Erhöhung der Betriebsystemeffizienz bei GBit-Netzwerkpaketerfassung

Mohannad Alnablsi
(alnablsi@cs.tu-berlin.de)

Seminar „Internet Measurement“ ,
Technische Universität Berlin

WS 2008/2009 (Version vom 23. Januar 2009)

Zusammenfassung

Die Zunahme der Größe und Geschwindigkeit von Rechnernetzwerken und die Erhebung von GBit-Netzwerken zum allgemeinen Standard stellt die Netzwerkdatenerfassung vor eine Herausforderung. Zur Lösung dieses Problems wurden viele Ansätze entwickelt, darunter hardwareoptimierte Systeme, die teuer oder spezifisch sind, oder Softwareansätze, die gewöhnliche Hardware benutzen. L. Deri hat zwei Arbeiten [Deri_BDP] und [Deri_nCap] über Softwareansätze zur Lösung dieses Problems Veröffentlicht. Dabei steht Linux im Mittelpunkt. Es wurden zwei Strukturen diskutiert, um die Effizienz des Systems in Hochgeschwindigkeitsnetzwerken zu erhöhen. In dieser Arbeit werden Probleme wie Interrupt-Live-Lock beim der Gerätebedienung, der effektive Umgang mit Speicher und Kopieren durch Benutzung effektiver Pufferstruktur sowie Probleme dabei, eine komplette Funktionalität für Programme, die im Benutzerraum laufen, anzubieten.

1 Einleitung

Netzwerkdatenerfassung basiert auf dem Prinzip, dass ein Programm den Netzwerkadapter benutzt, um Daten aus dem Netzwerk zu lesen und Informationen über das Netzwerk herauszufinden, wie z.B. Größe des Netzwerkverkehrs, Statistiken über die Belastung des Netzwerks oder die Größe der Pakete und Sicherheit und Robustheit des Netzwerks. Dabei unterscheidet man zwischen *passiver* und *aktiver* Datenerfassung. Für die passive Netzwerkpaketerfassung benutzt ein Programm den Netzwerkadapter, um auf den Netzwerkverkehr zu hören und Messungen zu machen. Bei der aktiven Netzwerkpaketerfassung reicht es nicht, nur auf den Verkehr zu hören, sondern in Abhängigkeit von den gemessenen Daten können bestimmten Daten in das Netzwerk zurückgespeist werden, um bestimmte komplizierte Messaufgaben zu erledigen.

In der Vergangenheit wurden zur Netzwerkdatenerfassung normale PCs benutzt, auf denen allgemeine Betriebssysteme liefen. Mit den heutigen Netzwerken, die sehr oft in Gigabit-Geschwindigkeitsbereichen arbeiten, stellt die Datenerfassung eine Herausforderung sogar für PC hoher Leistung [1] dar. Es wurde bestimmte Hardware entwickelt, um

[1] Die Kategorisierung der PC nach Leistung hängt immer vom Zeitpunkt der Aussage ab. Die Arbeiten die Deri in [Deri_BDP] und [Deri_nCap] veröffentlichte gehören zu dem Zeitraum 2004-2005. Hier ist mit “PC hoher Leistung” einen PC mit CPU in der Ordnung Pentium4 2 GHz gemeint.

Aufgaben der Datenerfassung zu optimieren. Denn mit den normalen Netzwerkadaptern und allgemeinen Betriebssystemen kann man maximal im Bereich von 100MBit ohne Datenverluste arbeiten. Bei Erhöhtem Verkehr treten Probleme auf, auf deren Lösung das Betriebsystem nicht optimiert ist.

Die besonderen Hardwarelösungen haben gewisse Nachteile, im Zusammenhand mit Zielgruppe, fürdenen diese Lösungen entworfen sind, denn solche Lösungen sind sehr teuer und auch spezifisch gestaltet, was ihre Einsetzbarkeit bei kleinen und mittleren Anwendungen verringert. Deri hat in [Deri_BDP] und [Deri_nCap] zwei Lösungsansätze vorgestellt, nach denen man mit normalen Netzwerkadaptern und den gewöhnlichen Anwendungen arbeitet, um Netzwerkdatenerfassung zu durchzuführen. Die heute benutzten Anwendungen zur Netzwerkdatenerfassung wie z.B. [tcpdump] und [Wireshark] basieren auf der [libpcap]-Bibliothek.

