Karlsruher Institut für Technologie

Frequenz-basierter Covert Channel mithilfe der Intel Turbo Boost Technology

Masterarbeit
von

Manuel Kalmbach B.Sc.

an der Fakultät für Informatik

Erstgutachter:	Prof. Dr. Frank Bellosa
Zweitgutachter:	Prof. Dr. Wolfgang Karl
Betreuender Mitarbeiter:	Mathias Gottschlag M.Sc.

Bearbeitungszeit: 06. November 2018 – 05. Mai 2019

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 5. Mai 2019

Abstract

Bisherige frequenzbasierte Covert Channel basieren auf dem Prinzip, dass entweder alle CPU-Kerne die gleiche CPU-Frequenz verwenden oder mithilfe des Betriebssystems, welches Schnittstellen bietet um die CPU-Frequenz eines anderen CPU-Kerns auszulesen. Durch Änderungen in aktuellen Mikroarchitekturen verwendet heute jedoch jeder CPU-Kern seine eigene CPU-Frequenz. Zudem kann das Betriebssystem den Zugriff auf die Information der CPU-Frequenz anderer CPU-Kerne einschränken und damit den Covert Channel verhindern.

Aufgrund dieser Punkte wurde es das Ziel dieser Arbeit einen Covert Channel zu entwickeln, welcher über die CPU-Frequenz zwischen zwei Prozessen auf einem Hostsystem kommuniziert. Dabei wurde speziell darauf geachtet, dass diese Prozesse keine gesonderten Rechte brauchen und auch nicht auf Informationen des Betriebssystems angewiesen sind. Um die CPU-Frequenz zu beeinflussen wird dabei die Intel Turbo-Boost Technologie verwendet, welche das automatische Übertakten von CPU-Kerne erlaubt. Dabei ist die von Turbo-Boost festgelegte Turbofrequenz auf allen CPU-Kernen gleich, weswegen es möglich ist, hierüber Daten zu übertragen. Durch Reduzieren von Störungsquellen, sind Datenrate von 1000 bit/s möglich, was jedoch mit einer Fehlerrate von 13,5% verbunden ist. Bei einem System im Leerlauf wurde Nutzdatenraten von 56 bit/s erreicht. Neben der Kommunikation zwischen zwei Prozessen ist es auch möglich diesen Covert Channel zwischen zwei virtuellen Maschinen einzusetzen. Hierbei wurden Nutzdatenraten von bis zu 9 bit/s erreicht. Im Vergleich zu bisherigen frequenzbasierten Covert Channels schneidet dieser deutlich besser ab. So erreichen vergleichbare Covert Channels Nutzdatenraten von bis zu 20 bit/s, wobei diese auf die Mithilfe des Betriebssystems angewiesen sind.

Inhaltsverzeichnis

Kapitel 1

Einführung

Daten sind Rohstoffe des 21. Jahrhunderts [34]. Es werden nicht nur immer mehr Daten gesammelt, es entstehen auch immer mehr Daten, welche vertraulich sind und nicht an die Öffentlichkeit gelangen sollten. Damit das nicht passiert, werden immer mehr Vorkehrungen zum Schutz dieser Daten getroffen. Programme, welche Zugriff auf vertrauliche Daten haben, werden isoliert, um unerlaubte Zugriffe auf das Internet oder ähnliches zu verhindern, bei denen diese Daten verloren gehen können. Nicht immer gelingt dies. Neben den allgemein bekannten Kommunikationswegen, wie Interprozesskommunikation, Netzwerkkommunikation, etc., gibt es noch weitere, nicht offensichtliche Kommunikationswege, über die ein isoliertes Programm trotzdem mit der Außenwelt kommunizieren kann.

Solche verdeckten Kommunikationskanäle werden Covert Channels genannt. Der Datenaustausch erfolgt hierbei über Kommunikationskanäle, die nicht von Sicherheitsrichtlinien erfasst und kontrolliert werden. Systeme wie beispielsweise das Betriebssystem, welche die Sicherheitsrichtlinien umsetzen, können dementsprechend diese auch nicht verhindern.

Es gibt viele unterschiedliche Arten, wie sich Covert Channels realisieren lassen. Meist werden Nebeneffekte von bestehenden Kommunikationskanälen oder Hardwareeigenschaften zum Aufbau eines Kommunikationskanals ausgenutzt. So gibt es Covert Channels, welche den TCP/IP-Header als Übertragungsweg nutzen [31] oder über die Unterschiede von Cache-Zugriffszeiten Daten austauschen [28]. In dieser Arbeit wird ein Covert Channel vorgestellt, welcher die CPU-Frequenz als Kommunikationskanal benutzt.

Die CPU-Frequenz als Kommunikationskanal wurde bisher weniger betrachtet, da sie einige Schwierigkeiten mit sich bringt. So haben sich in den letzten Jahren Hardwareänderungen ergeben, welche ein Ausnutzen der CPU-Frequenz erschweren. Durch die bei Skylake eingeführten Hardwareänderungen hat jeder CPU-Kern eine von den restlichen CPU-Kernen unabhängige CPU-Frequenz. In vorherigen Mikroarchitekturen hatten alle CPU-Kerne die gleiche CPU-Frequenz,

wodurch es möglich war, wie von Algappan et al. beschrieben, durch Auslastung eines CPU-Kerns das Betriebssystem dazu zu bringen die gemeinsame CPU-Frequenz aller CPU-Kerne zu ändern [8]. Ein Empfänger konnte daraufhin die CPU-Frequenz des einen CPU-Kerns ermitteln und damit die Informationen empfangen. Seit Skylake kann jeder CPU-Kern seine eigene Frequenz selbst bestimmen, was das Auslesen zwischen CPU-Kernen verhindert. Der Empfänger musste daraufhin auf Schnittstellen des Betriebssystems ausweichen, welche es erlauben die CPU-Frequenz aller CPU-Kerne auszulesen. Die Kontrolle über diese Schnittstellen hat das Betriebssystem, was dazu führt, dass dieses den Zugriff verweigern oder falsche Informationen liefern kann, um den Empfänger zu täuschen.

Des Weiteren kommt hinzu, dass die Auslastung des Systems Einfluss auf die Übertragungsfehler innerhalb des Kommunikationskanals haben. Daher ist der in dieser Arbeit beschriebene Covert Channel darauf ausgelegt, dass das System wenig ausgelastet ist.

Mithilfe der Intel Turbo-Boost Technologie lässt sich dieses Problem umgehen. Sie erlaubt das dynamische Übertakten einzelner CPU-Kerne. Dabei wird von Turbo-Boost in Abhängigkeit verschiedener Metriken eine globale Turbofrequenz bestimmt, welche für alle CPU-Kerne gilt. Damit hierbei eine Übertragung zustande kommt, nimmt der Sender Einfluss auf die Metriken von Turbo-Boost und dadurch Einfluss auf die Turbofrequenz. Der Empfänger selbst kann wieder durch das Zählen von Schleifendurchläufen die aktuelle Turbofrequenz bestimmen und somit Daten unabhängig vom Betriebssystem empfangen.

Auf Basis dieser Ansätze basiert der in dieser Arbeit beschriebene frequenzbasierte Covert Channel. Dabei werden mithilfe der Turbo-Boost Technologie Informationen betriebssystemunabhängig zwischen zwei Prozessen ausgetauscht. Durch ein Übertragungsprotokoll wird zusätzlich versucht durch Error-Correction-Codes und Sendungswiederholungen Übertragungsfehler zu minimieren.

Mit dem Covert Channel können bis zu 1000 bit/s erreicht werden, solange keine Störungen auftreten. Mit dem Übertragungsprotokoll und unter realen Bedingungen sind Nutzdatenraten von bis zu 56 bit/s möglich. Bei der Kommunikation zwischen virtuellen Maschinen war eine Nutzdatenrate von bis zu 9 bit/s möglich.

Diese Arbeit gliedert sich in mehrere Unterkapitel. Zu Beginn wird in Kapitel 2 auf die Grundlagen eines Covert Channels eingegangen, sowie auf die Architektur und ihre Besonderheiten, welche für in dieser Arbeit benötigt werden. Anschließend wird in Kapitel 3 der grundlegende Aufbau des Covert Channels mit dazugehörigem Übertragungsprotokolls beschrieben. Auf Basis dieses Entwurfs wird anschließend in Kapitel 4 die umgesetzte Implementierung erläutert und auf ihre Besonderheiten eingegangen. In Kapitel 5 wird sich mit den möglichen Gegenmaßnahmen zur Verhinderung des Covert Channels auseinandergesetzt. Anschließend wird in Kapitel 6 die Implementierung untersucht und aufge-

zeigt, welche Datenübertragungsraten mit ihr möglich sind. In Kapitel 7 wird anschließend analysiert, wie sinnvoll dieser Covert Channel ist und welche Stärken und Schwächen er im Vergleich zu bisherigen frequenzbasierten Covert Channels hat. Abschließend wird in Kapitel 8 ein Fazit über den Covert Channel gezogen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen eines Covert Channels sowie Hardwaredetails besprochen, welche später für den Entwurf und die Implementierung benötigt werden. Zu Beginn wird in Abschnitt 2.1 beschrieben, was ein Covert Channel im Allgemeinen ist und wie ein frequenzbasierter Covert Channel aufgebaut ist. Anschließend wird ein sich daraus ergebendes Angriffsszenario vorgestellt, die Eigenschaften der eingesetzten Hardware erklärt und näher auf die Funktionsweise der Intel® Turbo-Boost Technology eingegangen. Am Ende wird in Abschnitt 2.4 und 2.5 kurz auf die Grundlagen des Reed-Solomon-Codes und von TCP eingegangen.

2.1 Covert Channel

In dem Buch „Trusted Computer System Evaluation Criteria“ (auch „The orange Book“ genannt) des United States Government Department of Defense wird ein Covert Channel als ein Kommunikationskanal beschrieben, welcher von Prozessen ausgenutzt werden kann, um Nachrichten auszutauschen, die der Systemsicherheitsrichtlinie widersprechen [42, S. 81]. Dabei arbeitet ein Covert Channel verdeckt von jeglichen Sicherheitsfunktionen und benutzt selten herkömmliche Kommunikationskanäle wie eine Netzwerkverbindung oder einen seriellen Anschluss.

Das Sicherheitspotential eines Covert Channels wird dabei anhand der Bandbreite eingeordnet. Ab 100 bit/s wird das Sicherheitspotential als hoch und bei weniger als 1 bit/s als gering eingestuft. Diese Einstufung ist jedoch mit Vorsicht zu betrachten. Je nach Größe der Daten, auf die es ein Angreifer abgesehen hat, kann 1 bit/s viel oder wenig sein. Auch bei größeren Daten kann eine geringe Bandbreite ausreichend sein, wenn genügend Zeit für die Übertragung vorhanden ist. Wie groß das Potenzial eines Covert Channels ist, muss daher für jeden Fall

neu eingeschätzt werden. Für den allgemeinen Fall ist diese Einteilung jedoch ein gutes Mittel, um den Covert Channel zu klassifizieren.

