Optimierung des FreeBSD-Packet-Capturing-Stacks

Ringmap Packet Capturing Stack for High Performance Packet Capture in FreeBSD

Alexander Fiveg
6. Dezember 2010

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation.

Paket-Capturing oder *Sniffing* ist der Prozess des Abfangens von Netzwerkpaketen, mit dem Ziel diese zu speichern, zu analysieren und darzustellen. Aufgrund der limitierten Rechnerleistung und Ineffizienz der Software, kommt es leider oft dazu, dass nicht alle Pakete aus dem Netz untersucht werden können.

Die Hardwareressourcen eines Computers wie Bandbreite der internen Bussen, CPU-Zyklen-Rate und Speicher (RAM und Hintergrundspeicher) sind begrenzt. Das hat zur Folge, dass die Menge der ankommenden Paketen, die ein Computer pro Zeitintervall bearbeiten und speichern kann, auch nicht unendlich groß ist. Die "Geschwindigkeit" des Datentransports zwischen einem Peripherie-Gerät und RAM ist durch die Bandbreite des Bussystem begrenzt. Die Anzahl der im RAM befindlichen Pakete, die sich pro Zeitintervall bearbeiten lässt ist sowohl von der Leistung des Prozessorbusses als auch von der CPU-Leistung abhängig. Wenn die empfangenen Pakete auf die Festplatte geschrieben werden sollen, geschieht dies auch nicht schneller, als es der Hintergrundspeicher erlaubt. Jede von den obengenannten Hardwarebegrenzungen kann Datenverluste bei Capturing verursachen, wenn die Rate der ankommenden Pakete über die Performance-Grenzen der darunterliegenden Hardwarekomponenten steigt.

Aber nicht nur die Hardware stellt einen Flaschenhals für die Datenbearbeitung in einem Rechnersystem dar. Die Hardwareressourcen können von der Software ineffektiv benutzt werden. Zum Beispiel: wenn ein Programm wesentlich mehr Operationen ausführt, als zur Lösung des Problems nötig wären, dann erzeugt es eine unnötig hohe Systemlast und reduziert damit die Datenmengen, die es in einem Zeitintervall bearbeiten könnte.

Das Ziel dieser Arbeit ist es, die für Capturing relevante Komponente des Betriebssystem FreeBSD zu untersuchen, die potentiellen "Engstellen" in der Software, die zu den Datenverlusten führen können, herauszufinden, und effiziente Algorithmen zur Erhöhung des Datendurchsatzes und Reduzierung der Systemauslates beim Capturing zu erarbeiten, zu implementieren und auszuwerten.

1.2 Begriffserklärung

Packet-Capturing nennen wir den Prozess des Empfangens, Filterns und ggf. Darstellens des Datenverkehrs aus einem Netzwerk.

Packet-Capturing-Stack ist die Software, die für Capturing benötigt wird. Sie kann aus mehreren Komponenten bestehen. Im Betriebssystem FreeBSD sind diese Komponenten sowohl im Kern des Betriebssystem (wie Netzwerktreiber und Paketfilter) als auch in der Programm-Bibliotheken wie *libpcap* [5] implementiert.

Capturing-System ist ein Rechnersystem mit der Software, die für Packet-Capturing benötigt wird.

Capturing-Performance ist die Anzahl von Paketen, die das Capturing-System pro einer bestimmten Zeiteinheit aufnehmen, bearbeiten, und speichern kann.

1.3 Hardware- und Software-Voraussetzungen

Für das Projekt sind folgende Hardware und Software vorausgesetzt:

- **Hardware:**
  - PC: **x86**-Architektur
  - Intel 1GbE und 10GbE **825xx** Controllers.
- **Software:**
  - Betriebssystem: FreeBSD **-CURRENT, -STABLE**
  - Netzwerktreiber: *em, lem, ixgbe*

Die vorausgesetzte Hardware hat keine besonderen Ansprüche und ist überall zu bekommen. Die Software steht unter BSD-Lizenz, ist frei erhältlich und einfach auf der vorausgesetzten Hardware installierbar.

1.4 Verwandte Projekte

Zero-Copy BPF Buffers

Im Jahr 2007 haben Robert Watson und Christian Peron (Universität Cambridge) ihre Lösung zur Erhöhung der Performance von FreeBSD Capturing-Stack vorgeschlagen und implementiert [1]. Im Rahmen des "Zero-Copy BPF Buffers"-Projektes wurde ein neuer Capturing-Stack für das Betriebssystem FreeBSD entworfen und implementiert. Im diesem Stack wird das Kopieren der Paket-Daten in den Userspace, durch Memory-Mapping eliminiert. Dadurch werden es keine Kopier-Operationen und keine Systemaufrufe für den Paketzugriff mehr gebraucht. Da diese Operationen besonders viele CPU Zyklen für ihrer Ausführung brauchen, verbessert Zero-Copy-Model die Capturing-Performance.

Das Benutzen von Memory-Mapped-Buffers erfordert allerdings ein neues Konzept für den Zugriff auf Daten from Userspace. Deshalb wurden im Rahmen des *Zero-Copy* Projektes auch die Libpcap-Funktionen für den Zugriff auf Pakete im Shared-Buffer angepasst.

2 Analyse des standard Paket-Capturing-Stacks in FreeBSD

In diesem Kapitel werden die für Capturing relevante Aspekte dargestellt. Es wird dabei analysiert bei welchen Capturing-Komponenten unter welchen Umständen Engpässe bei der Paketerfassung auftretten können. Anhand der herausgefundenen Problemen werden Anforderungen und Ansätze für den Entwurf der neuen Capturing-Software formuliert (siehe Kapitel 3). Zu beachten ist, dass die dargestellten Aspekte sich vollkomen auf die für die Arbeit vorausgesetzten Hardware und Software beziehen (siehe Abschnitt 1.3).

2.1 Softwareaspekte beim Capturing

Abbildung 1 zeigt den standard FreeBSD-Packet-Capturing-Stack. In FreeBSD besteht der Capturing-Stack aus mehreren Komponenten. Das sind der **Netzwerk-Treiber**, die Software für die Paketfilterung (**BPF**[1]) und die **Userspace-Anwendungen**, die ggf. **libpcap-Funktionen** für den Zugriff auf empfangene Pakete benutzen.

Beim Capturing wird jedes aus dem Netz empfangene Paket zuerst in den internen Speicher des Netzwerkcontrollers kopiert. Sobald es möglich ist, macht der Controller den DMA-Transfer der vorhandenen Pakete in den RAM und meldet dies durch ein Interrupt. Weiter werden die Pakete vom BPF gefiltert. Nach der Filterung werden die Pakete von einer Userspace-Anwendung geholt, weiterbearbeitet und eventuell auf der Festplatte gespeichert oder ggf. auf dem Terminal dargestellt.

Im weiteren werden die Komponente des standard Packet-Capturing-Stacks einzeln betrachtet und analysiert.

2.1.1 Netzwerktreiber: Interruptbehandlung

Die erste Komponente des Packet-Capturing Stacks ist der Netzwerk-Treiber (siehe Abbildung 1). Für jedes empfangene Packet wird der Treiber einen neuen Paket-Puffer im RAM allozieren. Nachdem ein Paket vom Netzwerkcontroller in den Paket-Puffer kopiert wurde, meldet der Controller die Anwesenheit von neuen Daten im RAM durch ein Interrupt, und verursacht damit den Aufruf der *Interrupt-Service-Routine* (ISR).