"libpcap" ist eine Programmierbibliothek, die Funktionalitäten zur Netzwerkdatenerfassung anbietet. Das Besondere an libpcap ist die breite Einsetzbarkeit, weil es Versionen von libpcap für alle bekannte Betriebssysteme gibt. Dabei bietet libpcap die gleiche Programmierschnittstelle, was eine größere Einsetzbarkeit für Programme, die libpcap benutzen, bedeutet.

Deri hat in [Deri_BDP] und [Deri_nCap] die Funktionalität von libpcap beachtet, was die Einsetzbarkeit der beiden präsentierten Arbeiten erweitert.

2 Hintergrundinformationen

Als Aufhänger für die Veröffentlichung hat Deri in [Deri_BDP] einen Test durchgeführt, mit Konfigurationen, die die volle Leistung von 100MBit-Netzwerken ausnutzen. Deri hat in einem Rechner Netzwerkverkehr generiert und ein Netzwerkswitch benutzt um die tatsächliche Paketanzahl zu messen. Der Empfänger war ein PC geringer Leistung [2], mitdem Deri drei Betriebssysteme betrachtet.

Tabelle 1 zeigt die Quoten der empfangenen Pakete in jeder Konfiguration. Es wird klar, dass Linux eine sehr schlechte Leistung erbracht hat. FreeBSD hat eine bessere Leistung als Linux erbracht, die dennoch bei weniger als 35% liegt. Windows 2000 hat erstaunlicherweise eine bessere Leistung erbracht, die auch unter 70% liegt.

Betriebsystem:	Linux 2.4.x	Linux 2.4.x und mmap	FreeBSD 4.8	Windows 2000
Empfangene Pakete:	0.2%	1%	34%	68%

Tabelle 1: Prozentualer Anteil der empfangenen Pakete bei verschiedenen Betrieb-Systeme.

Um diese Ergebnisse nachvollziehen zu können, braucht man einige Hintergrundinformationen, die in diesem Kapitel erklärt werden.

2.1 Betriebssystemhintergrund

allgemeingenutzte Betriebssysteme haben in der Regel folgende Aufgaben:

1. Schaffung von Koordination zwischen den Programmen,
2. Anbieten einer Laufzeitumgebung mit Lesen, Schreiben und Benutzung der vorhanden Geräte für die Programme.

[2] Mit "PC geringer Leistung" ist ein PC mit CPU in der Ordnung Pentium III 500 MHz.

Man realisiert das durch Trennung des Systems in Benutzerraum und Betreibssystemraum. In dem Benutzerraum laufen Programme, die Benutzer spezifisch sind. Diese Programme haben nichts mit der direkten Organisation des Systems zu tun. In dem Betriebssystemraum befinden sich Operationen, die für Betriebssystemaufgaben zuständig sind (*scheduling, direkter Hardwarezugang ...*). Dieser Raum heißt der Kernel.

Benutzerraumprogramme dürfen keine Operationen vornehmen, die nur im Kernel erlaubt sind. Die Programme sollen Schnittstellen des Kernels benutzen, sodass diese Operationen nur über den Kernel möglich sind. In der Regel sollen alle Daten, die sich im Kernel befinden und von Benutzerraumprogrammen angefordert werden, in den Benutzerraum kopiert werden. Moderne Betriebssysteme erlauben es, die gefragten Daten zu *mappen*, also einen Zugang zu den gefragten Daten zu schaffen, ohne die Daten kopieren zu müssen. Diese Funktionalität wird als Memory-Mapping verwendet und mmap abgekürzt.

Diese Gestaltung des Betriebssystems schafft eine Hierarchie, die einerseits notwendig ist, um die Koordination zwischen den verschiedenen Programmen zu schaffen und die Konsistenz des Systems zu gewährleisten. Andererseits müssen die Programme viele Umwege machen um an Ressourcen zu kommen, was die Effizienz des Systems beeinflussen kann.

Während der Arbeit eines Systems müssen viele Aufgabe erledigt werden die vorher nicht voraussagbar sind, z.B: Lesen von Eingabegeräten *(Tastatur, Maus...)*. Das System kann entweder die ganze Zeit diese Geräte abfragen, oder man lässt das System arbeiten, und wenn diese Geräte Informationen zum Lesen haben oder ein Ereignis eingetreten ist, unterbricht ein Gerät das System durch ein Signal *(Interrupt-Anfrage)*. Dabei bekommt das System mit, dass dieses Gerät Aufmerksamkeit braucht und es reagiert entsprechend. Das System kann dabei jeden Interrupt *maskieren*, sodass solche Interrupt-Abfragen das System nicht unterbrechen können.