Es gibt eine Vielzahl an Covert Channel, welche unterschiedlichste Medien als Übertragungskanal verwenden. Dabei können unter anderem klassische Methoden wie das von Ji et al. beschriebenen Verfahren zum Verstecken von Nachrichten in normalem Netzwerk-Traffic [24] oder auch etwas ausgefallenere Methoden wie beispielsweise das von Castiglione et al. beschriebenes Verfahren für einen Covert Channel über Spam-Mail [13] zum Einsatz kommen. Neben den Covert Channels über Netzwerkprotokolle gibt es auch welche, die Hardwareeigenschaften ausnutzen, um zwischen zwei Prozessen eines Systems zu kommunizieren. Ein Beispiel hierfür ist das von Maurice et al. beschriebene Verfahren, bei dem der Cache als Covert Channel verwendet wurde [28]. Auch andere Hardwareeigenschaften wie etwa die Prozessortemperatur können, wie in dem Verfahren von Bartolini et al. beschrieben, als Covert Channel verwendet werden [10].

Der in dieser Arbeit beschriebene Covert Channel basiert darauf, dass durch Einflussnahme auf die CPU-Frequenz Daten übertragen werden können. Dies ist jedoch nicht der erste Covert Channel, welcher auf der CPU-Frequenz basiert. Es gibt den von Cioranesco et al. vorgestellten Covert Channel, bei dem die CPU-Frequenz eines CPU-Kerns durch Auslastung dessen beeinflusst wird und anschließend über die vom Betriebssystem bereitgestellte Sysstat-Schnittstelle ausgelesen wird [15].

Auch Algappan et al. haben sich mit einem CPU-Frequenz Covert Channel beschäftigt und sich dabei mehrere Herangehensweisen der Umsetzung eines Senders und Empfängers überlegt [8]. Als Ansatz für einen Sender wurden, neben dem einfachen Auslasten eines CPU-Kerns, auch das Ausnutzen der unterschiedlichen Frequenzstufen eines Prozessors betrachtet. Zum Auslesen der CPU-Frequenz wurden, neben den vom Betriebssystem über Sysstat bereitgestellten Informationen, auch die Möglichkeit der Berechnung über die Anzahl der Schleifendurchläufe oder das Auslesen des Model Specific Register betrachtet.

Die zu diesem Zeitpunkt aktuellste Veröffentlichung zum Thema frequenzbasierter Cover Channel stammt von Miedl et al [30]. Der von ihnen beschriebene Ansatz versucht, durch Auslastung des Systems den Frequenz-Governor des Betriebssystems dazu zu bewegen die CPU-Frequenz wie gewünscht zu ändern. Die vom Sender auf diesem Weg veranlasste Frequenzänderung wird vom Empfänger erkannt und nach Ende der Übertragung von einem Neuronalen Netzwerk analysiert.

Alle bisher vorgestellten Covert Channel benötigen entweder die Kooperation des Betriebssystems zum Auslesen der CPU-Frequenz oder basieren auf dem Prinzip, dass alle CPU-Kerne in die gleiche Frequenzdomäne haben. Bei heutigen Prozessoren besitzt jedoch jeder CPU-Kern eine eigene Frequenzdomäne. Dies führt dazu, dass die darauf basierenden Covert Channel nicht mehr funktionieren.

Covert Channel, die die CPU-Frequenz über Schnittstelle des Betriebssystems auslesen sind auf dessen Mithilfe angewiesen und daher leicht zu unterbinden. Ziel dieser Arbeit ist es daher Betriebssystem unabhängig zu sein und Mithilfe von Turbo-Boost die CPU-Frequenz anderer CPU-Kerne zu beeinflussen und somit einen Kommunikationskanal zwischen CPU-Kernen aufzubauen.

2.2 Angriffsszenario

In dieser Arbeit werden zwei Angriffsszenarien betrachtet. Bei dem ersten Angriffsszenario gibt es auf einem System zwei Prozesse A und B, in welche der Angreifer eigenen Code eingeschleust hat. Die Systemsicherheitsrichtlinie erlaubt es, dass Prozess A Zugriff auf das Internet hat, Prozess B jedoch nicht. B hingegen hat Zugriff auf vertrauliche Daten, welche nicht in die falschen Hände kommen dürfen. Zudem ist es den beiden Prozessen nicht erlaubt, miteinander zu kommunizieren. Für den Angreifer ergibt sich die Situation, dass er theoretisch über den infiltrierten Prozess B Zugriff auf die vertraulichen Daten hat, diese aber nicht ansehen kann, da er keine Möglichkeit hat, von außen mit Prozess B zu kommunizieren. Prozess A hingegen steht über den Internetzugang in direktem Kontakt zu dem Angreifer. Durch Zuhilfenahme eines Covert Channels, welcher die Systemsicherheitsrichtlinie umgeht, kann der Angreifer einen Kommunikationskanal zwischen beiden Prozessen aufbauen und so an die vertraulichen Daten gelangen.

Das zweite Angriffsszenario ist entsprechend analog zu dem ersten. Hierbei laufen beide Prozesse in getrennten virtuellen Maschinen auf dem gleichen Hostsystem. Bei diesem Szenario erfolgt die Kommunikation des Covert Channels nicht zwischen zwei Prozessen auf einem System, sondern zwischen zwei Prozessen innerhalb unterschiedlicher virtueller Maschinen. In modernen Rechenzentren wird vermehrt auf Virtualisierung gesetzt, um eine höhere Auslastung auf den Systemen zu erreichen [32]. Dennoch liegt die Auslastung von Systemen in Rechenzentren im Tagesdurchschnitt bei nur 6% [23].

2.3 Architektur

Bei dem, in der Evaluation verwendeten Prozessor, handelt es sich um einen Intel® Xeon® Silver 4108. Dieser Prozessor basiert auf der Skylake Mikroarchitektur, welche in Abschnitt 2.3.1 näher vorgestellt wird. Unter anderem besitzt diese Architektur einige Neuerungen, welche dazu führen, dass die bisherigen frequenzbasierten Covert Channel nur noch eingeschränkt und mit der Kooperation des Betriebssystems funktionieren.

Für die spätere Evaluierung und Implementierungen wurden zusätzliche die in Abschnitt 2.3.4 beschriebene RAPL-Schnittstelle, mit der der Energieverbrauch bestimmt werden kann, und die in Abschnitt 2.3.5 beschriebene Instruktion zur Frequenzmessung verwendet.

2.3.1 Skylake Mikroarchitektur

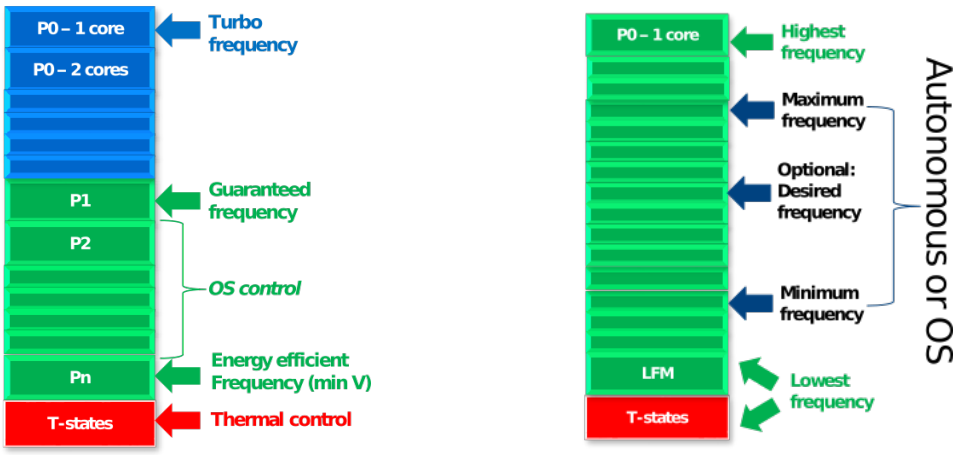
Die Skylake Mikroarchitektur wurde Ende 2015 von Intel® vorgestellt und beruht auf einem 14-nm-Verfahren. Für diese Arbeit sind besonders die Neuerungen der Power Management Funktionen von Bedeutung. So wurde die Intel® Speed-Shift-Technology eingeführt und die Möglichkeit geschaffen, für jeden CPU-Kern eine eigene Frequenz setzen zu können. Im Folgenden wird zuerst in Abschnitt 2.3.2 auf die Speed-Shift-Technology eingegangen. Anschließend wird in Abschnitt 2.3.3 genauer auf die Funktionsweise von Turbo-Boost eingegangen und die möglichen Turbo- und AVX-Frequenzen näher betrachtet.

2.3.2 Speed-Shift-Technology

Mit der Skylake Mikorarchitektur wurde die Speed-Shift-Technologie eingeführt, welche den Performancezustand des Prozessors weitestgehend selbstständig regelt [37]. Das Betriebssystem ist in der Lage, der Speed-Shift-Funktion zusätzliche Hinweise zu geben und die maximale und minimale Performance einzustellen. Vor Skylake hat das Betriebssystem selbst mithilfe eines Governors die aktuelle Performance geregelt. Dieser Governors hat dabei die aktuelle Auslastung des Systems und weitere Parameter analysiert und dementsprechend die Frequenz erhöht oder verringert.

Die Performance wird dabei als P-State verwaltet [37]. Wie in Abbildung 2.1a zu erkennen, gibt es die P-States P0 bis Pn. P1 ist dabei als die garantierte Basisfrequenz definiert. Alle States größer als P1 sind dabei jeweils Frequenzen kleiner als die Basisfrequenz. Das Gegenteil davon ist der P0-State. Bei ihm ist die Frequenz höher als die Basisfrequenz. Bei P0 wird die aktuelle von Turbo-Boost festgelegte Turbo-Frequenz verwendet.

Wie in Abbildung 2.1b zu sehen, hat sich der Umgang mit P-States geändert. Durch die Speed Shift Technologie wird der aktuell beste P-State innerhalb einer Frequenzdomäne gesetzt. Der in dieser Arbeit verwendete Prozessor hat pro CPU-Kern eine eigene Frequenzdomäne wodurch der P-State für jeden CPU-Kern einzeln bestimmt wird. Durch Messungen zeigte sich, dass auf dem eingesetzten Testsystem der Wechsel von der Basisfrequenz hin zu den Turbofrequenzen im Durchschnitt 11 Millisekunden dauert. Ohne Speed Stift Technology ist dies entsprechend von dem Performance-Governor abhängig.



(a) Alte P-State Funktionsweise

(b) Neue P-State Funktionsweise

Abbildung 2.1: Darstellung der alten und neuen Funktionsweise von P-States [37].