Da die Interruptbehandlung mit der höchsten Priorität ausgeführt wird, sind alle derzeit laufenden Prozesse auf der aktuellen CPU unterbrochen. Dies kann das Capturing-Performance negativ beeinflussen, wenn die Interruptbehandlung zu große Ausführungszeit hat. Die andere Software-Komponenten, die auf empfangene Pakete zugreifen wollen, können in diesem Fall nicht genügend CPU-Zyklen bekommen. Dies kann zu den vollen Puffern und als Ergebnis zu den Paketverlusten führen[2] [8]. Deshalb ist es sehr wichtig, dass die Interruptbehandlung-Funktionen effizient implementiert sind und dadurch möglichst kurze Ausführungszeiten haben.

Die Interruptbehandlung wird in zwei Phasen ausgeführt: die eigentliche ISR und *delayed-ISR*. Die ISR dient nur zum Herausfinden der Ursache des Interrupts und Planen zu einem späteren Zeitpunkt der Ausführung von *delayed-ISR*. Der größte Rechenaufwand bei der Interruptbehandlung findet während der Ausführung der *delayed-ISR* statt. Von allen Aufgaben, welche die *delayed-ISR* erledigt, sind die Speicher-Allozierungen, wahrscheinlich, diejenigen die den Großteil der CPU-Zeit beanspruchen. Im Kontext von *delayed-ISR* wird

[1] Berkley Packet Filter [2, 7, 6]

[2] Dies ist auch als “Receive Livelock” Problem bekannt.

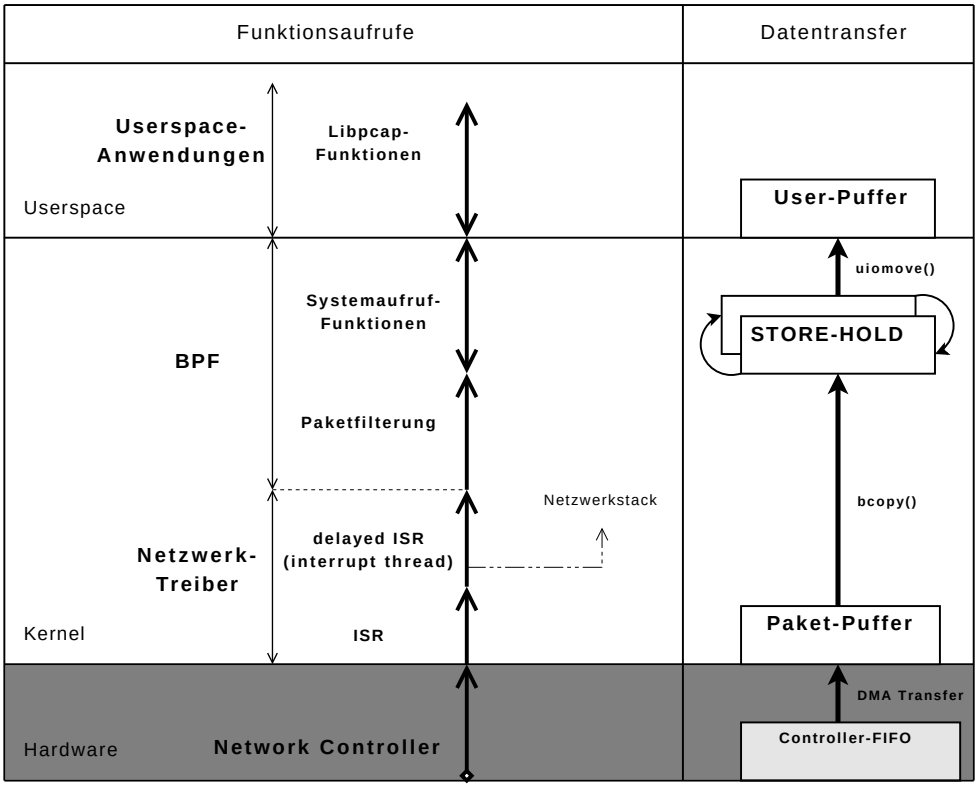

Abbildung 1: FreeBSD Packet Capturing Stack

für jedes empfangene Paket einen neuen Paket-Puffer alloziert. Dies rettet die bevor empfangene Pakete von Überschreibung hat aber einen Nachteil, der sich in Vergrößerung des pro-Paket Overhead zeigt, denn Speicherallozierung ist eine relativ "teure" Operation, die höhe Systemlast und damit die Paketverluste während Capturing verursachen kann.

Eine mögliche Lösung: Die Speicherallozierungen lassen sich durch die Anwendung von einem *Ring-Buffer* ersetzen. Der Ring-Buffer ist eine Datenstruktur, die ein Array mit den zwei Zeiger: *HEAD* and *TAIL* präsentiert. Das Besondere am Ring-Buffer ist es, dass er eine feste Größe hat und die Speicherbereiche, die er verwaltet, nur ein mal alloziert und dann ein nach einander beschrieben und gelesen werden. Dabei, zeigt *HEAD* auf den nächsten freien Speicherbereich, der beschrieben wird. *TAIL* zeigt auf den zuletzt gelesenen Bereich. Im Unterschied zum normalen Array werden die ältesten Inhalte überschrieben, wenn der Ring-Buffer voll ist und weitere Elemente abgelegt werden.

2.1.2 Berkley Packet Filter (BPF)

Der BPF ist die weitere Komponente des FreeBSD Packet-Capturing-Stacks. Der BPF registriert im System für jeden Prozess, der die Pakete erfassen will ein symbolisches Device: `/dev/bpf[0-9]`, und bietet dazu eine Menge von Systemaufrufen (read, ioctl, etc. . .) für den Zugriff auf die Pakete und zur Steuerung der Paketfilterung.

Die in den Paket-Puffern befindliche Pakete werden vom BPF gefiltert, und diejenigen, die durch die Filterregeln akzeptiert wurden, werden in den STORE-HOLD-Puffer kopiert (siehe Abbildung 1). Aus dem STORE-HOLD-Puffer werden die Pakete durch den *read*-

Systemaufruf in Userspace übertragen[3].

Der BPF wird auf dem aktuellen FreeBSD vollkommen in Kernelspace ausgeführt. Es gibt aber auch in der libpcap-Library eine BPF-Implementierung, was die Paketfilterung auch im Userspace erlaubt. Aber das Ausführen von Filterung-Routinen im Kern, sofort nach dem Paketempfang, hat seine Vorteile. Die Kopier-Operationen haben mit dem Unterschied zu den anderen wesentlich längere Ausführungszeiten. Deshalb eliminiert das möglichst frühe Ausführen der Filterung das unnötiges Kopieren von Daten, die anhand der vorhandenen Filterregeln nicht akzeptiert wurden.

Der Performance-Nachteil des vorgestellten BPF-Models besteht darin, dass es die "teure" Paket-Kopier-Operationen im Kernelspace hat und für den Übertragung der Pakete in den Userspace die Systemaufrufe anfordert, was eine zusätzliche Kopier-Operation mit dem Ausführung-Kontextwechsel beeinflusst.

Eine mögliche Lösung: Die Kopier-Operationen und Systemaufruf lassen sich durch die *Memory-Mapping* eliminieren. Zum Beispiel, wenn die Paket-Puffer in den Address-Raum einer Capturing-Anwendung eingeblendet werden, dann braucht diese Anwendung keine Kopier-Operationen und keine Systemaufrufe mehr, um auf die Pakete zugreifen zu können. Dafür muss aber ein Synchronisation-Verfahren erarbeitet werden, wenn die mehrere Prozesse auf den gleichen Paket-Puffer zugreifen wollen.