Die meisten Geräte generieren Interrupt-Anfragen, u.a. Tastatur, Plattenkontroller, Netzwerkkontroller und Energiekontroller. Für diese Arbeit sind Interrupt-Abfragen und Ihre Behandlung von großer Bedeutung, denn das beeinflusst die Effizienz des Systems.

2.2 Ursache der schlechten Performance *"Interrupt handling issue"*

Live-Lock ist eine Situation, in der ein Programm andere Programme blockiert, weil es immer beschäftigt ist und höhere Priorität besitzt. Dieses Problem tritt bei Hochgeschwindigkeits-Netzwerken in Form eines Interrupt-Live-Locks auf.
Die Netzwerkkarte generiert eine Interrupt-Anfrage, wenn die Karte Aufmerksamkeit von dem Betriebssystem braucht, z.B. wenn ein neues Paket ankommt. Das Betriebssystem wird unterbrochen und startet einen Interrupt-Handling-Mechanismus.
Dieser Mechanismus kann in Abhängigkeit von der Art der Anfrage unterschiedlich sein, z.B. Kopieren das ganzen Paketes oder Stellen einer Aufgabe in einer Warteschlange des Systems, sodass diese später behandelt wird, wenn andere wichtigere Aufgaben abgeschlossen sind. Bei Hochgeschwindigkeits-Netzwerken kommen viele Pakete an und die Netzwerkkarte generiert für jedes Paket eine Interrupt-Anfrage. Das System wird unterbrochen und ist immer mit der Behandlung der Interrupt-Anfragen beschäftigt. So hat es keine Zeit, um andere Aufgaben zu erledigen, wie z.B. das tatsächliche Kopieren von Netzwerkpaketen. Dieses Problem heißt Interrupt-Live-Lock und es führt zu einem großen Verlust von Paketen.

2.3 Device Polling

Als Lösung für das Interrupt-Live-Lock-Problem wurde in den Betriebssystemen der Polling-Mechanismus eingeführt. Dieser Mechanismus funktioniert folgendermaßen:

2.3.1 Funktionsweise

1. Bei Ankunft eines Paketes wird eine Interrupt-Anfrage generiert.
2. Das Betriebssystem maskiert (deaktiviert) dieses Netzwerkkarten-Interrupt, sodass die Netzwerkkarte das Betriebssystem nicht mehr unterbricht (bei Ankunft eines neuen Paketes).
3. Danach liest das Betriebssystem periodisch von der Netzwerkkarte, sodass das System alle Pakete liest ohne für jedes Paket unterbrechen und eine spezielle Behandlung für jedes Paket einzeln vornehmen zu müssen. Außerdem kann das System zwischendurch auch andere wichtige Aufgaben erledigen.
4. Wenn das Betriebssystem vom Treiber erfährt, dass es keine weiteren Aufgaben bei der Netzwerkkarte zu erledigen gibt, demaskiert es den Netzwerkkarten-Interrupt, sodass bei Ankunft von neuen Paketen das System unterbrochen wird.

2.3.2 NAPI, FreeBSD

Device-Polling ist in FreeBSD ab Version 4.5 (Aktuelle Version 7.0)implementiert, in Linux wurde Device-Polling mit NAPI eingeführt (ab Kernel 2.4.23).
Allerdings sollte der Netzwerkkartentreiber bei den beiden Betriebssystemen pollingfähig sein.

2.4 Motivation PF_Ring

Deri hat in [Deri_BDP] einen Test mit einer 1GBit-Netzwerkkonfiguration für die Betriebssysteme Linux und FreeBSD jeweils mit und ohne Benutzung von Device-Polling durchgeführt. Er beschreibt, dass es bei Erhöhung der Paketanzahl es am Anfang keine Paketverluste gibt. Ab einem bestimmten Niveau entsteht in den Systemen ohne Device-Polling Paketverlust und die Anzahl der empfangenen Pakete sinkt trotz Erhöhung der Anzahl der gesendeten Pakete. Dieses Problem tritt wegen des Interrupt-Live-Lock auf. Bei den Systemen mit Device-Polling wird ein Niveau erreicht, auf dem Paketverlust entsteht, aber die Anzahl der empfangenen Pakete mit wachsendem Verlust an Paketen steigt. Aus dieser Beobachtung stellt Deri auch fest, dass FreeBSD eine bessere Paketempfangsquote hat als Linux, obwohl FreeBSD dabei Probleme mit Belastung der CPU hat.