Mode	Basisfrequenz	Turbofrequenzen [GHz] / Aktive CPU-Kerne							
		1	2	3	4	5	6	7	8
Normal	1,8 GHz	3,0	3,0	2,7	2,7	2,1	2,1	2,1	2,1
AVX2	1,4 GHz	2,9	2,9	2,3	2,3	1,8	1,8	1,8	1,8
AVX512	900 MHz	1,8	1,8	1,5	1,5	1,2	1,2	1,2	1,2

Tabelle 2.1: Die Turbofrequenzen des Intel® Xeon® Silver 4108 Prozessors [49].

2.3.3 Intel® Turbo Boost

Die Intel® Turbo Boost Technologie (fortan Turbo-Boost genannt) erlaubt das automatische Übertakten des Prozessors unter der Bedingung, dass die Thermal Design Power (TDP) eingehalten wird. Turbo-Boost legt dabei anhand mehrerer Eigenschaften die aktuelle Turbofrequenz global für alle CPU-Kerne fest. Dabei muss ein aktiver CPU-Kern nicht die Turbofrequenz nutzen muss, sondern kann auch auf der Basisfrequenz operieren kann[1]. Die Turbofrequenz wird entsprechend verwendet, wenn der CPU-Kern im P0-State ist, welcher entsprechend von dem Betriebssystem gefordert oder von der Speed Shift Technologie ausgewählt wurde.

[1]Beim Ausführen von AVX-Operationen wird eine vorgegebene AVX CPU-Frequenz verwendet, welche von der Basis- und Turbofrequenz abweichen kann.

Abbildung 2.2: Schematische Darstellung der Turbofrequenz in Abhängigkeit der aktiven CPU-Kerne [38].

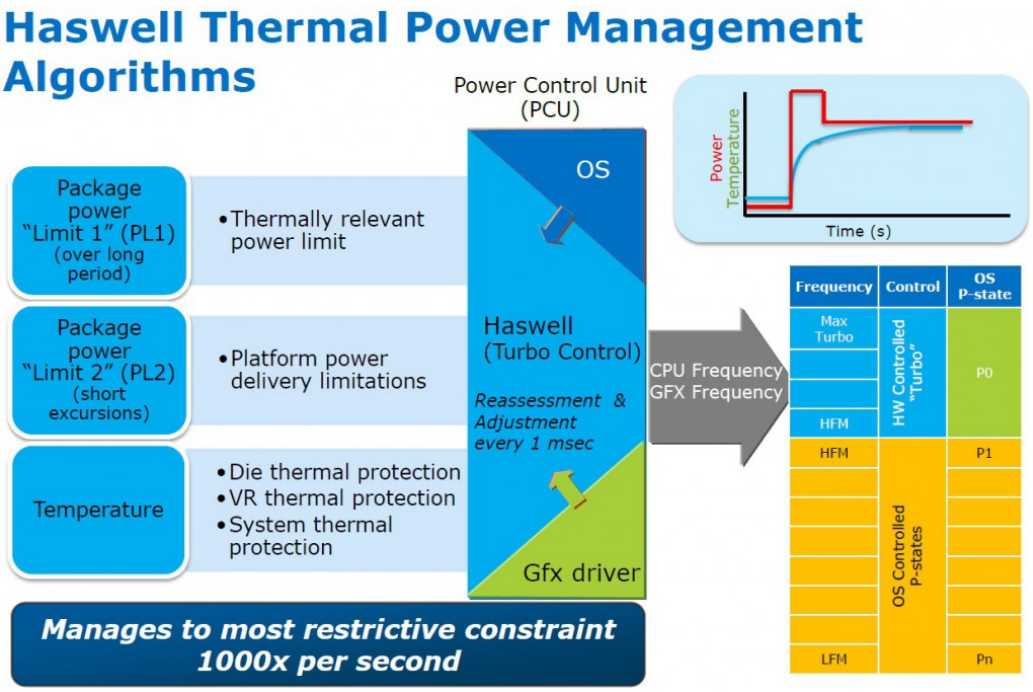


Abbildung 2.3: Darstellung der Entscheidungskriterien für Turbo Boost auf der Haswell Mikroarchitektur [40].

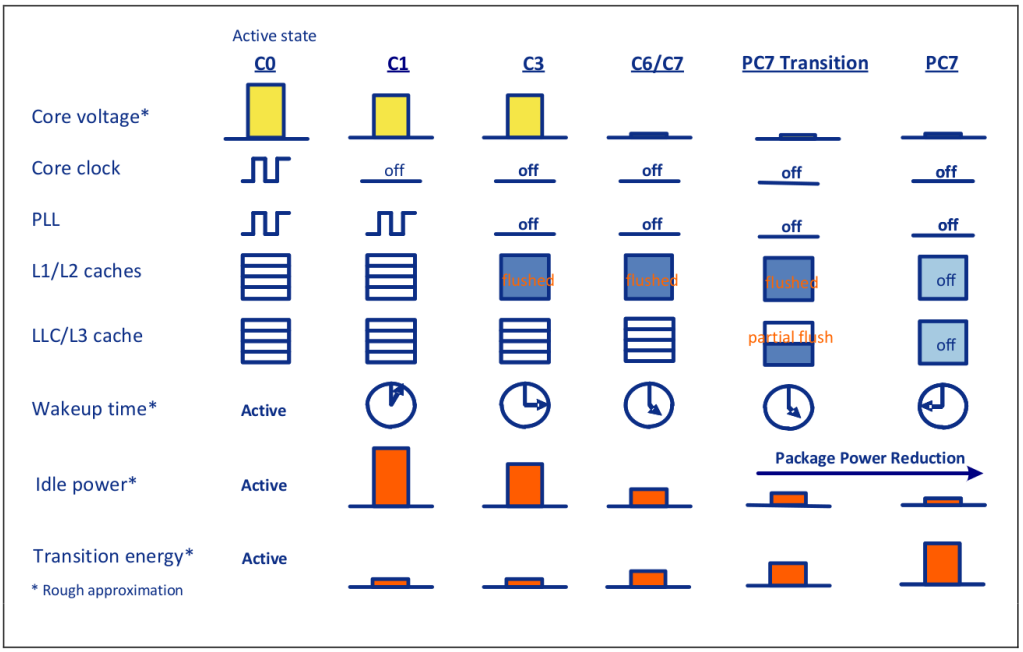


Abbildung 2.4: Eine Übersicht der unterschiedlichen C-States mit dem dazugehörigen Status des CPU-Kerns [46].

Wie in Tabelle 2.1 und Abbildung 2.2 zu erkennen gibt es unterschiedliche Turbofrequenzen, welche unter anderem abhängig von der Anzahl der aktiven CPU-Kerne sind. Daneben benutzt Turbo-Boost zusätzlich die aktuelle Temperatur und elektrischen Verbrauch, um die Turbofrequenz zu bestimmen [7]. In Abbildung 2.3 sind die Entscheidungskriterien aufgelistet, die neben den aktuell aktiven CPU-Kernen Einfluss auf die Turbofrequenz haben. In der Abbildung wird gezeigt, dass Turbo-Boost jede Millisekunde die Frequenz anpasst. Bei den gemachten Versuchen zeigte es sich, dass dies auch bei Skylake zutrifft.

Der hier beschriebene Covert Channel basiert auf der Anzahl aktiver CPU-Kerne und der daraus resultierenden Frequenzänderung. Ob ein CPU-Kern schläft oder wach ist, ist anhand von C-States festgelegt [29]. Dabei sind C-States Prozessorzustände, welche im ACPI-Standard festgelegt sind und der Energieverwaltung dienen [4]. Je höher die C-State ist, desto mehr Komponenten werden abgeschaltet, was entsprechend die Energieaufnahme verringert jedoch die Aufwachzeit verlängert. In Abbildung 2.4 ist ein Teil der C-States mit den jeweiligen abgeschalteten Komponenten dargestellt. Bei Turbo-Boost gilt ein CPU-Kern als aktiv, wenn er sich im C0 oder C1-State befindet. Alle CPU-Kerne in einem C-State höher oder gleich C3 gelten als schlafend [7].

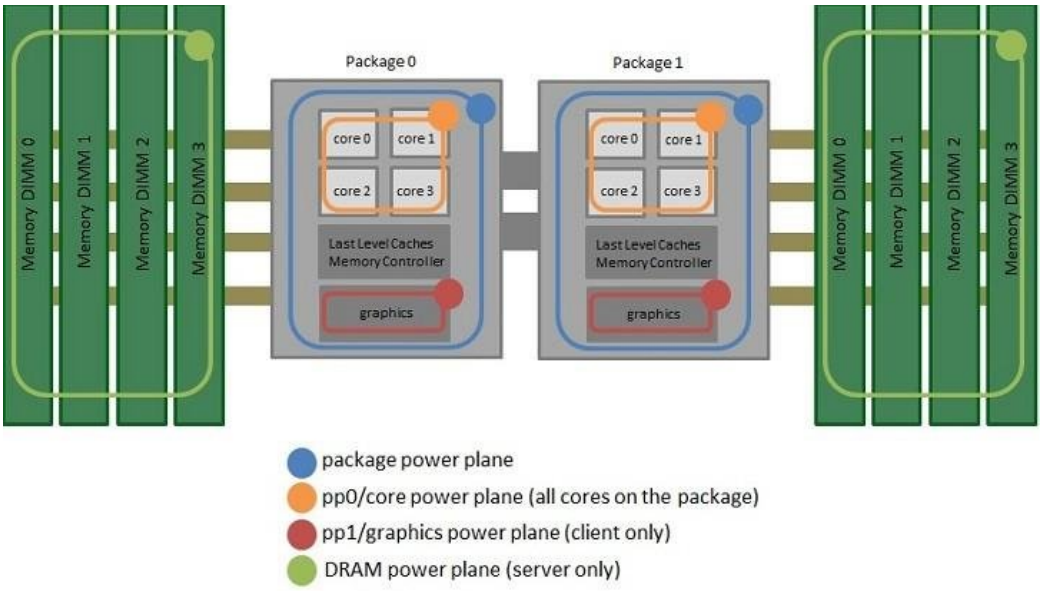


Abbildung 2.5: Darstellung der Power-Domains eines Prozessors [51].

2.3.4 Running Average Power Limit (RAPL)

Die mit der Sandy Bridge Mikroarchitektur eingeführte Running Average Power Limit (RAPL) Schnittstelle ermöglicht es, den Energieverbrauch des Prozessors zu steuern und zu überwachen [22]. Wie in Abbildung 2.5 zu sehen, ist der Prozessor in einzelne Domains unterteilt, die sich einzeln auswerten und steuern lassen. Da immer mehrere Prozessoren zu einer Domain innerhalb eines Packages zusammengefasst werden, kann der Energieverbrauch immer nur für die Domain bestimmt werden, jedoch nicht für einen einzelnen Prozessor. Für das Auslesen des Energieverbrauches wurde das von Vince Weaver entwickelte Programm raplread verwendet [43].

2.3.5 RDTSC

Die Read Time-Stamp Counter (RDTSC) Instruktion liest den Time-Stamp Counter (TSC) des aktuellen CPU-Kerns aus und schreibt ihn in die EDX- und EAX-Register. Der TSC wird mit einer konstanten Frequenz aktualisiert, welche unabhängig von der aktuellen CPU-Frequenz ist und auch die Basisfrequenz übersteigen kann [21, Vol. 2B 4-541, Vol. 3B 17-42].