2.1.3 Libpcap

Libpcap ist eine Programm-Bibliothek, die eine Menge von Funktionen zum bequemen Zugriff auf empfangene Pakete anbietet [5]. Mehrere bekannte Sniffer wie *tcpdump* oder *Wireshark* benutzen libpcap für Packet-Capturing. Auf FreeBSD benutzen die libpcap-Funktionen die BPF-Systemaufrufe für die Paketfilterung und für Zugriff auf gefilterte Pakete, und stellen den Userspace-Anwendungen eine bequeme und einfache Schnittstelle für den Paketenzugriff.

Während Packet-Capturing sind die Aufgaben der libpcap-Funktionen, die Pakete in den Userspace zu holen und den Anwendungen, die auf die Pakete zugreifen wollen einen Pointer auf Paketpuffer zu übergeben. Dabei besteht keinen großen Overhead, der sich deutlich reduzieren lässt.

2.2 Namenskonventionen

Um die Unverständlichkeiten mit den Bezeichnungen zu vermeiden, geben wir hier die Namen, unter denen die standard FreeBSD-Capturing-Stack und der neuen im Rahmen des Projektes entwickelten Capturing-Stack in den weiteren abschnitten auftauchen werden:

ringmap: Wird als Bezeichnung für die im Rahmen des Projektes entwickelte Capturing-Software benutzt.

generic: Wird als Bezeichnung für die standard FreeBSD-Capturing-Software benutzt.

[3] Die genaue Beschreibung ist in *bpf(9)* und *bpf(4)* Manuals [6, 7] und in den Papers von Fabian Schneider [9, 10] zu finden.

2.3 Zusammenfassung

In den vorherigen Abschnitten wurden die für Capturing-Performance relevanten Software-Komponenten dargestellt. Dabei wurde versucht herauszufinden, welche der vorgestellten Komponenten unter welchen Umständen die Performance des Capturing negativ beeinflussen können. Anhand der herausgefundenen Problemen werden die konkrete Anforderungen für den Entwurf der neuen Capturing-Software im Kapitel 3 gestellt.

Betrachten wir noch mal den in Abbildung 1 dargestellten Capturing-Stack des aktuellen FreeBSD. Stellen wir uns den Fall vor, wenn alle Pakete durch die BPF-Filterung akzeptiert sind. Dann gibt es pro Paket drei Kopier-Vorgänge:

1. DMA Transfer in den RAM
    - ist mit Hilfe der Hardware realisiert
2. Kopieren in STORE-HOLD-Puffer
    - ist mit Hilfe der Kernelfunktion `bcopy(9)` gemacht.
3. Kopieren in Userspace[4]
    - ist durch den Systemaufruf `read(2)` erledigt, und verursacht damit ein Kontext-Wechsel

Die erste Kopier-Operation, die durch DMA-Transfer erledigt wird, ist unvermeidlich und nicht optimierbar[5]. Die restlichen zwei Kopier-Operationen, ständiges Speicherallozierung für jedes neue Paket und der Kontext-Wechsel stellen ein Flaschenhals im Capturing-System, falls die Rate der ankommenden Pakete nah zu $1GBit/sec$ oder höher steigt (siehe Abschnitt 4.2.2 Ergebnisse).

Die bestehende Probleme lassen sich durch die Anwendung von *Memory-Mapped-Buffers* und *Ring-Buffer* eliminieren. Memory-Mapping eliminiert die Notwendigkeit der Kopier-Operationen und der Systemaufrufe. Die Anwendung des Ring-Buffer eliminiert die Speicherallozierungen. Da die Kopier-Operationen, Systemaufrufe und Speicherallozierungen, sind die Operationen, welche mit dem Unterschied zu den anderen sehr große Menge von System-Ressourcen anfordern, soll die Eliminierung dieser Operationen die Capturing-Performance erhöhen (siehe Ergebnisse der Messungen 4.2.2).

[4] Das Verwenden von Zero-Copy-BPF Buffers eliminiert diese Kopier-Operation

[5] Eigentlich lässt sich der DMA-Transfer der Paket-Daten in den RAM durch bestimmte Interrupt-Moderation Einstellungen begrenzt optimieren [3, 4]

3 Anforderungen für den neuen ringmap Capturing-Stack

In diesem Kapitel sind die Anforderungen für den Entwurf des neuen *ringmap* Packet-Capturing-Stacks formuliert.

3.1 Eliminierung der Kopier-Operationen

Die zwei Paket-Kopier-Operationen, vom Paket-Puffer in den STORE-HOLD-Puffer und vom STORE-HOLD-Puffer in den Puffer der Userspace-Anwendung, sollen eliminiert werden. Die Pakete sollen sofort nach dem DMA-Transfer im virtuellen Speicher des User-Capturing-Prozesses zugreifbar sein. Dafür müssen die Paket-Puffer in den Adressraum der Capturing-Anwendung eingeblendet werden. Das entsprechende Synchronisationssverfahren muss für den Zugriff auf Paket-Puffer erarbeitet werden (siehe Abbildung 2).

Dies beansprucht der Implementierung der neuen Kernel-Funktionen und der entsprechenden Modifizierung von *libpcap*-Bibliothek.

3.2 Eliminierung der Speicherallozierungen

Keine neue Paket-Puffer sollen für die neue ankommende Pakete alloziert werden. Dafür müssen alle Speicherbereiche für die Pakete und *mbuf*'s[6] als Ring-Buffer verwaltet werden. Alle Paket-Puffer müssen nur ein Mal alloziert und während der Paketerfassung als Ring-Buffer benutzt werden. Um das zu realisieren muss der generischen Netzwerktreiber entsprechend modifiziert werden.

3.3 Keine Systemaufrufe für den Zugriff auf Pakete

Es sollen möglichst keine Systemaufrufe während Capturing auftreten, denn die Systemaufrufe verursachen den Ausführungs-Kontextwechsel, was relativ zu den anderen Operationen, eine sehr große Menge von System-Ressourcen anfordert.

- Dafür müssen außer den Paket-Puffern noch andere Datenstrukturen mit den für das Capturing relevanten Daten in den Userspace eingeblendet werden, damit der User-Prozess ohne Systemaufrufe die Daten vom Treiber bekommt und auch Informationen an den Treiber liefern und damit das Capturing steuern kann.

- Systemaufrufe sind nur während der Initialisierung des Capturing-Prozesses und während der Abwesenheit von neuen Paketen für das Blockieren des auf die Pakete wartenden Prozesses erlaubt.

3.4 Transparenz

Die Funktionalität des neuen Capturing-Stacks soll für die standard Anwendungen möglichst unsichtbar sein. Das heißt, die standard Funktionen des Netzwerktreibers, Protokoll-Stacks, BPF's sollen nicht durch die Anwesenheit von *ringmap* gestört werden. Das Aktivieren des *ringmap* soll z.B. über ein Systemaufruf realisiert werden. Dabei im *ringmap*-Modus können Protokoll-Stack und standard Packet-Capturing-Stack deaktiviert werden. Das zurücksetzen des System in den default-Mode soll auch ohne Rekompilieren und ohne System-Neustart möglich sein.