3 PF_RING

Deri beschreibt in [Deri_BDP] einen Test bei 1GBit-Netzwerkkonfigurationen, wobei hauptsächlich drei Paketgrößen verwendet wurden. Im Datenerfassung-PC wurden nur die ersten 128 Bytes von jedem Paket erfasst. Das verwendete Programm basiert auf libpcap. Nach diesem Test hat man die Ergebnisse registriert, wie in der Tabelle 2:

Paketgröße	Linux NAPI Standard libpcap	Linux NAPI mmap libpcap	FreeBSD Polling
64 Bytes	2.5%	14.9%	97.3%
512 Bytes	1.1%	11.7%	47.3%
1500 Bytes	34.3%	93.5%	65.1%

Tabelle 2: Anteil der Empfangene Pakete je nach Konfiguration.

Mit diesem Test wurde eindeutig gezeigt, dass Linux bei GBit-Netzwerken unter Mängeln leidet, die Verbesserung erfordern. Die Memory-Map[3] (mmap)-Version sowie Device-Polling sind Verbesserungen, die allerdings nicht ausreichend sind. Im Gegensatz zum ersten Test schwanken die Leistungen von FreeBSD und Linux. FreeBSD hat eine bessere Leistung bei kleinen, Linux jedoch nur bei großen Paketen.

Unter Verwendung von mmap wird der Weg der Pakete vom Netzwerkadapter zu dem Benutzerraum um den Schritt vom Benutzerraum zum Kernel verkürzt. Trotzdem erbringt Linux eine schlechte Leistung bei kleinen Paketen. Das macht es notwendig, eine Lösung für Linux zu entwerfen, um den Weg vom Netzwerkadapter zum Benutzerraum weiter zu verkürzen.

3.1 Begründung für PF_RING

Der Weg Netzwerkadapter → Kernel → Benutzerraum soll entweder verkürzt oder vereinfacht werden. Mittels mmap hat man den Weg Kernel → Benutzerraum vereinfacht, sodass die Pakete nicht vom Kernel in den Benutzerraum kopiert werden müssen, sondern direkt gelesen werden. Bei dem Weg Netzwerkadapter → Kernel Tratten zwei Probleme auf:

1. Bei Ankunft neuer Pakete generiert der Netzwerkadapter Interrupt-Anfragen, um die Aufmerksamkeit des Kernels zu bekommen und die Pakete lesen zu lassen. Das führt zu dem Problem Interrupt-Live-Lock. Dieses Problem kann sehr effizient mittels Device-Polling gelöst werden.
2. Der Netzwerkkartentreiber empfängt die Pakete und kopiert sie in die Struktur des Kernels, sodass diese Pakete an die passende Anwendung weitergeleitet werden. Diese Aufgabe wird in dem Kernel geschedulet (in einen Ablaufplan eingetragen). Wenn die Paketanzahl zu hoch ist, werden viele Pakete nicht rechtzeitig von der Netzwerkkarte kopiert, so werden sie überschrieben und dabei gehen sie verloren. Deri präsentiert in [Deri_BDP] eine Lösung für dieses Problem, in der vermieden wird, dass die Pakete in die Kernelstruktur gehen müssen, wobei der Kerneloverhead reduziert wird.

Die Lösung von Deri basiert auf normalen Netzwerkadaptern und einer Socketstruktur, die für Netzwerkdatenerfassung optimiert ist. Seine Lösung heißt PF_RING und benutzt mmap, Device-Polling und eine besondere Pufferstruktur, um eine bestimmte Effizienz zu erreichen.

3.2 Aufbau von PF_RING

PF_RING ist eine neue Socketart. Es soll auf Paketaufnahmen optimiert sein, dabei kann man folgende charakteristische Eigenschaften auflisten:

- PF_RING benutzt einen Ringpuffer zum Speichern der angekommenen Pakete. Die Größe dieses Puffer ist gleich für alle Sockets. Der Benutzer kann diese Größe einstellen.
- Die beim PF_RING-Socket empfangenen Pakete werden normalerweise nur in den Ringpuffer kopiert und nicht an andere Schichten in dem System weitergeleitet. Das spart Systemoverhead und führt zur Erhöhung der Leistung. Wenn der Ringpuffer voll ist oder eine bestimmte Aufnahmerate erreicht wurde, werden die Pakete ignoriert.

[3] "memory mapping" einen Bereich vom Kernel zu dem Benutzerraum. So kann das Programm die Daten aus dem Kernel direkt lesen ohne die Daten vom Kernel in den Benutzerraum zu kopieren. Sehe 2.1.