Die RDTSC Instruktion selbst garantiert nicht, dass sie serialisiert oder der Prozessor keine Änderung des Ausführungsablaufes vornimmt [21]. Sollte dies benötigt werden, so muss entsprechend eine Operation zur Serialisierung im Voraus ausgeführt werden. Alternativ kann auch Read Time-Stamp Counter and Pro-

cessor ID (RDTSCP) benutzt werden, welches für eine Serialisierung vor dem Ausführen von RDTSC sorgt [16]. Da in der Evaluierung die Kernel-based Virtual Machine (KVM) als Hypervisor verwendet wird, welche im Gegensatz zu VMware, keine Unterstützung für RDTSCP bietet, wird in dieser Arbeit RDTSC verwendet [18].

Als Operation zum Serialisieren wird meist die CPUID Instruktion verwendet. Diese kann ohne spezielle Privilegien von jedem Prozess ausgeführt werden und sorgt für eine Serialisierung. Abhängig von dem Inhalt des EAX-Registers, schreibt CPUID die Prozessorinformationen und dessen Funktionsliste in die EAX-, EBX-, ECX- und EDX-Register. Zudem ist die Ausführungszeit der CPUID-Operation von dem übergebenen Parameter im EAX-Register abhängig. In Intels® White Paper, wie mithilfe der RDTSC-Instruktion Programme gebenchmarkt werden können, wird die Abhängigkeit der Laufzeit vom Übergabeparameter nicht berücksichtigt, was entsprechend zu Messfehlern führen kann [16].

2.4 Reed-Solomon-Code

Der Reed-Solomon-Code ist eine von Irving Reed und Gustave Solomon 1960 entwickelte Klasse von zyklischen Blockcodes zur Fehlerkorrektur [36]. Heutzutage findet dieser Code in kommerziellen Produkten, wie in Comapct Disks, digitalen Fernsehsignalen und verschiedenen Mobilfunkstandards Anwendung [48].

Der Reed-Solomon-Code bietet einen einfachen Weg zum Kodieren der Nachrichten. Wenn man die Nachricht $a = (a_0, a_1, ..., a_k)$ als Koeffizienten eines Polynoms $a(x) = a_0 + a_1x + a_2x^2... + a_{k-1}x^{k-1}$ einsetzt, diese an den Stützstellen $c_0, c_1, ..., c_{n-1}$ auswertet, erhält man das komplette Codewort c [25]. Dieses Codewort wird anschließend an die Nachricht angehängt und versendet. Mit diesem Code können bis zu $n/2$ Symbole korrigiert werden. Gerade für Bündelfehler ist dies ideal, da diese zu mehreren Bitfehlern innerhalb eines Symbols führen und somit nur das Symbol korrigiert werden muss und nicht jeder Bitfehler einzeln.

2.5 TCP

Das Transmission Control Protocol (TCP) ist ein Protokoll der Transportschicht, welches für einen sicheren Datenaustausch zwischen zwei Endpunkten sorgt [5]. Dabei werden Datenverluste erkannt und behoben sowie versucht eine Netzüberlastungen zu verhindern.

Um Datenverluste zu erkennen, enthält jedes Datenpaket eine Sequenznummer, welche vom Empfänger ausgelesen, die nächste zu erwartende Sequenznummer berechnet und diese anschließend im nächsten Datenpaket als Quittung

(ACK) an den Sender zurückgesendet wird. Hierdurch weiß der Sender, ob ein Datenpaket beim Empfänger angekommen ist oder es erneut gesendet werden muss. Falls ein ACK ausbleibt, sorgt ein Timeout für das erneute Senden des Datenpakets. In dieser Arbeit wird eine ähnliche Technik verwendet, um zu erkennen, ob eine Nachricht beim Empfänger erfolgreich angekommen ist.

Das TCP-Paket enthält selbst keine Angabe über die Größe der Nutzdaten. Die Datengröße des darüber liegenden Protokolls sowie der im TCP-Header stehenden Datenoffset ergibt sich die Datengröße innerhalb eines TCP-Pakets. Im Gegensatz hierzu wird in diesem Ansatz eine feste Datenlänge verwendet, um ein simpleres Übertragungsprotokoll zu erhalten.

Zur Verhinderung von Netzüberlastungen wird TCP meist mit einem Staukontrollverfahren kombiniert. Je nach Staukontrollverfahren beginnt der Sender zuerst mit einer kleinen Menge an Daten pro Datenpaket und erhöht die Anzahl der Daten pro Datenpaket bei jedem erfolgreich empfangen ACK-Nachricht [12]. Kommt es zu einem Verlust eines Datenpaketes, bleibt dessen ACK-Nachricht aus und es wird wieder von neuem begonnen, nur eine kleine Menge an Daten pro Datenpaket zu versenden. Da es bei dem hier beschriebenen Covert Channel nur einen Sender gibt, ist ein Staukontrollverfahren, wie bei TCP nicht nötig. Es könnte jedoch verwendet werden, um eine Paketgröße an das aktuelle Störungsaufkommen anzupassen.

Kapitel 3

Entwurf

Ziel dieser Arbeit ist es, einen Covert Channel zu realisieren, der mithilfe einer Manipulation der CPU-Frequenz Daten zwischen zwei CPU-Kernen überträgt, ohne dabei auf Funktionen des Betriebssystems angewiesen zu sein. Um dies umzusetzen, muss man die CPU-Frequenz beeinflussen können. Der in diesem Kapitel beschriebene Entwurf ist dabei so weit wie möglich allgemein gehalten, sodass dieser auch auf andere Arten des Covert Channels angewandt werden kann, die Turbo-Boost verwenden.

Zu Beginn werden in Kapitel 3.1 die grundlegenden Methoden beschrieben, wie die CPU-Frequenz beeinflusst werden kann und wie sich damit ein Sender und Empfänger realisieren lassen. Kapitel 3.2 behandelt mögliche Störungsquellen des Covert Channels, sowie die daraus resultierenden Fehler. Anschließend wird im folgenden Kapitel darauf eingegangen, wie der Einfluss von Störungen minimiert und somit eine Datenübertragung sichergestellt werden kann. Am Ende wird in Kapitel 3.3 abschließend der Entwurf des Kommunikationsprotokolls sowie der Aufbau einzelner Datenpakete beschrieben.

3.1 Grundlegender Entwurf

Wie in bisherigen Veröffentlichungen (Kapitel 2.1) beschrieben, kann durch Auslastung eines CPU-Kerns Einfluss auf die CPU-Frequenz anderer CPU-Kerne genommen werden. Durch die in Kapitel 2.3.1 erläuterten Neuerungen der Intel Architektur wird dies jedoch verhindert. Bereits veröffentlichte Covert Channels sind daher nicht mehr möglich oder erfordern die Kooperation des Betriebssystems.

Der in dieser Arbeit beschriebene Covert Channel setzt an dieser Stelle an. Die Intel Turbo Boost Technologie (siehe Kapitel 2.3.3) erlaubt das dynamische Übertakten der CPU-Kerne. Durch die Manipulation der Eigenschaften, die Turbo-Boost zur Festlegung der Turbofrequenz benutzt, ist der Sender in der Lage die

Turbofrequenz des Systems zu beeinflussen und somit Daten an den Empfänger zu schicken. Dies ermöglicht den Aufbau eines Covert Channels zwischen zwei CPU-Kernen und somit auch die Kommunikation zwischen zwei virtuellen Maschinen, die auf derselben Host-CPU laufen (siehe Kapitel 6.7). Eine schematische Darstellung der Übertragung ist in Abbildung 3.1 zu sehen.

In dem im Folgenden beschriebenen Szenario wird die Abhängigkeit der Turbofrequenz von der Anzahl der aktiven CPU-Kerne ausgenutzt. Die anderen Abhängigkeiten sind teils schwer beeinflussbar oder ließen nur eine niedrige Übertragungsrate zu. So benutzt Turbo-Boost beispielsweise auch die CPU-Temperatur zum Bestimmen der Turbofrequenz. Einen Temperatur-Covert Channel wurde bereits von Bartolini et al. vorgestellt [10]. Dabei zeigte sich, dass der CPU-Kühler die Benutzung des Covert Channels erschwert. Auf dem in dieser Arbeit verwendeten Testsystem konnte selbst mit allen ausgelasteten CPU-Kernen keine Drosselung der Turbofrequenz aufgrund der Temperatur festgestellt werden. Rein die Aktivität, beziehungsweise Inaktivität der CPU-Kerne hat zu einer Drosselung der Turbofrequenz geführt.

Sender

Wenn ein Sender mithilfe dieses Mechanismus ein Bit an einen Empfänger übertragen möchte, muss dieser die Anzahl der aktiven CPU-Kerne verändern. Um ein Eins-Bit zu senden, wird ein CPU-Kern für einen vorgegebenen Zeitraum aktiv gehalten und sorgt somit für eine Verringerung der Turbofrequenz. Analog dazu wird bei einem Null-Bit der CPU-Kern für den vorgegebenen Zeitraum schlafen gelegt.

Empfänger

Der Empfänger misst die aktuelle Turbofrequenz und erkennt anhand deren Variation die übertragenen Daten des Senders. Hierzu werden die Unterschiede der Turbofrequenz als Eins oder Null interpretiert. Zum Messen der aktuellen Turbofrequenz muss der Empfänger seinen CPU-Kern so weit auslasten, dass dieser die Turbofrequenzen ausnutzt, weil er sonst nur die aktuelle Basisfrequenz misst.

Asynchrone Datenübertragung

Damit der Empfänger erkennt, wann der Sender Daten schickt, wird in teils ein zusätzlicher Takt verwendet, welcher vorgibt, wann ein Bit versendet wird und wann nicht. Dieser Takt kann, wie Beispielsweise bei SPI [19], über ein zusätzliches Signal neben dem Datensignal übertragen werden oder aus dem Datensignal zurückgewonnen werden. Da ein zusätzliches Taktsignal innerhalb dieses Covert

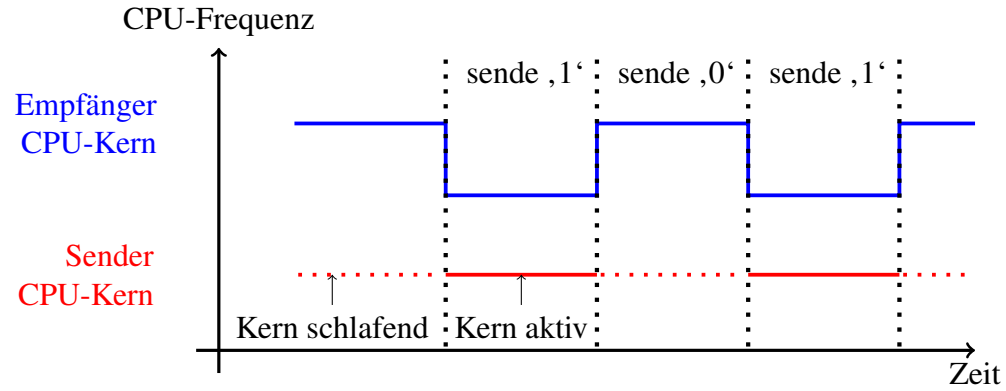


Abbildung 3.1: Schematische Darstellung einer Übertragung. Um eine Eins zu senden, wird der Sender-Kern für einen vorgegebenen Zeitraum aktiv gehalten. Die sinkende Turbofrequenz wird vom Empfänger gemessen und entsprechend interpretiert.