[6] Siehe Manual mbuf(9)

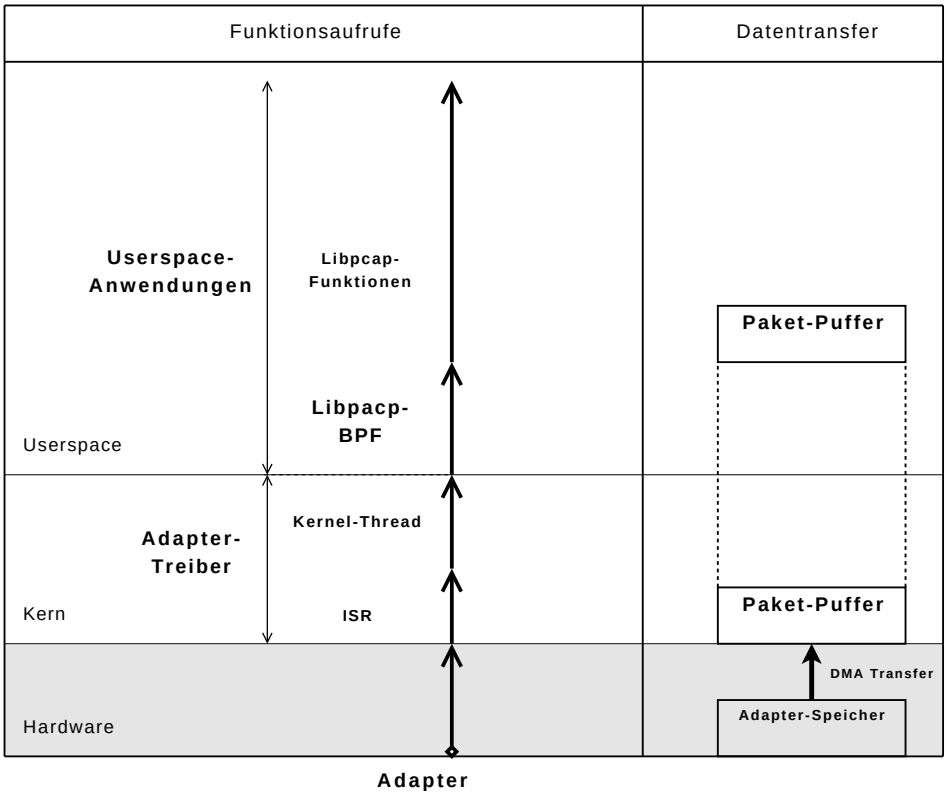

Abbildung 2: Der neue ringmap-Capturing-Stack. Ziel des Projektes

Da der *ringmap* anders als der standard FreeBSD-Capturing-Stack die empfangene Pakete den Userspace-Prozessen zur Verfügung stellen sollte, müssen die Userspace-Prozesse anders auf die Pakete zugreifen. Um der neue Stack einsetzbar zu machen soll am besten die libpcap-Library für ihn angepasst werden, sodass für alle Capturing-Anwendungen, die auf libpcap basieren, keine Änderungen benötigt werden.

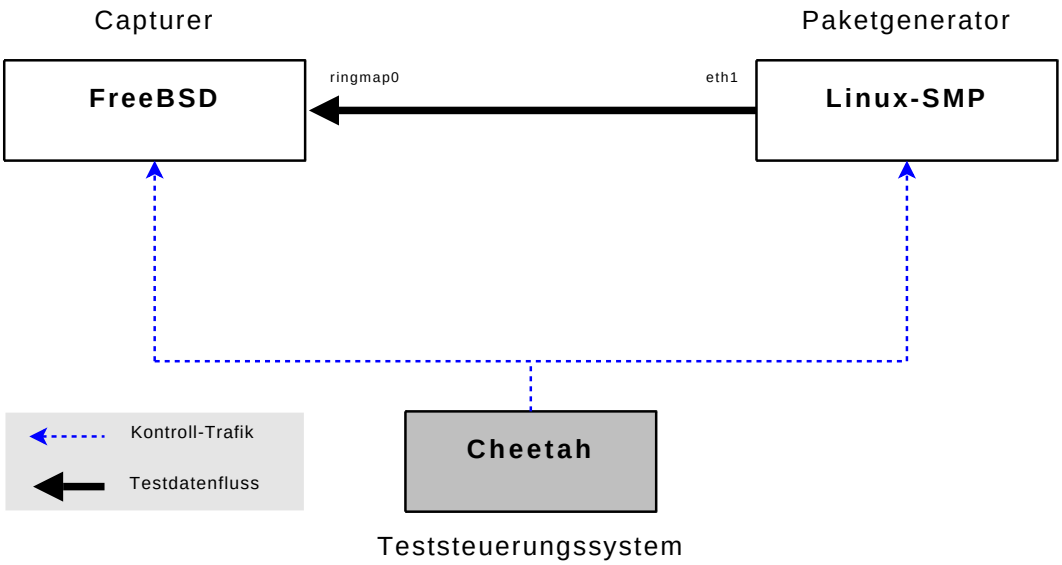

Abbildung 3: Netzwerk-Diagramm des Testsbeds

4 Leistungsbewertung

In diesem Kapitel werden die Ergebnisse des Leistungsvergleichs zwischen den *generischen* und den im Rahmen des Projektes entwickelten *ringmap*-Packet-Capturing-Stacks dargestellt. Für die Tests ist ein Netzwerk mit drei Hosts aufgebaut. Das Netzwerkdiagramm der Testsumgebung ist in Abbildung 3 dargestellt. Auf dem **Paketgenerator** ist Verkehr mit unterschiedlichen Charakteristiken erzeugt und auf dem **Capturer** erfasst. Dabei sind die Paketverluste und die Systemlast während der Datenerfassung gemessen.

4.1 Messaufbau

Im Folgenden beschreibe ich die für Tests eingesetzten Knoten (Abbildung 3):

- Paketgenerator
  - Ein leistungsfähigen Rechner zum Generieren des Test-Verkehrs.
  - OS: Linux-SMP
  - Linux Kernel Packet generator [11] wird für die Generierung des Netzverkehrs benutzt.
- Capturer
  - OS: FreeBSD mit *generic* und *ringmap*
    - FreeBSD-7.2, i386-Kernel (32 Bit)
    - Interrupt-Throttling default Parameter:
      - 8000 Interrupts pro Sekunde maximal
  - Hardware:
    1. FreeBSD-1:
       - **CPU:** AMD Athlon(tm) 64 Processor 2214.45-MHz
       - **Netzwerkadapter:** PCI, Intel Dual Port Gigabit Ethernet Controller
    2. FreeBSD-2:

    * **CPU:** 4 x Intel(R) Xeon(R) CPU 1.60GHz
    * **Netzwerkadapter:** PCIe, Intel HP NC360T PCIe DP Gigabit Server Adapter

- Teststeuerungssystem
  - OS: Linux
  - Scripte zur Teststeuerung.

4.1.1 Erzeugung von Verkehr

Der Verkehr für die Tests wird mit *Linux-Kernel-Packet-Generator*(*pktgen*) [11] erzeugt. *pktgen* ist ein Linux-Kernel-Module, das benutzt wird um die UDP-Pakete zu generieren und diese ins Netz zu senden. Für die Steuerung von *pktgen* wird das `/proc`-Filesystem benutzt.