- Ein Netzwerkadapter, der zu einem PF_RING-Socket gebunden ist, kann nur im Lesemodus verwendet werden (nur passive Funktionalität), bis dieses Socket geschlossen wird.
- PF_RING unterstützt mmap, Benutzerraumprogramme greifen in den Ringbuffer mittels mmap() direkt zu.
- Das System stellt einen Dateideskriptor zum PR_RING-Socket zur Verfügung. Dieser Deskriptor soll vom Benutzerraumprogramm zuerst geöffnet werden. Allerdings liest das Programm nicht sofort aus diesem Deskriptor, sondern ruft die Operation mmap() auf, um auf den Ringpuffer zugreifen zu können. Dabei wird das Kopieren vom Kernel in den Benutzerraum vermieden.
- Innerhalb des Ringpuffers wird keine Speicher-Reservierung/ -Freigabe durchgeführt. Das geschieht einmal am Anfang beim Kreieren des Sockets. Neue Pakete werden in den gleichen Speicherbereich geschrieben, in dem es alte Pakete gab. So überschreiben sie die alten Pakete.

Abbildung 1 zeigt eine schematische Struktur von der Lösung PF_RING.

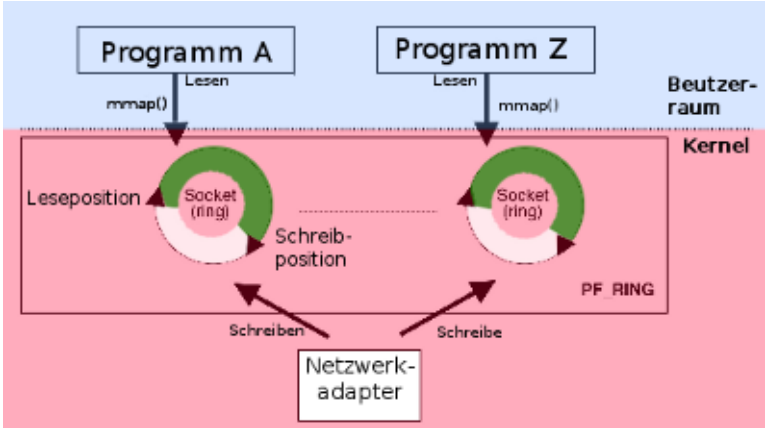

Abbildung 1: Die Struktur von PF_Ring aus [Deri_BDP]

3.3 Vorteile

Bei der Gestaltung von PF_RING wurden Ziele angestrebt, um die Effizienz des Systems möglichst zu erhöhen und die Unterstützung bestehenden Anwendungen anzubieten.

Man kann folgende Vorteile aufzählen:

- Die Benutzung eines Ringpuffers macht es möglich, Pakete kontinuierlich zu speichern ohne unnötige Speicherreservierung.
- Die Pakete werden nicht in der Kernelstruktur behandelt.
- Da die Pakete nicht in die Kernelstruktur gehen, werden die Interrupts schneller behandelt, was den Aufwand verringern kann sogar in Systemen die nicht Device-Polling-fähig sind.
- Die Paketaufnahme nach einer bestimmten Rate kann sehr effizient implementiert werden, weil die Pakete gelöscht werden können ohne dass sie in den Kernel gehen.
- Unter Verwendung von mmap() können Programme im Benutzerraum den Ringpuffer erreichen ohne Systemoverhead, da die Pakete zwischen Kernel und Benutzerraum nicht kopiert werden müssen.

- Durch die Verwendung von einem Ringpuffer pro Socket können die Programme effektiver parallel arbeiten (keine Kernellocks zwischen Programmen notwendig).
- PF_RING bietet die gleiche Programmierschnittstelle wie libpcap an. Dadurch können alle Programme, die auf libpcap basieren, PF_RING benutzen ohne Änderungen an ihren Funktionalitäten.

3.4 polling vs. select()

Aus der Programmseite kann von PF_RING in zwei Weisen gelesen:

Busy-Polling: Hier soll das Programm die ganze Zeit in einer endlosen Schleife laufen und den Puffer abfragen, ob es Daten zum Lesen gibt (Busy-Waiting). Diese Arbeitsweise verbraucht die ganze verfügbare CPU-Leistung, auch wenn es keine Daten zum Lesen gibt.

Polling mittels select(): "select()" ist eine Betriebssystemoperation, die ein Programm blockiert, solange die von ihm angeforderten Ressourcen noch nicht vorhanden sind. Blockierte Programme verbrauchen keine Rechenleistung.