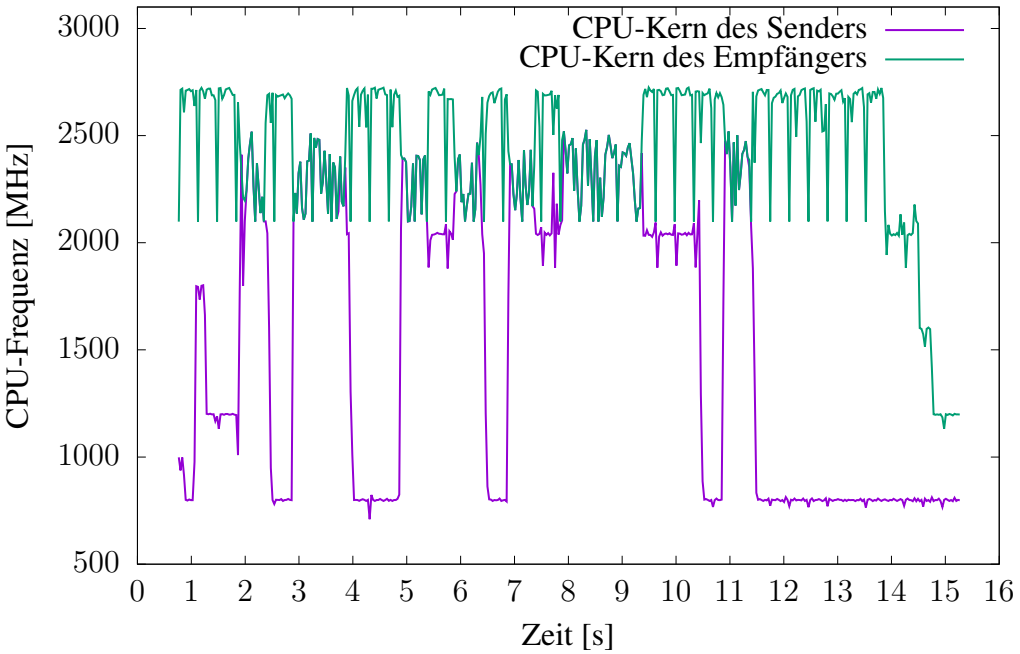


Abbildung 3.2: Proof of Konzept des gezeigten Covert Channels. Sobald die Sender-CPU Berechnungen anstellt, sinkt die Frequenz der Empfänger-CPU auf die nächstniedrigere Turbofrequenz.

Channels nicht möglich ist und auch die Möglichkeit der Taktrückgewinnung, beispielsweise mit dem Manchester-Code [44], eine Halbierung der Übertragungsrate zur Folge hat, wird hier eine asynchrone Datenübertragung verwendet. Dabei wird anhand eines Anfangssignals der Beginn einer Datenübertragung erkannt und mit der im Voraus festgelegten Sendezeit pro Bit eingelesen.

3.2 Störungen

Wie bei den meisten Kommunikationskanälen treten auch hier Störungen auf, die die Übertragung beeinflussen können. Im anschließenden Kapitel 3.2.1 wird auf die möglichen Störungen und daraus resultierenden Übertragungsfehler eingegangen und im darauf folgenden Kapitel 3.2.2 werden die getroffenen Gegenmaßnahmen erläutert, die den Einfluss dieser Störungen minimieren und Übertragungsfehler verhindern.

3.2.1 Arten von Übertragungsfehlern

Aufgrund der Komplexität des Covert Channels gibt es diverse Störungsquellen. Auf der Hardwareebene können Interrupts zu Unterbrechungen der Sende- und Empfangsprozesse führen und somit Fehler im Übertragungsvorgang erzeugen. Durch Interrupts werden oft weitere Aktivitäten des Betriebssystems ausgelöst, welche in eigenen Kernel-Threads ausgeführt werden. Durch diese zusätzliche Arbeit kann es passieren, dass weitere CPU-Kerne aktiv werden und somit die Turbofrequenz beeinflussen. Selbst kurzzeitig aktive CPU-Kerne sorgen für eine Beeinflussung der Turbofrequenz für mindestens eine Millisekunde, da Turbo-Boost die aktuelle Turbofrequenz mit 1000 Herz aktualisiert.

Neben den Hintergrundaktivitäten des Betriebssystems verursachen auch weitere laufende Programme Störungen. Der Grund hierfür ist der Scheduler des Betriebssystems. Er weißt Prozessen einen CPU-Kern zu, was entsprechend Einfluss auf die Turbofrequenz haben kann. Zudem kann der Scheduler den Sende- und Empfangsprozess unterbrechen oder auf einen anderen CPU-Kern verschieben, was zu Übertragungsfehlern führen kann. Dabei können diese und die oben genannten Störungsquellen zu folgenden Übertragungsfehlern führen:

Einzelbitfehler

Bei einem Einzelbitfehler ist nur ein einzelnes Bit falsch, unabhängig von anderen auftretenden Fehlern. So ein Fehler kann durch falsche Annahmen über die maximale Turbofrequenz oder durch das kurzzeitige Aktivieren weiterer CPU-Kerne passieren. In Grafik 3.3b ist dargestellt, wie ein solcher Fehler aussehen kann.

Der Fehler eine Eins anstelle einer gesendeten Null zu erkennen, tritt am häufigsten auf. Wird beim Übertragen einer Null ein zusätzlicher CPU-Kern aktiv, so sinkt die Turbofrequenz und es wird anstelle einer Eins eine Null erkannt. Der Grund hierfür ist, dass der Empfänger zu Beginn die maximale Turbofrequenz ermittelt und sie als ein erkanntes Null-Bit festlegt. Sinkt die Turbofrequenz entsprechend, so wird dies als ein gesendetes Eins-Bit interpretiert. Ist ein weiterer CPU-Kern während des Sendevorgangs einer Eins aktiv, so kommt es nicht zu einem Fehler, weil alle Frequenzen kleiner als die am Anfang festgelegte maximale Turbofrequenz als ein übertragenes Eins-Bit erkannt werden.

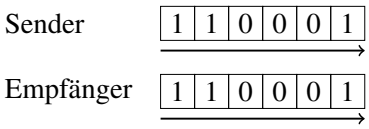
Der umgekehrte Fall, dass eine Eins gesendet und eine Null erkannt wird, tritt auf, wenn zu Beginn nicht die höchste Turbofrequenz als Null definiert wurde. Möglich ist dies, wenn ein Prozess S durchgehend einen CPU-Kern aktiv hält und somit der Empfänger beim Start eine falsche höchste Turbofrequenz ermittelt. Pausiert sich Prozess S während eines Übertragungsvorganges, so kann dies dazu führen, dass sich der aktive CPU-Kern schlafen legt und somit die Turbofrequenz steigt. Dies hat zur Folge, dass eine übertragene Eins als Null erkannt wird. Eine übertragene Null wird auch weiterhin als eine Null erkannt.

Bündelfehler

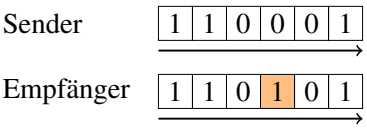
Wie in Grafik 3.3c zu erkennen, werden bei einem Bündelfehler mehrere hintereinander liegende Bits falsch erkannt. Sie entstehen auf dem gleichem Weg wie Einzelbitfehler, durch zusätzlich aktive CPU-Kerne oder eine zu Beginn falsch erkannte höchste Turbofrequenz.

Synchronisationsfehler

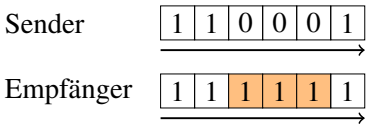
Durch die Wahl einer asynchronen Datenübertragung kann es zu Synchronisationsfehlern kommen. Beispielsweise kann der Scheduler kurzzeitig den Sender- oder Empfängerprozess unterbrechen, wodurch wie in Grafik 3.3d zu erkennen Synchronisationsfehler auftreten können. Durch das nicht vorhandenen Synchronisationssignal, kann die Information verloren gehen, wann ein Bit gesendet wird. Zu Beginn einer Übertragung wird über eine Startsequenz und die festgelegten Sendedauer eines Bits die Synchronisation hergestellt. Kommt es während einer Übertragung zu einem Synchronisationsfehler, so sind die übertragenen Daten meist nicht mehr rekonstruierbar.



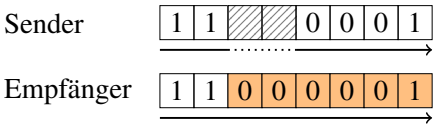
(a) Eine Übertragung ohne Fehler. Der Sender sendet ‚110001' und der Empfänger empfängt ‚110001'.



(b) Ein Einzelbitfehler. Aufgrund von Störungen erkennt der Empfänger das Bit falsch.



(c) Bei einem Bündelfehler flippen mehrere Bits direkt hintereinander.



(d) Synchronisationsfehler durch z.B. den Scheduler, der den Sender kurzzeitig unterbricht.

Abbildung 3.3: Darstellung der typischen Fehler, die in diesem Covert Channel auftreten können.

3.2.2 Übertragungsfehler minimieren

Die einfachste Lösung, um Fehler zu beheben, die aufgrund von Störungen aufgetreten sind, ist das erneute Übertragen der Daten. Dies benötigt jedoch viel Bandbreite und sollte daher nur erfolgen, falls alle benutzten Methoden zur Verminderung des Einflusses von Übertragungsfehlern nicht erfolgreich waren. Im Folgenden werden die Methoden beschrieben, welche in dieser Arbeit eingesetzt wurden, um Fehler durch Störungen zu minimieren. Zur Minimierung der Einflüsse von Einzelbitfehler wird Codespreizung eingesetzt. Ein Error-Correction-Code behandelt mögliche Bündelfehler und bei irreparablen Fehlern müssen die Daten erneut gesendet werden.

Verringerung der Bandbreite

Wie in Tabelle 3.4 zu sehen kann es während einer Übertragung zu vielen Störungen kommen. Vor allem, wenn wie hier im Beispiel zwei virtuelle Maschinen laufen, und das Betriebssystem die Arbeit der CPU-Kerne festlegen darf. Wenige Störungen treten auf, wenn das Betriebssystem keine Arbeit auf die CPU-Kerne verteilen darf und auch sonst kein Programm läuft. Hierbei ist eine Bandbreite von 1000 bit/s möglich, welche der Aktualisierungsfrequenz von Turbo-Boost entspricht.