Der *Linux-Kernel-Paket-Generator* wurde für die Experimente ausgewählt, weil er mehrere Vorteile mit dem Unterschied zu den anderen Software (z.B. Nemesis, Scapy, Iperf, etc...) für die Erzeugung des Netz-Verkehr bietet. Vor allem handelt es sich dabei um eine sehr hohe Paket-Rate, die mit *pktgen* erzeugt werden kann.

4.1.2 Messung der CPU-Auslastung und der Paketverluste beim Capturing

Auf dem FreeBSD-Host werden beim Test-Ablauf die auf dem Paketgenerator generierten und gesendeten Pakete erfasst. Dabei wird die Anzahl der empfangenen Pakete und die Systemload beim Capturing auf dem Capturer gemessen. Bei allen Experimenten werden die erfasste Pakete nicht auf die Festplatte geschrieben, sondern nur im RAM gezählt.

Messung von Paketverlusten

Für das Packet-Capturing wird eine einfache Anwendung implementiert, die für den Paketzugriff die Bibliothek *Libpcap* benutzt. In dieser Capturing-Anwendung ist ein *callback*-Funktion enthalten, die für jedes empfangene Paket aufgerufen wird, und derer Aufgabe ist es, die empfangene Pakete zu zählen. Das Paketverlust (P_{los}) wird als Differenz zwischen den Anzahl der empfangenen (P_{rcv}) und der gesendeten Pakete (P_{send}) berechnet:

$$P_{los} = P_{send} - P_{rcv} \tag{1}$$

Messung von CPU-Auslastung

Die CPU-Last wird auf FreeBSD über die *sysctl*-Variable `kern.cp_time` abgefragt:

```
% sysctl kern.cp_time
kern.cp_times: 27281 1333 301046 513 1001093319
%
```

Bei der Abfrage der `kern.cp_time`-Variable werden auf dem Standard-Output die Zeiten ausgegeben, welche die CPUs seit dem Start des Betriebssystem jeweils im `user`-, `nice`-, `syst`-, `intr`- und `idle`-Modus verbracht haben.

Beim Präsentieren der Tests-Ergebnisse wird aber nicht die volle CPU-Last, die aus den `user- nice- syst- intr- idle`-Load besteht, sondern lediglich die **Systemload** (`syst`) angegeben. Da die Implementierung des neuen Stack hauptsächlich Kernel-Code betrifft,

interessiert uns vor allem die Systemload (`syst`)[7]. Der Absicht der Tests ist das Prüfen, ob unsere Entwurf-Ansätze korrekt sind, und ob sie zu dem gewünschten Ziel führen. Anders gesagt, zu prüfen, ob die Ausführung des Kernel-Codes vom neuen *ringmap*-Capturing-Stack eine niedrigere Systemlast als beim *generic*-Capturing-Stack verursacht.

4.2 Ergebnisse

In diesem Kapitel werden die Ergebnisse der im Rahmen des Projektes durchgeführten Experimente dargestellt. Der erste Abschnitt stellt die Ergebnisse der Experimente mit dem neuen *ringmap*-Packet-Capturing-Stacks dar. Es werden die Systemload und Paketverluste beim Capturing in Abhängigkeit von der Date-Rate des Verkehrs und der anderen Parameter dargestellt.

Im Abschnitt 4.2.2 vergleichen wir die Performance des ringmap- mit dem generic-Capturing-Stack, um den Performance-Gewinn durch den neuen *ringmap*-Capturing-Stack quantitativ zu bestimmen.

4.2.1 Ringmap-Paket-Capturing-Stack

In diesem Abschnitt sind die Ergebnisse der Experimenten mit *ringmap*-Capturing-Stack dargestellt. Das Ziel der Experimenten die Capturing-Performance des *ringmap*-Stacks in Abhängigkeit von den folgenden Parameter herauszufinden:

- Paket-Ringpuffer-Größe
- Daten-Rate des erfassten Verkehr.

Paketverluste in Abhängigkeit von der Anzahl der Slots im Paket-Ringpuffer

Das Ziel dieses Experiments ist es, die Abhängigkeit der Paketverluste während Capturing von der Größe des Paket-Ringpuffers (Ring-Buffer) herauszufinden. In diesem Experiment ist eine Reihe von Tests durchgeführt. Für alle Tests ist der Verkehr mit den kleinsten 64-Bytes Pakete und mit der maximal erreichbaren Bit-Rate (c.a. $696 Mbit/sec$) generiert.

- Konfiguration auf dem Capturer:
  - **Hardware:** FreeBSD-2 (PCIe)
  - **Betriebssystem:** FreeBSD **7.2**, **non-SMP Kernel**
- Verkehrsparameter:
  - Paketlänge: 64-Bytes
  - Paketmenge: 15000000
  - Bit-Rate: etwa $696 MB/sec$

Paketverluste: In Abbildung 4 sind die Paketverluste während Capturing dargestellt. Auf der X-Achse wird die Anzahl der Slots im Ring-Buffer dargestellt. Auf der Y-Achse die prozentuelle Anzahl der Paketverluste. In allen durchgeführten Experimenten mit 256 oder mehr Paket-Slots sind die Paketverluste relativ konstant und liegen unter 0.02%, d.h. die Erhöhung der Anzahl der Slots ab 256 bringt keinen weiteren Gewinn mehr.

[7] Aufgrund der default Treiber-Einstellungen für Interrupt-Moderation, bleibt intr-Load konstant. Deshalb interessiert uns Interrupt-Load auch nicht

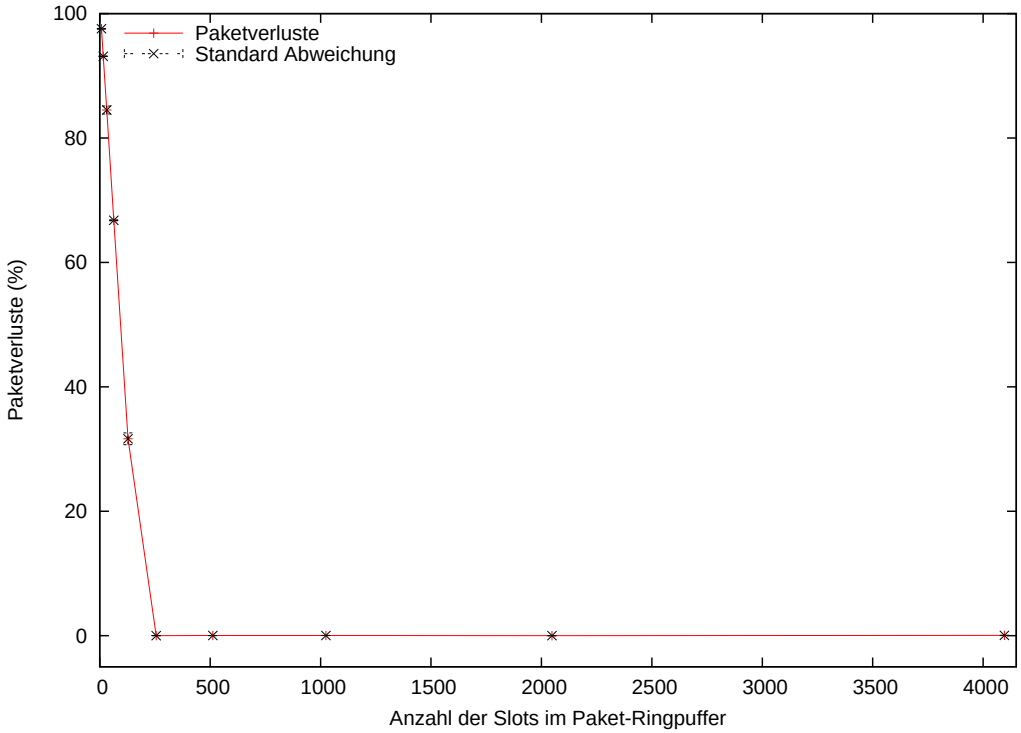

Abbildung 4: Paketverluste in Abhängigkeit von der Anzahl der Slots im Paket-Ringpuffer

Performance in Abhängigkeit von Paketlänge, Bit-Rate und Paket-Rate

Das Ziel des Experiments ist es, die Abhängigkeit der Capturing-Performance von der Paketsgröße und Daten-Rate im Netzverkehr herauszufinden.