PF_RING unterstützt Polling mittels select(), sodass ein Monitoringprogramm blockiert wird, solange es keine Pakete zum Lesen gibt. Dabei kann die ersparte Rechenleistung benutzt werden, um bestimmte Statistiken zu bilden oder um bestimmte Filteraufgaben zu realisieren. Eine Voraussetzung für die Benutzung von select() ist die Definition eines bestimmten Zeitintervalls. Das aufrufende Programm wird dann maximal für diese bestimmte Zeit blockiert. Datenverluste während des Deblockierens werden so vermieden.

Deri hat in [Deri_BDP] festgestellt, dass Linux mittels PF_RING eine GBit-Netzwerkdatenerfassung mit CPU-Belastung von weniger als 30% vergleichsweise besser als FreeBSD mit Device-Polling ist. Bei der Netzwerkdatenerfassung wird in FreeBSD die CPU intensiv belastet, sodass es bei der Benutzung von FreeBSD zur Datenerfassung keine Möglichkeiten gibt, gleichzeitig andere Aufgaben auf dem gleichen PC durchzuführen (z.B. Analysen von den erfassten Daten).

Für die passive Datenerfassung stellt PF_RING eine gute Lösung dar. Über die aktive Datenerfassung wird im nächsten Kapitel diskutiert.

4 nCap: *"Wirespeed Capture and Transmission"*

Während einer aktiven Teilnahme eines Systems an Messungen von Netzwerksverkehr *(d.h. dass das System Daten ins Netzwerk speist, und nicht nur liest)* kommen Verzögerungen bzw. Latenzen beim Einspeisen der Daten ins Netzwerk manchmal vor. Dies führt zu Inkonsistenzen der Datenmessung, wenn das System nicht effizient aus dem Netzwerk lesen bzw. schreiben kann. Alle passiven Netzwerkerfassungprogramme sind für Monitoring gedacht. Sie wurden nicht für die Teilnahme am Netzwerkverkehr konzipiert. Komplizierte Netzwerkverkehrsanalysen stellen bei hoher Geschwindigkeit eine Herausforderung dar.

Deri präsentiert in [Deri_nCap] eine Lösung für die aktive Netzwerkdatenerfassung, die sowohl für schnellen Datenempfang als auch für schnelles Datensenden optimiert ist. Diese Lösung wurde mit dem Name nCap präsentiert.

4.1 Aufbau

"nCap" stellt ein Netzwerkadapter zur Verfügung, der direkt vom Benutzerraum zum Lesen und Schreiben benutzt werden kann. Dieser Adapter ist ein Gerät, das sich in */dev/ncap/-Gerätsname-* befindet. Dabei wird sichergestellt, dass der Kernel keine Kontrolle über diesen Netzwerkadapter hat, wenn ein Programm ihn über nCap aufruft. Es

kann auch kein weiteres Programm diesen Adapter verwenden, bis das erste Programm fertig ist.

"nCap" besteht aus zwei Komponenten, die auch zwei Schnittstellen präsentieren:

- eine Benutzerraumbibliothek, die eine API bereitstellt mit Funktionen zum Paketesenden und -empfangen aus dem Benutzerraum.
- einen Treiber für die Netzwerkkarte, der eine Schnittstelle zum Netzwerkadapter präsentiert.

Der Gerättreiber von nCap kreiert bei der Initialisierung des Netzwerkadapters zwei Ringpuffer,in denen Pakete zum Senden und Empfangen geschrieben werden.
Der Treiber informiert den Adapter über die Lese- und Schreibpositionen innerhalb dieses Puffers.

Bei der Ankunft eines Paketes schreibt der Netzwerkadapter dieses Paket *(mittels DMA)* direkt in den Ringpuffer, ohne dass die Pakete durch die Kernelstruktur reisen und kopiert werden müssen. Andererseits bietet die Benutzerraumbibliothek einen direkten Zugriff auf diesen Ringpuffer für andere Programme an, die im Benutzerraum existieren, ohne dass diese Programme mit Kernelaufrufen arbeiten müssen.
Wenn ein Programm ein Paket sendet, wird dieses Paket von nCap-API direkt in den Sendepuffer geschrieben und die Leseposition für den Netzwerkadapter durch den Treiber aktualisiert. Das merkt der Netzwerkadapter sofort und sendet dieses Paket. Eine schematische Struktur von nCap wird in der Abbildung 2 dargestellt.