Störungen	
Länge	Anzahl
1 ms	109
2 ms	5
3 ms	2
4 ms	1
6 ms	1

(a) System ohne Last

Störungen	
Länge	Anzahl
1 ms	4
2 ms	1

(b) Isolierter Kern

Störungen	
Länge	Anzahl
1 ms	145
2 ms	8
3 ms	5
4 ms	1
5 ms	1
7 ms	1

(c) Zwei gleichzeitig laufende virtuelle Maschinen

Abbildung 3.4: Anzahl der Fehler die durch Absenkung der Turbofrequenz, im Zeitraum von einer Sekunde auftreten. Gemessen auf dem in Kapitel 6.2 beschriebenen System.

Sobald Störungen hinzukommen, muss die Bandbreite gesenkt werden, um eine fehlerfreie Übertragung zu ermöglichen. Bei einer zu hohen Bandbreite können aufgrund von Übertragungsfehlern die übertragenen Daten nicht mehr vollständig rekonstruiert werden. Durch eine Verringerung der Bandbreite liegt das Signal für ein Bit entsprechend länger Zeit an. Ein Bit wird damit über mehrere Millisekunden gesendet. Einzelbitfehler sorgen hierbei für einen Fehler von einer Millisekunde innerhalb der Übertragungszeit eines Bits. Da ein Bit über mehrere Millisekunden übertragen wird, kann aus den restlichen richtig übertragenen Bits das originale Signal rekonstruiert werden. Die optimale Bandbreite wird in Kapitel 6.6 ermittelt.

Ein Ansatz, bei dem die Bandbreite nicht deutlich verringert werden muss, jedoch trotzdem Störungen und Einzelbitfehler korrigiert werden können, wäre die Verwendung von Codespreizung. Hierbei kann ein Bit durch vier Bit repräsentiert werden, welche jeweils entsprechend einer Metrik verschoben werden. Aus ‚abc' kann hierbei ‚abcabcabcabc' werden. Die Idee hinter Codespreizung ist, dass kurze Störungen nur ein Teilbit beschädigt, welches anschließend rekonstruiert werden kann. Die Verschiebung ermöglicht es zudem, dass auch kleine Bündelfehler zu keinem Fehler führt. Bei einer längeren Störung werden mehrere Teilbits von unterschiedlichen Bits gestört. Mit den restlichen fehlerfreien Teilbits kann anschließend das Signal rekonstruiert werden. Ein ähnliches Verfahren wird bei der Compact-Disc in Verbindung mit dem Reed-Solomon-Code eingesetzt, um Fehler aufgrund von Kratzern zu minimieren [48]. Bei Versuchen hat sich gezeigt, dass Codespreizung auch die gewünschte Fehlerkorrekturwirkung erzielt. Das Ziel einer hohe Bandbreite bei geringer Fehlerrate wurde jedoch nicht erreicht. Es hat

sich gezeigt, dass mit einer Bandbreite von 500 bit/s und dem oben beschriebenen Verfahren eine höhere Fehlerrate erreicht wird, als mit der reinen Absenkung der Bandbreite auf 125 bit/s. Zusätzlich hat sich gezeigt, dass die Störungen so schlimm waren, dass der Error-Correction-Code die fehlerhaften Daten als fehlerfrei erkannt hat. Die fehlerhafte Erkennung von Übertragungsfehlern ließe sich mit einem anderen Error-Correction-Code beheben, jedoch behebt diese nicht die große Anzahl an Übertragungsfehlern. Um eine Übertragung mit wenigen Fehlern zu erreichen, muss die Bandbreite bei Codespreizung gleich sein wie bei den Versuchen ohne Codespreizung, wobei Codespreizung bei gleicher Bandbreite eine vier mal geringere Übertragungsrate besitzt. In diesem Covert Channel wird daher kein Codespreizung eingesetzt.

Error-Correction-Code

Codespreizung korrigiert Einzelbitfehler und bei einer entsprechend großen Spreizung auch kurze Bündelfehler. Längere Bündelfehler können mit Codespreizung nicht korrigiert werden. Um diese Fehler zu erkennen und falls möglich zu beheben, kommen Error-Correction-Codes zum Einsatz. Diese Codes fügen den Daten zusätzliche Bits hinzu, welche es ermöglichen, eine bestimmte Anzahl an Bit-Fehlern zu erkennen und eine geringere Anzahl an Bit-Fehlern zu korrigieren.

In dem, in Kapitel 3.3 vorgestellten, Protokoll, welches in dieser Arbeit verwendet wird, wird ein Reed-Solomon-Code (Kapitel 2.4) eingesetzt. Die eingesetzte Variante des Reed-Solomon-Code ist dabei in der Lage, byteweise Fehler zu erkennen und zu korrigieren. Es können dabei halb so viele Bytes repariert werden wie Paritätsbytes existieren.

Retransmission

Falls die zuvor genannten Verfahren einen Übertragungsfehler nicht beheben konnte, so müssen die Daten neu gesendet werden. Um den Sender mitzuteilen, welche Daten irreparabel beschädigt sind und erneut gesendet werden müssen, wird zusätzlich ein Kommunikationskanal vom Empfänger zum Sender benötigt. Ohne einen solchen Rückkanal müsste der Sender davon ausgehen, dass jede Übertragung erfolgreich ist und mögliche Übertragungsfehler behoben werden konnten. Alternativ könnte er auch alle Daten so lange senden, bis die Wahrscheinlichkeit sehr hoch ist, dass die Übertragung erfolgreich war.

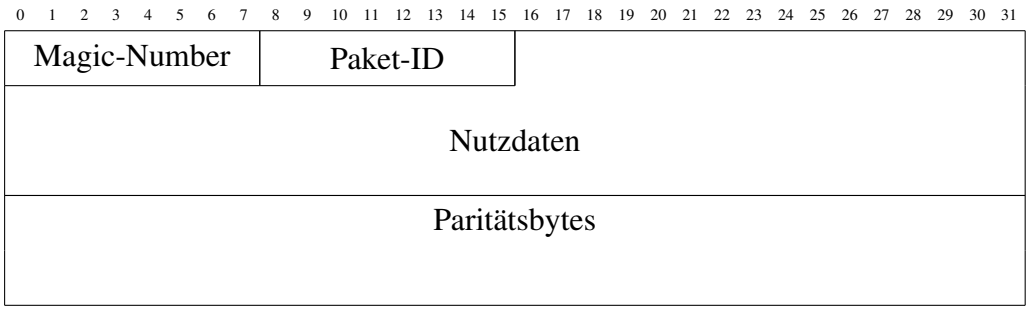


Abbildung 3.5: Die Abbildung zeigt den Aufbau eines Datenpakets. Die Magic-Number und Paket-ID sind dabei jeweils ein Byte groß. Die Nutzdaten und Paritätsbytes haben ein Vielfaches eines Bytes als Größe.

3.3 Konkreter Entwurf

In den Kapiteln 3.1 und 3.2 wurden das grundlegende Design des Covert Channels erläutert und auf mögliche Fehlerquellen sowie auf die Möglichkeit, diese zu verhindern, eingegangen. Basierend auf diesen Erkenntnissen wurde das im Folgenden beschriebene Kommunikationsprotokoll designt. Dabei wurde darauf geachtet, dass Protokoll entsprechend einfach zu halten und dennoch eine hohe Datenrate zu erreichen.

Um das Ziel einer hohen Datenrate zu erreichen, wurde besonders versucht die Anzahl der erneuten Übertragungen zu minimieren. Da die Wahrscheinlichkeit, dass ein Fehler die übertragenen Daten irreparabel beschädigt, mit der Größe der am Stück übertragenen Datenmenge steigt, wurde entschlossen größere Datenmengen in einzelne Datenpakete aufzuteilen. So wird versucht die Anzahl der Daten, die aufgrund von Fehlern erneut gesendet werden müssen, zu minimieren und somit auch weniger Zeit für die Übertragung der Daten zu brauchen. Im Folgenden wird zuerst der Aufbau eines Datenpaketes beschrieben und anschließend darauf eingegangen, wie das Übertragungsprotokoll aufgebaut ist.

3.3.1 Aufbau eines Datenpaketes

In Abbildung 3.5 ist der Aufbau eines Datenpakets dargestellt. Ein Datenpaket setzt sich aus einer Magic-Number, Paket-ID, den Nutzdaten und Paritätsbytes zusammen. Dieses Datenpaket ließe sich auch als Paket für den Rückkanal verwenden. In dieser Arbeit jedoch ist nicht vorgesehen, dass der Empfänger mehr Daten an den Sender schickt als die Bestätigung über das erfolgreiche Empfangen einer Nachricht. Daher erhält der Rückkanal ein eigenes Paketformat, das ACK-Paket. Dies enthält nur die Paket-ID des zuletzt erfolgreich empfangenen Datenpaketes. Durch Ersetzen des ACK-Pakets durch ein Datenpaket lässt sich

dieser Covert Channel in einen vollständigen, bidirektionalen Kommunikationskanal erweitern.

Im Vergleich zu Protokollen, wie Beispielsweise TCP (siehe Kapitel 2.5), hat hier jedes Datenpaket eine feste Größe. Dies erleichtert die Auswertung eines Datenpaketes.

Magic-Number

Jedes Datenpaket beginnt mit einer Magic-Number. Diese ist ein Byte groß und hat den Zweck der Erkennung des Beginns eines Datenpaketes sowie zur Synchronisierung von Bitanfängen. Die Magic-Number muss dabei so gestaltet sein, dass sie normalerweise nicht durch Rauschen erzeugt werden kann. Dadurch fallen `0000 0000` oder `1111 1111` als Magic-Number weg, da diese Folgen sehr häufig auftreten. Als Magic-Number wurde `1100 1010` (`0xCA`) gewählt. Diese Zahl zeichnet sich dadurch aus, dass sie viele Wechsel zwischen Eins und Null hat und auch zwei Folgen von Eins und Null. In Kombination mit Codespreizung ergibt sich so eine Folge, bei der es sehr unwahrscheinlich ist, dass ein oder mehrere Prozesse dieses Aktivitätsmuster zufällig erzeugen.

Paket-ID

Der Zweck der Paket-ID ist es, sicherzustellen, dass mehrfach gesendete Pakete erkannt werden, und um dem Sender zurückzumelden, ob das Datenpaket vollständig angekommen ist. Dazu erhalten die Datenpakete eine aufsteigende Paket-ID. Um die maximale Anzahl an Datenpaketen nicht durch die Größe der Paket-ID zu limitieren, wird die Paket-ID beim Überlaufen zurückgesetzt.