- Konfiguration auf dem Capturer:
  - **Hardware:** FreeBSD-2
  - **Betriebssystem:** FreeBSD **7.2**, **non-SMP Kernel**
  - **Treiber:**
    - Paket-Ringpuffer-Größe: 1024 Slots
- Verkehrsparameter:
  - Paketlängen: 64- , 200-, 300-Bytes
  - Paketmengen: 15000000

Paketverluste: Bei allen Experimenten ergibt sich für Paketgrößen über 200 Bytes die Paketerfassungsrate 100%. Nur in den Experimenten mit der kleinsten Paketgröße von 64-Bytes und nur bei der höchsten erreichte Bit-Rate von $627 MB/sec$ ergibt sich ein sehr kleines Paketverlust geringer als 0.02% aller generierten Pakete.

Systemload: Die Ergebnisse der Systemload-Messung werden in den Abbildungen 5 und 6 dargestellt. Die maximal erreichte Systemload beim Capturing in allen Experimenten liegt unter 12% und wird erreicht beim Capturing des Verkehrs mit den kleinsten 64-Bytes-Paketen.

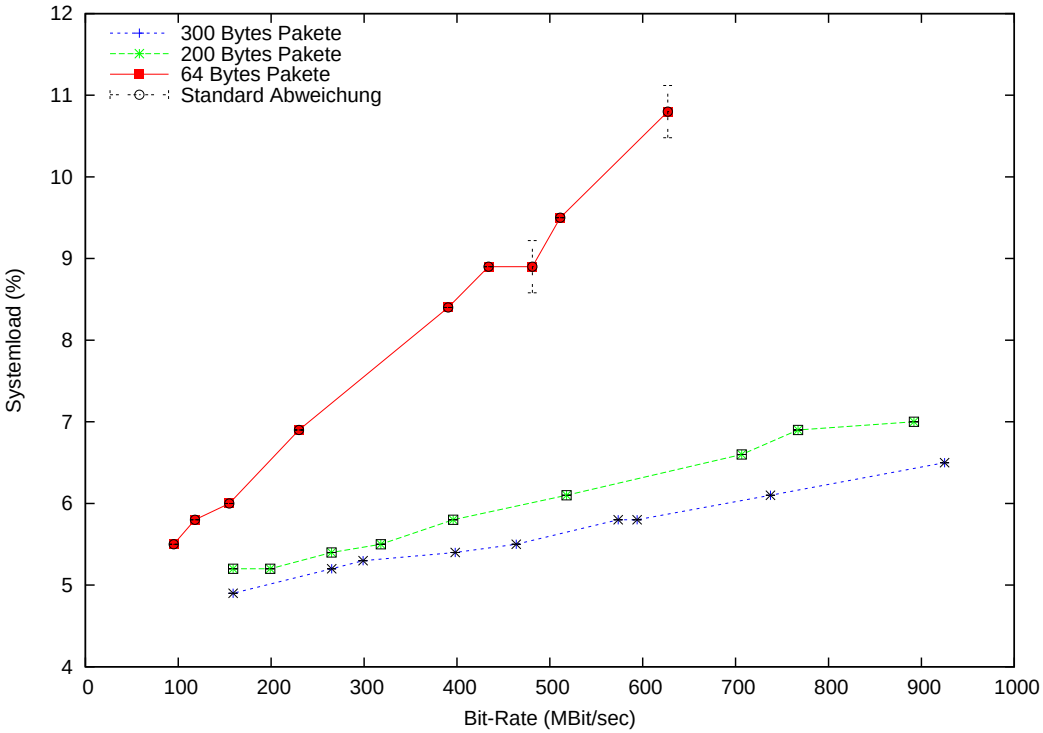

Abbildung 5: Systemload in Abhängigkeit von Bit-Rate beim Capturing

In Abbildung 6 wird die Systemload in Abhängigkeit von der Paket-Rate dargestellt. Bei diesen Ergebnissen kann man deutlich sehen, dass die Systemload beim Capturing von der Paket-Rate und nicht von der Paket-Größe beeinflusst wird. Drei Verkehrsströme mit unterschiedlichen Paketgrößen verursachen fast identisch gleiche Systemload (die Unterschiede sind geringer als 0.5%) wenn die Paketraten in diesen drei Verkehrsströmen gleich sind. Dies lässt sich einfach erklären. Im neuen *ringmap*-Treiber wurden alle Paket-Kopie-Operationen und Speicherallozierungen entfernt. Der Userspace-Prozess bekommt den Zugriff auf die Pakete sofort nach dem DMA-Transfer. Aus diesen Gründen wird beim Capturing mit dem *ringmap*-Treiber im Kernelspace die geringstmögliche Arbeit erledigt, die pro Paket-Puffer skaliert und nicht von der Paket-Größe abhängt.

4.2.2 Vergleich generic- mit ringmap-Paket-Capturing-Stack

Das Ziel dieses Experiments ist es, die Capturing-Performance von *ringmap*-Capturing-Stack mit dem *generic*-Capturing-Stack zu vergleichen.

Konfiguration auf dem Capturer:

- **Hardware:** FreeBSD-2
- **Betriebssystem:** FreeBSD **7.2**, **non-SMP Kernel**
- **Treiber:**
  - Paket-Ringpuffer-Größe: 1024 Slots
  - BPF-Puffer-Größe: 10MB

Verkehrsparameter:

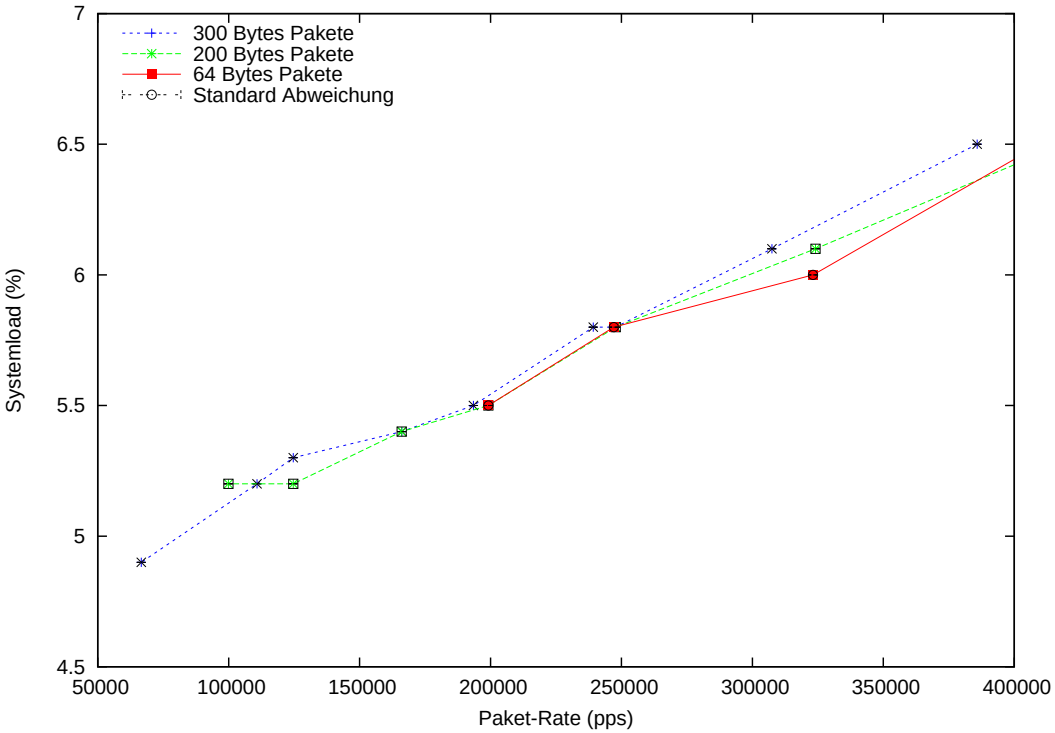

Abbildung 6: Systemload in Abhängigkeit von Paket-Rate beim Capturing

- Paketlängen: 64-, 200-, 300-Bytes
- Paketmengen: 5000000-, 10000000-, 15000000 Pakete

Paketverluste: In Abbildung 7 sind die Paketverluste dargestellt. Auf der X-Achse wird die generierte Datenrate dargestellt. Auf der Y-Achse der prozentuelle Anzahl der Paketverluste. Bei allen Tests entstanden die Paketverlust bei *generic*-Stack lediglich für Pakete der Größen 200 und 64 Bytes. Dabei kann der *generic*-Stack für die 64-Bytes-Pakete bei der Bit-Rate von 393Mbit/s und höher (entsprechende Paket-Rate > 819545) nur eine konstante Anzahl von Paketen erfassen, etwa 262147 Pakete, unabhängig von der generierte Paketmenge.

Systemload: Die Ergebnisse der Systemload-Messung sind in Abbildung 8 dargestellt. Auf X-Achse wird die generierte Datenrate dargestellt. Auf der Y-Achse die Systemload. Bei Erfassung der Verkehr mit dem *generic*-Stack liegt die Systemload wesentlich höher als beim *ringmap*-Stack. Dies erklärt sich dadurch, dass der *generic*-Stack für jedes Paket mehrere Kopie-Operationen im Kernelspace ausführt und für den Paketzugriff von der Userspace einen Systemaufruf verwendet. Bei hohen Daten-Raten ist diese Vorgehensweise des *generic*-Stack nicht mehr effizient.

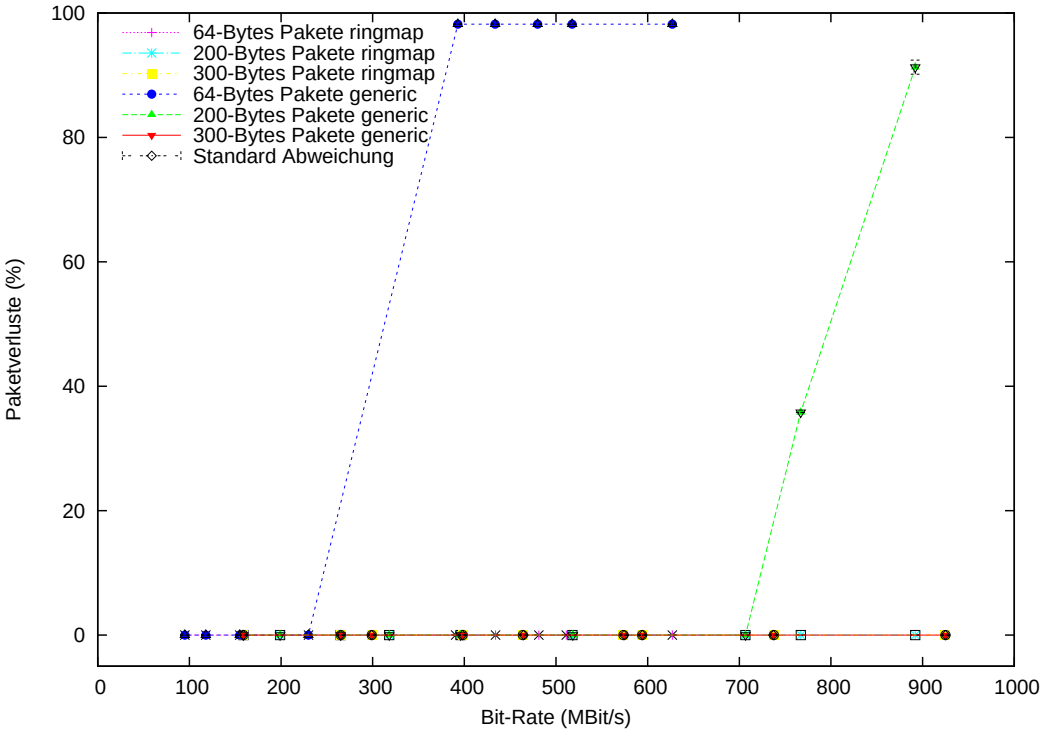

Abbildung 7: Paketverluste in Abhängigkeit von Bit-Rate beim Capturing. Vergleich ringmap- und generic-Packet-Capturing-Stacks

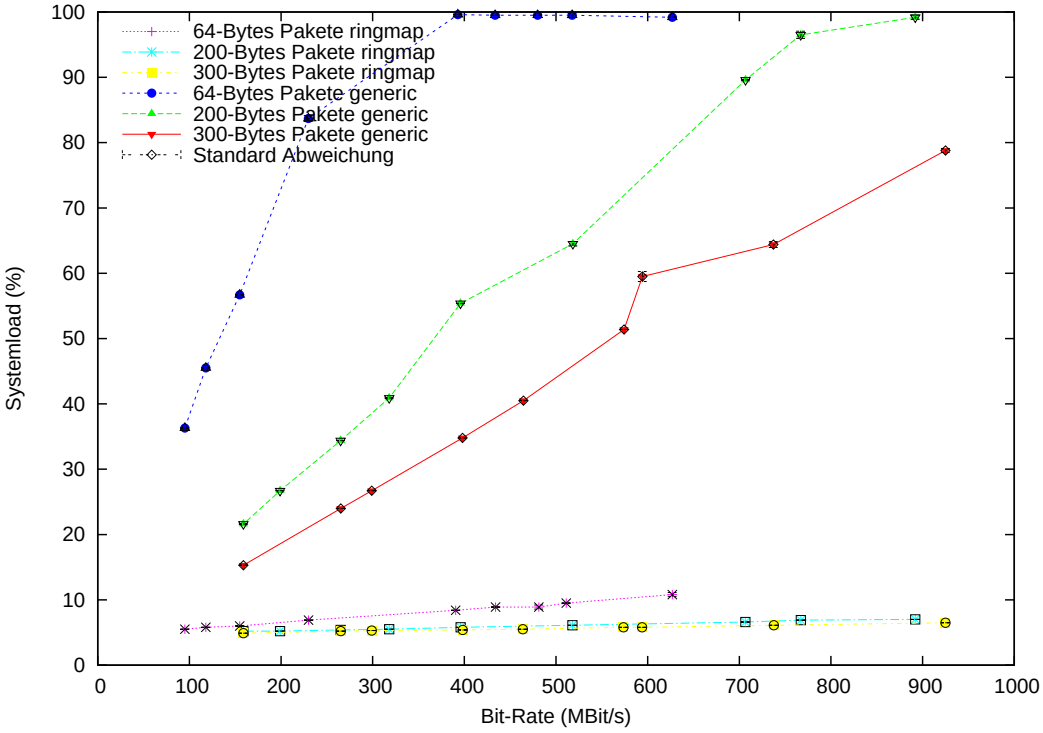

Abbildung 8: Systemload in Abhängigkeit von Bit-Rate beim Capturing. Vergleich ringmap- und generic-Packet-Capturing-Stacks