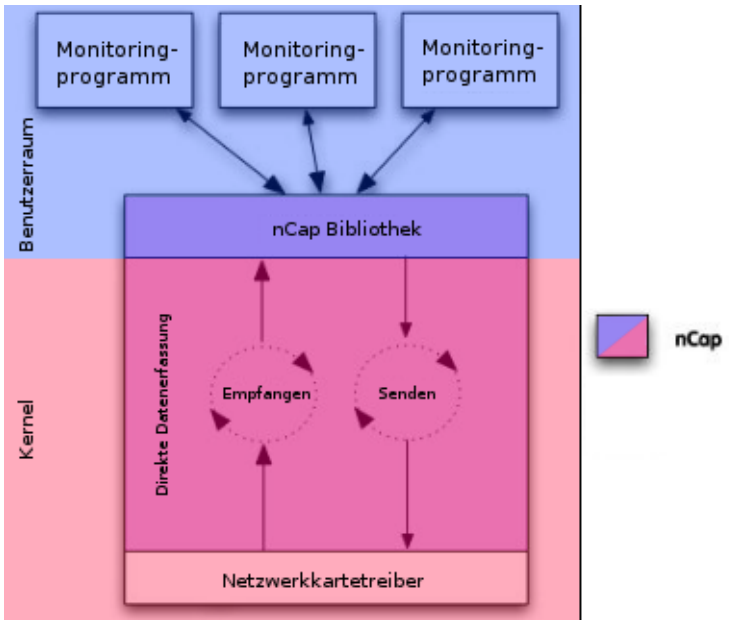

Abbildung 2: Die Struktur von nCap aus [Deri_nCap]

4.2 Benutzung des Netzwerkkontrollers

Pakete durch den Netzwerkkontroller direkt in den Benutzersraum lesen oder schreiben zu lassen verkürzt den Weg in den Kernel um mindestens eine Schicht.Denn in der traditionellen Arbeitsweise werden die Pakete durch den originalen Netzwerkadapter zunächst empfangen und dann als einzelne Pakete an den Kernel weitergeleitet. Das Gleiche gilt in die andere Richtung. Denn mittels nCap-API werden die Pakete vom

Benutzerraum einmal in den Sendepuffer geschrieben. Der Netzwerkadapter sendet die Pakete von dem Sendepuffer direkt ins Netzwerk, ohne sie in Schichten des Kernels weiterzuleiten.
Das Lesen und Schreiben mittel DMA ist sehr schnell und braucht keine Interaktion vom CPU. Dabei gibt es nur eine Schreiboperation sowohl in die Richtung Netzwerk → Benutzerraum als auch die umgekehrte Richtung, was zu großer Effizienz führt. Dabei entsteht einen Nachteil. Der nCap-Treiber muss mit verschieden Netzwerkadapter umgehen können, was eine Frage von Entwicklung ist.

4.3 Vorteile

Die Gestaltung von nCap, die für Geschwindigkeitserhöhung und Latenzverringerung optimiert ist, bringt viele Vorteile für Netzwerkmessungen und Monitoring. Diese Vorteile können folgendermaßen aufgelistet werden:

- "nCap" als Lösung bietet einen direkten Zugang ins Netzwerk vom Benutzerraum. Dabei vermeidet man den Weg über den Kernel und die entstehende Leistungsverluste.
- Der nCap-Treiber kreiert und reserviert die Ringpuffer am Anfang einmalig, sodass bei jeder Paketankunft oder bei jedem Paketversand keine Speicherreservierung und -freigebung notwendig ist, was die Effizienz des Systems erhöht.
- Paketversand ist bei nCap ein einziges Verändern der Leseposition im Netzwerkadapter. der Rest ist Hardware-Implementierung im Netzwerkadapter, was zwei große Vorteile erbringt:
  1. Extrem hohe Senderaten können erreicht werden.
  2. Latenz des Sendprozessìst minimiert, was für weitere Entwicklungsgebieten wie z.B. aktive Messungen und spezifische Netzwerkverkehrgenerierung von Interesse ist.

5 nCap vs. PF_RING

Nachdem man die Kerneigenschaften und den Aufbau von PF_RING kennen gelernt hat, kann man PF_RING und nCap an bestimmten Punkten vergleichen, denn durch die Unterschiede findet man unterschiedliche Einsatzgebiete für beide Lösungen.