Die Paket-ID fungiert als zusätzliche Sicherheitsinstanz um fehlerhafte Pakete zu erkennen. Dabei überprüft der Empfänger, ob ein angekommenes Paket die erwartete Paket-ID besitzt. Bei zu vielen Übertragungsfehlern kann es passieren, dass der Error-Correction-Code die Fehler nicht erkennt und die Nachricht als fehlerfrei markiert. Dann kann die Paket-ID benutzt werden, um die Wahrscheinlichkeit zu senken, dass ein als fehlerfrei erkanntes Datenpaket Fehler enthält. Wenn die Paket-ID nicht der erwarteten entspricht, liegt augenscheinlich ein Fehler vor und das Datenpaket kann als irreparabel beschädigt angesehen werden.

Die Paket-ID ist, genau wie die Magic-Number, ein Byte groß. Die Paket-ID würde auch mit weniger Bits auskommen. Die Größe von einem Byte ergibt sich jedoch daher, dass versucht wurde, eine byteweise Paketausrichtung zu erhalten. Diese Ausrichtung erleichtert das spätere Auswerten der empfangen Datenpakete und ermöglicht es des Weiteren, zu einem späteren Zeitpunkt weitere Kontrollbits einzufügen. Dabei kann die Größe der Paket-ID verkleinert werden, um somit Platz für weitere Kontrollbits zu schaffen.

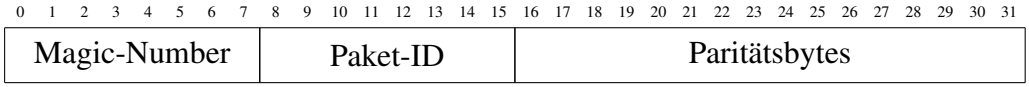


Abbildung 3.6: Das ACK-Paket sendet der Empfänger an den Sender, um das letzte erfolgreich empfangene Datenpaket zu bestätigen. Dabei ist das ACK-Paket ein normales Datenpaket ohne Daten.

Nutzdaten

Das Nutzdatenfeld enthält die Daten, die vom Sender zum Empfänger übertragen werden sollen. Die Größe des Datenfeldes ist dabei fest vorgegeben. In Kapitel 6.6 wird analysiert, welche Größe sich hierfür am besten eignet. Sollten weniger Daten übertragen werden, als das Datenfeld groß ist, so muss der restliche Platz mit Null aufgefüllt werden. Dies ermöglicht ein einfacheres Protokoll, was jedoch für den Empfänger bedeutet, dass dieser wissen muss, wie viele Daten er empfangen soll.

Paritätsbytes

Am Ende eines Datenpaketes befinden sich die Paritätsbytes. Wie in Kapitel 3.2.2 bereits erwähnt kommt hier ein Reed-Solomon-Code (Kapitel 2.4) zum Einsatz. Dabei wird der Error-Correction-Code nicht nur auf die Nutzdaten angewandt, sondern auch auf die Paket-ID. Dies stellt sicher, dass alle wichtigen Daten ohne Fehler übertragen wurden.

ACK-Paket

Die Funktion des ACK-Pakets (Aufbau siehe Abbildung 3.6) ist an das ACK-Flag von TCP (siehe Kapitel 2.5) angelehnt. Nach Erkennen eines Datenpaketes sendet es der Empfänger an den Sender, um den Eingang eines Datenpaketes zu bestätigen. Die Paket-ID ist dabei die Paket-ID des zuletzt korrekt empfangenen Datenpaketes. Auf diesem Weg kann der Empfänger dem Sender mitteilen, dass ein Fehler beim Übertragen aufgetreten ist und das Datenpaket erneut gesendet werden muss.

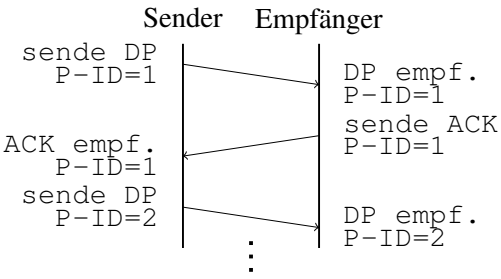
3.3.2 Sendeablauf

Auf der Sicherungsschicht wurde ein einfaches Kommunikationsprotokoll gewählt. In Abbildung 3.7 ist der Sendeablauf, sowie die möglichen Fehlerfälle schematisch dargestellt.

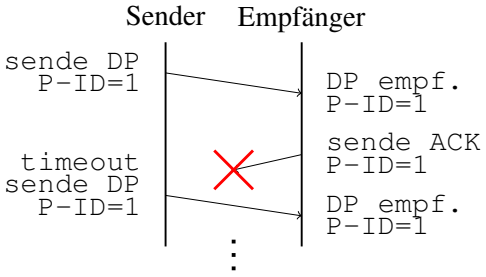
Bei diesem Kommunikationsprotokoll sendet der Empfänger nach jedem erhaltenen Datenpaket ein ACK-Paket an den Empfänger. Das ACK-Paket enthält

die Paket-ID des zuletzt erfolgreich übertragenen Datenpakets. Sollte das zuletzt erhaltene Datenpaket irreparabel beschädigt sein, so wird die Paket-ID des zuletzt korrekt übertragenen Datenpaketes verwendet. Sollte das ACK-Paket irreparabel beschädigt beim Sender ankommen, so wird dieser annehmen, dass die letzte Übertragung nicht erfolgreich war und das letzte Datenpaket von neuem senden.

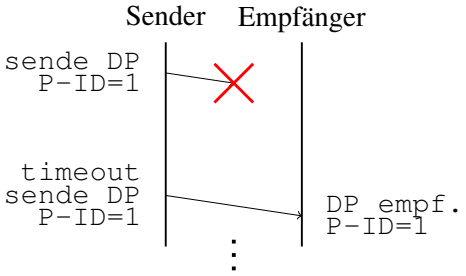
Der Empfänger reagiert nur auf ein erkanntes Datenpaket. Wenn dieser kein Datenpaket erkennt, wird er auch kein ACK-Paket senden. Dem gegenüber steht der Sender, welcher direkt nach dem Senden eines Datenpaketes auf dem Übertragungskanal nach einem ACK-Paket lauscht. Trifft dies nicht innerhalb eines vorgegebenen Zeitraumes ein, wird ein Timeout ausgelöst, welcher das zuletzt gesendete Datenpaket von neuem sendet. Auch bei einem irreparabel beschädigten empfangenen ACK-Paket wird der Sender das zuletzt gesendete Datenpaket von neuem senden. Als Timeout wird die doppelte Übertragungszeit eines Datenpaketes verwendet. Dies hat den Grund, dass es sein kann, dass der Empfänger erst zum Ende einer Übertragung eine Magic-Number erkennt und damit annimmt, dass ab hier das Datenpaket beginnt. Hat der Empfänger den Anfang eines Datenpaketes erkannt, so wird er die vorgegebene Länge des Datenpaketes einlesen und anschließend das ACK-Paket senden. Jegliche erneuten Sendevorgänge des Senders bleiben dabei unentdeckt. Damit geht entsprechend auch der nächste Sendevorgang verloren. Um dies zu umgehen wird die doppelte Übertragungszeit eines Datenpaketes als Timeout verwendet. Das Timeout ließe sich auch wie bei TCP (siehe Kapitel 2.5) dynamisch zur Laufzeit an entsprechende Begebenheiten anpassen, hierauf wird jedoch nicht weiter eingegangen.



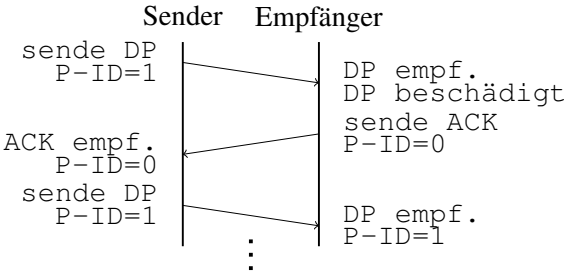
(a) Datenübertragung ohne Fehler



(b) ACK-Paket nicht erkannt oder beschädigt. Sender sendet Datenpaket (DP) nach Timeout erneut.



(c) Datenpaket (DP) nicht erkannt oder beschädigt. Sender sendet Datenpaket nach Timeout erneut.



(d) Das Datenpaket (DP) wurde irreparable beschädigt. Empfänger sendet P-ID des zuletzt erfolgreich emfangenen Datenpaket.

Abbildung 3.7: Schematische Darstellung der Verhalten von Sender und Empfänger während einer Datenübertragung.

Kapitel 4

Implementierung

In diesem Kapitel wird die genaue Implementierung des zuvor in Kapitel 3 beschriebenen Entwurfes für einen frequenzbasierten Covert Channel beschrieben. Wie schon im Entwurfskapitel lässt sich auch die Implementierung in Sender und Empfänger trennen. Um das festgelegte Kommunikationsprotokoll umzusetzen, werden Sender und Empfänger beide abwechselnd senden und empfangen. Daher wurde bei der Implementierung darauf geachtet, dass die gleichen Funktionen von Sender und Empfänger gleichermaßen verwendet werden können und somit keine extra Implementierung benötigt wird. Zudem wurde bei der Implementierung darauf geachtet, dass ein Großteil der Funktionen modular aufgebaut ist und sich somit die Auswertefunktion einfach austauschen lässt. Diese Modularität wurde gewählt, um einen einfachen Weg zu haben Teile des Programms auszutauschen und so die Funktionalität zu erweitern. Gerade beim Ausprobieren unterschiedlicher Kodierungsverfahren oder Signalanalyseansätzen hat sich dies als sehr nützlich erwiesen.

Zu Beginn wird in Abschnitt 4.1 mit der Beschreibung der Implementierung des Senders begonnen, gefolgt von der Beschreibung der Implementierung des Empfängers in Abschnitt 4.2. Aufgrund der Anforderung des Kommunikationsprotokolls, dass eine ACK-Nachricht mit Information zu dem zuletzt erfolgreich empfangen Datenpaket zurückgeschickt werden muss, ist es nötig, dass das Signal zur Laufzeit ausgewertet wird. Daher wird abschließend in Abschnitt 4.3 auf die Implementierung der Signalauswertung zur Laufzeit eingegangen.

4.1 Implementierung des Senders

Die Implementiernug des Senders basiert auf dem in Kapitel 3.1 beschriebenen Entwurf. Um eine Eins zu senden, muss dafür innerhalb eines vorgegebenen Zeitraumes ein CPU-Kern aktiv gehalten werden und analog dazu zum Senden einer

Null der CPU-Kern schlafen gelegt werden. Im Anschluss werden die einzelnen Stufen beschrieben, die die zu sendenden Daten während eines Sendevorgangs durchlaufen.

Datenpakete vorbereiten

Als Erstes werden die Datenpakete für das spätere Senden vorbereitet, so dass sie nicht mehr während des Sendevorgangs erzeugt werden müssen, da beispielsweise durch Festplattenzugriffe unnötige Störungen entstehen könnten, die versucht werden zu vermeiden. Hierzu wird werden direkt beim Einlesen der zu sendenden Daten die entsprechenden Datenpakete mit dazugehörigen Packet-ID und Paritätsbytes erzeugt. Der Sender muss später nur das zu sendende Paket auswählen und es kann direkt gesendet werden. Für das ACK-Paket wird diese Methode der Vorarbeit nicht geleistet, da hier keine Daten eingelesen werden müssen und auch der Reed-Solomon-Code nur auf eine kleine Datenmenge angewandt werden muss. Hier wird zur Laufzeit die entsprechende Antwort generiert.

