	neu

./aso-134/src/pluginManagerSim/	
pluginManagerSim.c	neu: Plugin Manager Simulator
pluginManagerSim.h	neu: Plugin Manager Simulator
Makefile	neu

Tabelle 8: Hinzugefügte und geänderte Dateien

E Installationsanleitung für den Plugin Manager Simulator

Der Plugin Manager Simulator wurde bis jetzt unter RedHat 7.3 und RedHat 8.0 eingesetzt. Für die Installation ab Diplomarbeit-CD sind folgende Schritte notwendig. Sourcen ab CD installieren:

```
# cd /opt
# cp /mnt/cdrom/src/ibm_aso-134_for_promethos.tgz /opt/
# tar xzf ibm_aso-134_for_promethos.tgz

# cp /mnt/cdrom/src/hhl_2.4.17-np4gs3-sonic.tgz /opt/
# tar xzf /hhl_2.4.17-np4gs3-sonic.tgz
```

Konfigurationsdateien kopieren:

```
# cp -rf /mnt/cdrom/config/npcp.d /etc/
```

Software Developer Toolkit installieren (siehe [9])
Verzeichnisse für Picocode erstellen:

```
# mkdir /opt/npref/
# mkdir /opt/npref/bin
# mkdir /opt/npref/bin/np_pico
```

ASO konfigurieren und kompilieren:

```
# cd /opt/aso-134/src
# ln -sf /usr/bin/perl /bin/perl
# ./configure sonicrh
# make
```

Picocode kompilieren:

```
# cd npdd/pico
# make np3b
# ln -sf /opt/aso-134/obj/i386-redhat-linux/npdd/pico/link/np3b.elf \
  /opt/npref/bin/np_pico/np3b.elf
```

Messung durchführen:

```
# cd ../exe/i386-redhat-linux
# npdd start
# ./npcp
# ./NPCtrlD
# ./start_test.sh
```

Ergebnis wird über den klogd Daemon ausgegeben:

```
# tail -310 /var/log/messages
```

Auswertung der Resultate (siehe Excel Dateien auf der CD)

Literatur

[1] Ralph Keller, Lukas Ruf, Amir Guindehi, Bernhard Plattner, *PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing*;
TIK, ETH Zuerich, 2002.

[2] Silicon Software System, *Application Reference Board for the IBM PowerNP NP4GS3 Network Processor User Manual*;
Silicon Software System, 13. März 2002.

[3] IBM, *IBM PowerNP, Reference Design Software for Linux, User's Guide*;
IBM Microelectronics Division, 30. September 2002.

[4] IBM, *IBM PowerNP, Advanced Software Offering, User's Guide*;
IBM Microelectronics Division, 30. September 2002.

[5] IBM, *IBM PowerNP, Advanced Software Offering, Application Programming Interfaces Reference* ;
IBM Microelectronics Division, 30. September 2002.

[6] IBM, *IBM PowerNP, Advanced Software Offering, Control Application Programming Interfaces Reference*;
IBM Microelectronics Division, 30. September 2002.

[7] IBM, *IBM PowerNP, Advanced Software Offering, Forwarding Picocode Design Reference*;
IBM Microelectronics Division, 30. September 2002.

[8] IBM, *IBM PowerNP, Software Developer's Toolkit, User's Guide*;
IBM Microelectronics Division, 30. September 2002.

[9] IBM, *IBM PowerNP, Software Developer's Toolkit, Tutorial*;
IBM Microelectronics Division, 30. September 2002.

[10] IBM, *IBM PowerNP, Assembler Language Programmer's Guide and Opcode Summary*;
IBM Microelectronics Division, 30. September 2002.

[11] IBM, *IBM PowerNP, NP4GS3, Network Processor*;
IBM Microelectronics Division, 15. Februar 2002.

[12] IBM, *IBM PowerNP, NP4GS3, Network Processor Hardware Reference Manual*;
IBM Microelectronics Division, 13. Februar 2002.

[13] IBM, *IBM PowerNP, NP4GS3, Application Note - Preliminary Performance of NP4GS3-Based Systems*;
IBM Microelectronics Division, 20. December 2001.

[14] BraodCom, *BCM5700 PCI-X 10/100/1000BASE-T CONTROLLER*;
Broadcom Corporation, 2002

[15] Erik Andersen, *BusyBox - The Swiss Army Knife of Embedded Linux* ;
http://www.busybox.net, 2002

[16] Monta Vista Inc., *MontaVista Linux Professional Edition*;
http://www.mvista.com/pro/index.html, 2002

[17] Wehrle, Pählke, Ritter, Müller, Bechler, *Linux Netzwerkarchitektur*;
Addison-Wesley, München, 2002.

[18] Andrew S. Tanenbaum, *Computer Networks*;
Prentice Hall, New Jersey, 1996.

[19] W. Richard Stevens, *UNIX Network Programming, Networking APIs: Sockets and XTI*;
Prentice Hall, New Jersey, 1998.

[20] S. Keshav, *An Engineering Approach to Computer Networking*;
Addison-Wesley, Massachusetts, 1997.

[21] A. Rubini, J. Corbet, *Linux Device Drivers - 2nd Edition*;
O'Reilly & Associates, Cambridge, 2001.

[22] Lukas Ruf, *Latex Essentials*;
TIK, ETH Zuerich, 2002.