- PF_RING ist eine Lösung für passive Paketeerfassung, nCap bietet darüber hinaus die Möglichkeit, Pakete zu senden, mit der gleichen Effizienz.
- PF_RING und nCap bieten legacy Unterstützung für Anwendungen, die [libpcap] benutzen, indem sie libpcap-Funktionen anbieten.
- Die gesamte Funktionalität von PF_RING befindet sich im Kernel, nCap befindet sich im Kernel und im Benutzerraum.
- PF_RING ist ein neues Socket, womit man normale Systemaufrufe benutzen kann, nCap ist eine komplette Lösung, die einen direkten Zugriff auf die Pakete anbietet, ohne die Verwendung von Sockets.
- PF_RING arbeitet effektiv bei Betriebssystemen, die Device-Polling unterstützen und bei Netzwerkadaptern, die Polling-fähig sind. nCap dagegen braucht nur einen Netzwerkadapter, der sich für Lesen und Schreiben direkt steuern lässt. Dafür müssen viele Netzwerkadapter in nCap implementiert werden.
- Mit PF_RING und nCap lasen sich Programme für Netzwerkmessungen bei Netzwerken hoher Leistung mit kleinen Anforderungen an Hardware realisieren.

Deri hat in [Deri_nCap] einen Test durchgeführt mit GBit-Netzwerkkonfiguration und zwei PCs, auf denen Linux 2.4.x lief. Auf einem PC mit Pentium4 1.7GHz CPU wurde PF_RING installiert und auf dem zweiten wurde nCap installiert und Pentium III 550 MHz CPU.

Für Paketgrößen von 64 Bytes, bei denen Linux schwächer war, registrierte Deri, dass bei nCap und diesem PC geringer Leistung eine Datenerfassungsmenge von ca. 205 Mbits erreicht wurde, während bei dem PC mittlerer Leistung und PF_RING eine Menge von ca. 202 Mbits erreicht wurde. Aus diesem Testergebnis kann man nachvollziehen, welchen Vorteil nCap bringen kann.

6 Zusammenfassung

In dieser Arbeit wurden die Schwächen der heutigen allgemeinen Betriebssysteme, die nicht zur Datenerfassung und -messung optimiert sind aufgezeigt. Das Prinzip Device-Polling und seine Rolle bei der Lösung von Interrupt-Live-Lock wurde ausführlich diskutiert. Eine Optimierung der Behandlung von den im Kernel empfangenen Paketen wurde auch vorgeschlagen, um den Kerneloverhead bei hohen Empfangsraten zu minimieren, indem das Prinzip von Ringpuffern benutzt wird. Zuerst wurde PF_RING als Ausnutzung von dem Prinzip des Ringpuffers dargestellt. Dabei werden die Netzwerkpakete durch den Netzwerkkartentreiber von dem Netzwerkadapter gelesen und in den Ringpuffer kopiert, ohne durch die Kernelstruktur und Warteschlangen zu gehen.

Als zweite Anwendung des Ringpuffer-Prinzips wurde nCap beschrieben. Hierbei sind zwei Ringpuffer notwendig. Ein Ringpuffer ist für die empfangenen Pakete und einer für gesendete Pakete zuständig.

Als Schlüsselidee bei nCap im Gegensatz zu PF_RING ist auf das Kopieren der Daten vom Netzwerkadaptertreiber zu verzichten, denn der Netzwerkadapter als Gerät übernimmt die Aufgabe, die Pakete direkt in den Empfangsringpuffer zu kopieren. Dies ist ein großer Optimierungschritt, da man nun auch mit PC geringer Leistung Netzwerkdaten in Hochgeschwindigkeitsbereichen erfassen und generieren kann, was mit teuren hardwareoptimierten Lösungen vergleichbar ist.

Das Einsetzen der diskutierten Techniken kann den mittleren und kleinen Forschungseinrichtungen bzw. Unternehmen erlauben, in hohem Tempo Netzwerkdaten zu erfassen und zu generieren ohne höhere Kosten zur Anschaffung teurerer Hardwarelösungen tragen zu müssen. Dies ist nicht nur für kleine Unternehmen, sondern auch für Universitäten in den Entwicklungsländern sehr wichtig.

Literatur

[Deri_BDP] Luca Deri: *Improving Passive Packet Capture: Beyond Device Polling*; Proceedings of SANE 2004, 2004.

[Deri_nCap] Luca Deri: *nCap: Wire-speed Packet Capture and Transmission*; Proceedings of E2EMON, 2005.

[libpcap] Network Research Group at Lawrence Berkeley Laboratory: *libpcap*; http://www.tcpdump.org

[tcpdump] The Tcpdump team:*tcpdump*; http://ww.tcpdump.org.

[Wireshark] The Wireshark team: *Wireshark*; http://www.wireshark.org.