5 Zusammenfassung

Das Ziel des Projektes ist es, die für Capturing relevante Software des Betriebssystem FreeBSD zu untersuchen, die Probleme, die zur Reduzierung der Capturing-Performance führen, herauszufinden und aufgrund der gefundenen Problemen den neuen (ringmap)-FreeBSD-Packet-Capturing-Stack zu erarbeiten, zu implementieren und auszuwerten. Die neue Lösung basiert auf der für das Projekt vorausgesetzte Hardware (siehe Abschnitt 1.3).

5.1 Erreichte Ziele

5.1.1 Verbesserung der Capturing-Performance

Beim Einsatz des im Rahmen des Projektes implementierten *ringmap*-Stacks wird die Capturing-Performance unter gleichen Bedingungen höher als beim *generic*-Stack4.2.2. Die **Systemload** beim Capturing mit dem *ringmap*-Stack ist deutlich geringer als mit dem *generic*-Stack. Bei allen durchgeführten Experimenten war die Systemload beim Capturing mit dem *ringmap* kleiner als 12%. Die Systemload bei der Verwendung von *generic*-Stack variiert von 13% bis 100%. Bezüglich der Systemload zeigt der *ringmap* einen deutlichen Gewinn (siehe Abschnitt 4.2.2).

Anders sieht es mit den **Paketverlusten** aus. Beide Stacks zeigen identisch gute (100%) Paketerfassungsrate für den Fall wenn die Paketgröße über 200 Bytes liegt. Bei Paketen kleiner als 200 Bytes, und daraus verursachten höheren Paketraten, $> 450000 pkts/sec$, verliert der *generic*-Stack (mit 10MB BPF-Puffer-Größe) bis zu 100% aller der generierten Pakete. Selbst bei maximaler Belastung verliert der *ringmap*-Stack weniger als 0.02% der Pakete.

5.1.2 Transparenz

Alle auf *libpcap* basierte Anwendungen wie *tcpdump, wireshark, etc...* benötigen keine Änderungen um mit dem *ringmap*-Stack die Pakete zu erfassen.

5.2 Einschränkungen des ringmap-Packet-Capturing-Stack

Die Benutzung der *ringmap*-Software verursacht auf dem Capturing-System (bzw. für die Ausgewählte Adapter) folgende Einschränkungen:

Kein TCP/IP Protokollstack während Capturing: Für das Capturing mit dem *ringmap* muss das Netzwerk-Interface in Monitoring-Mode gesetzt werden. Aus diesem Grund besteht keine Möglichkeit, das gleiche Interface gleichzeitig für Capturing und Kommunikation zu nutzen.

Libpcap Einschränkungen: Für das Anpassen der Libpcap an den *ringmap*-Stack wurden einige Code-Stücke im Libpcap-Quell-Code modifiziert. Dadurch ist die Funktionalität der neuen *ringmap*-Libpcap ist nicht mit der *generic*-Libpcap identisch. Der `to_ms`-Parameter [5] für die `pcap_open_live()`-Funktion ist deaktiviert, aber aus Kompatibilitätsgründen geblieben.

Alle genannte Einschränkungen sind in dem Sinne nicht kritisch, dass sie sich eliminieren oder, für bestimmte Anforderungen, anpassen lassen. Der Source-Code von *ringmap* ist auf Google-Code veröffentlicht und unter dem Namen *ringmap* auf dem Server zu finden. An der zukünftigen Weiterentwicklung des Projektes habe ich großes Interesse und biete meine Hilfe an.

5.3 Zukünftige Themen

5.3.1 Performance-Vergleich mit dem Zero-Copy-BPF-Buffers

Im Lauf der Test-Phase des Projektes, war der *Zero-Copy-BPF-Buffers* Projekt noch im alpha-Stadium und nicht bereit für das Testen (siehe Abschnitt 1.4). Inzwischen ist der Zero-Copy-BPF-Buffers in der neusten FreeBSD-8.0-Version vorhanden und soll stabil sein. Daher soll die Auswertung von Zero-Copy-BPF-Buffers der nächste Schritt sein.

5.3.2 10Gbit Capturing

Die moderne 10GbE Netzwerkadapter besitzen auf dem Chip mehrere *queues* für die ankommende und gesendete Pakete. Sie sind auch in der Lage die Trafik zu filtern und die Pakete mit den spezifischen Mustern zu einer oder mehreren *queues* zu zuordnen.

Der weitere Schritt des Projektes ist die Erweiterung der ringmap-Funktionalität für die Anwendung auf 10GbE *multi-queue* Controller.

Literatur

[1] FreeBSD Developer Summit presentation describing Zero-copy BPF. http://www.watson.org/~robert/freebsd/2007asiabsdcon/20070309-devsummit-zerocopybpf.pdf.

[2] Wiki page: Berkeley packet filter. http://en.wikipedia.org/wiki/Berkeley_Packet_Filter.

[3] Intel. Interrupt moderation using intel GbE controllers. Technical report, Intel, 2007.

[4] Intel. *PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer's Manual*, 4.0 edition, 2009.

[5] Van Jacobson, Craig Leres, and Steven McCanne. *FreeBSD Subroutines: pcap - Packet Capture library*, 2004. http://www.freebsd.org/cgi/man.cgi?query=pcap&apropos=0&sektion=3&manpath=FreeBSD+7.2-RELEASE&format=html.

[6] Steven McCanne and Van Jacobson. *FreeBSD Kernel Developers Manual: Berkeley Packet Filter*, 2007. http://www.freebsd.org/cgi/man.cgi?query=bpf&apropos=0&sektion=0&manpath=FreeBSD+7.2-RELEASE&format=html.

[7] Steven McCanne and Van Jacobson. *FreeBSD Kernel Interfaces Manual: Berkeley Packet Filter*, 2007. http://www.freebsd.org/cgi/man.cgi?query=bpf&apropos=0&sektion=0&manpath=FreeBSD+7.2-RELEASE&format=html.

[8] Jeffrey Mogul and K.K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15:217–252, 1997. http://portal.acm.org/citation.cfm?id=263335.

[9] Fabian Schneider. Performance evaluation of packet capturing systems for high-speed networks. Master's thesis, Technische Universität München, 2005. http://www.net.t-labs.tu-berlin.de/papers/S-PEPCSHN-05.pdf.

[10] Fabian Schneider, Jörg Wallerich, and Anja Feldmann. Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. *Lecture Notes in Computer Science*, 4427:207–217, 2007. http://www.net.t-labs.tu-berlin.de/papers/SWF-PCCH10GEE-07.pdf.

[11] Uppsala Universität. *pktgen the linux packet generator*, 2004. ftp://robur.slu.se/pub/Linux/net-development/pktgen-testing/pktgen_paper.pdf.

Listings