Konvertierung ins Binärformat

Zum einfachen Erhalten der zu sendenden Bits aus den Daten wird zu Beginn ein zweidimensionales Array berechnet mit allen 8-Bit-Zahlen, ihrer jeweiligen Repräsentation im Dualsystem. Auf diesem Weg kann beispielsweise mit `array[5][6]` das sechste Bit der Zahl `0x5` ausgegeben werden, ohne jedes mal Bitoperation vornehmen zu müssen.

Senden eines Null-Bit

Den CPU-Kern selbst schlafen legen kann ein User-Mode-Programm nicht. Ein CPU-Kern schlafen legen beziehungsweise in einen C3 oder höheren State versetzen kann nur das Betriebssystem. Daher muss der Sender hier darauf hoffen, dass er nicht läuft, auch kein weiteres Programm auf dem CPU-Kern läuft und das Betriebssystem den CPU-Kern entsprechend schlafen legt. Daher ist dieser Angriff auch nur bei einer geringen CPU-Auslastung möglich. Ein Programm lässt sich über die C-Library Funktion `usleep` schlafen legen [6].

Bei den ersten Sendeversuchen hat sich gezeigt, dass das Senden einer Null dazu führt, dass die Sendedauer eine Millisekunde kürzer ist als gewollt und das vorhergehende Eins-Bit entsprechend eine Millisekunde länger gesendet wird. Dies liegt daran, dass beim Senden eines Null-Bits zuerst die Anzahl der zu senden Bits bestimmt wird und anschließend `usleep` aufgerufen wird. Bis der gewollte C-State-Wechsel eintritt, vergeht zudem Zeit. Da Turbo-Boost immer die Turbofrequenz für eine Millisekunde bestimmt, reicht die kurze Zeitspanne aus, dass

```
1 start_time = clock_gettime();
2 delta = 0;
3 while(delta < sendedauer_pro_bit) {
4    sqrt(rand());
5    delta = clock_gettime() - start_time;
6 }
```

Abbildung 4.1: Pseudocode der Implementierung des Senders zum Senden eines Eins-Bits.

weiterhin die niedrigere Turbofrequenz bestehen bleibt. Um dies zu verhindern wird jeder Sendevorgang eines Eins-Bits etwas verkürzt und dementsprechend der Sendevorgang eines Null-Bits verlängert, um so ein deutliches Signal zu erhalten.

Senden eines Eins-Bit

Ein Eins-Bit lässt sich leichter senden als ein Null-Bit. Dazu reicht es aus den CPU-Kern aktiv zu halten. Am einfachsten geht dies, in dem man ihm eine Aufgabe gibt, wie zum Beispiel eine Schleife auf einen bestimmten Wert zählen lassen. Aufgrund von unterschiedlichen Frequenzen hat die Instruktion für eine Addition eine variable Laufzeit und es kann somit nicht bis zu einem festen Wert gezählt werden. Eine bessere Lösung ist das Benutzen eines Zeitmessers, welcher wie in Abbildung 4.1 entsprechend überprüft, dass genügend Zeit für das Senden eines Eins-Bits vergangen ist.

4.2 Implementierung des Empfängers

Im Vergleich zum Sender ist der Empfänger aufwendiger zu implementieren. So muss hier zuerst die aktuelle Frequenz bestimmt werden und anschließend werden daraus die Signale rekonstruiert.

Frequenzerkennung

Die Frequenz des aktuellen CPU-Kerns lässt sich auf mehreren Wegen bestimmen. Zum Beispiel kann die von Betriebssystem bereitgestellte Sysfs-Schnittstelle verwendet werden. Diese hat den Nachteil, dass das Betriebssystem in der Lage ist, diese Informationen zu manipulieren oder ganz vorzuenthalten, und die Schnittstelle wie von Alagappan et al. festgestellt, eine geringe Aktualisierungsfrequenz aufweist [8]. Um die CPU-Frequenz mit einer höheren Frequenz auslesen zu können, kann auf das Model-Specific-Register (MSR) zurückgegriffen wer-

den. Dies erlaubt das Auslesen der aktuellen CPU-Frequenz für alle CPU-Kerne mit einer deutlichen höheren Frequenz. Auf das MSR kann jedoch nur im Kernel-Mode zugegriffen werden, womit diese Abfragen vom Betriebssystem abhängig sind und somit für einen Covert Channel, welcher unabhängig vom Betriebssystem sein soll, nicht infrage kommen.

Um die CPU-Frequenz unabhängig vom Betriebssystem bestimmen zu können, wird die Anzahl an Durchläufen einer Schleife innerhalb eines Zeitraums gemessen. Anhand der Anzahl der Schleifendurchläufe lässt sich die aktuelle CPU-Frequenz erkennen. Dies hat den Nachteil, dass nur die CPU-Frequenz des CPU-Kerns bestimmt werden kann, auf dem der Empfänger ausgeführt werden kann, was für diesen Covert Channel jedoch kein Problem ist, da die Turbofrequenz auf allen CPU-Kernen gleich ist. Der größte Vorteil dieser Methode ist es, dass es schwer ist, auf diesem Weg das Auslesen der CPU-Frequenz zu verhindern. Zudem ermöglicht es diese Methode, die CPU-Frequenz mit einer hohen Auflösung zu bestimmen.

Für diese Methode der Frequenzbestimmung ist eine Zeitmessung nötig. Dies kann Beispielsweise durch den Aufruf der Bibliotheksfunktion `clock_gettime` erfolgen. Der Rückgabewert dieser Funktion ist jedoch vom Betriebssystem beziehungsweise von der auf dem System vorhandenen Bibliothek abhängig. Daher wird in dieser Implementierung eine Zeitmessung auf Basis des „time-stamp counter“ (TSC) verwendet. Zum Auslesen des TSC wird, wie in Kapitel 2.3.5 beschrieben, die RDTSC-Instruktion verwendet.

Die Implementierung der Frequenzerkennung mithilfe der RDTSC-Instruktion basiert auf der Arbeit von Gabriele Paoloni, welcher den Einsatz von RDTSC zum Benchmarken der Ausführungszeit von Programmen beschreibt [16]. Dabei wurde darauf geachtet, dass es zu keinen Fehlern durch ein nicht initiiertes EAX-Register kommt, da die CPUID-Operation den Inhalt des EAX-Registers als Eingabeparameter verwendet und entsprechend diesem Parameter andere Werte zurückliefert, was zu unterschiedliche Laufzeiten der Instruktion führt.

In Abbildung 4.2 ist die Schleife zur Bestimmung der aktuellen CPU-Frequenz in Pseudocode dargestellt. Dabei ist anzumerken, dass nicht nur die Anzahl der Durchläufe der Messschleife gemessen wird, sondern auch die Dauer der Signalauswertung. Dadurch wird erreicht, dass alle Messpunkte das gleiche Intervall haben.

Logging

Zur nachträglichen Auswertung und Analyse der Übertragung werden alle Messpunkte aufgezeichnet. Dazu wird zum Start des Empfängers ein großes Array angelegt, in welche Daten zur Laufzeit gespeichert werden. Beim Beenden des Programms werden alle gespeicherten Daten in eine Datei geschrieben.

```
1  ref = 0;
2  zähler = 0;
3  for(;;) {
4     gemessene_schleife();
5
6     current = asm_RDTSC();
7
8     zähler++;
9     if(current - ref > referenz_zyklen) {
10       signal_auswerten(zähler);
11       ref = current;
12       zähler = 0;
13    }
14 }
```

Abbildung 4.2: Pseudocode der Messschleife zur Bestimmung der aktuellen CPU-Frequenz.

Turbofrequenzabstufungen

Wie in Kapitel 2.3.3 beschrieben, besitzt der eingesetzte Prozessor mehrere Turbostufen. Dies hat zur Folge, dass die zwei, durch Empfänger und Sender aktiv gehaltenen CPU-Kerne, nicht zu einer Turbofrequenzabsenkung führen. Daher muss ein weiterer CPU-Kern dauernd aktiv gehalten werden, um zu erreichen, dass die Aktionen des Senders zu einer Turbofrequenzänderung führen. Daher startet der Empfänger beim Programmstart einen weiteren Thread, welcher durchgehend einen CPU-Kern auslastet. Würde dieser zusätzliche Thread bei jedem Empfangsvorgang von neuem gestartet werden, müsste die Wartezeit zwischen dem Starten der Empfangsbereitschaft und dem eigentlichen Senden verlängert werden. Ansonsten kommt es zu Übertragungsfehlern, weil der neu gestartete Thread zuerst auf dem ursprünglichen CPU-Kern des Empfängers ausgeführt wird und der Scheduler diesen erst nach kurzer Zeit auf einen weiteren CPU-Kern auslagert. Die gleichen Probleme treten auch auf, wenn man versucht mithilfe von mehreren Threads das Signal des Senders zu verstärken.

4.3 Signalerkennung

Das Herzstück des Empfängers ist die Signalerkennung und Auswertung. Aufgrund des Übertragungsprotokolls ist es nötig, dass zur Laufzeit alle Signale ausgewertet und auch die empfangenen Daten auf Übertragungsfehler überprüft werden. In Abbildung 4.3 ist die gesamte Aufzeichnung einer Datenübertragung von 10 Byte mit einer Sendedauer von einer Millisekunde pro Bit dargestellt. Der ein-

gezeichnete Schwellwert wird dabei von Hand bestimmt, um für die spätere Evaluation vergleichbare Ergebnisse zu erhalten und um die Schwellwerterkennung als Fehlerursache für Übertragungsfehler auszuschließen.

Der Schwellenwert lässt sich beim Start des Programms automatisch bestimmen. Dabei wird über einen kurzen Zeitraum die Dauer der Schleifendurchläufe gemessen und anschließend gemittelt. Hierdurch ergibt sich die durchschnittliche Anzahl an Schleifendurchläufe für eine Frequenzstufe. Durch das Aktivhalten weiterer CPU-Kerne wird die nächste Frequenzstufe erreicht und erneut die Messung durchgeführt. Durch anschließendes Mitteln beider Werte ergibt sich der Schwellwert.

Alle Messpunkte über dem Schwellwert werden dabei als eine Null erkannt und alle darunter liegenden als eine Eins. Die eigentliche Übertragung der Daten ist auf der ganz rechten Seite zu erkennen. Bei dem Abfall der Messpunkte auf der linken Seite unter den Schwellwert handelt es sich um den Startvorgang des Senders. Hier liest dieser die zu sendenden Daten ein und erstellt die Datenpakete. Zwischen dem Start des Senders und der Beginn des Sendeprozesses liegt eine Sekunde, in der der Sender schläft, um den Startprozess deutlich von der Datenübertragung in den Aufzeichnungen zu trennen.

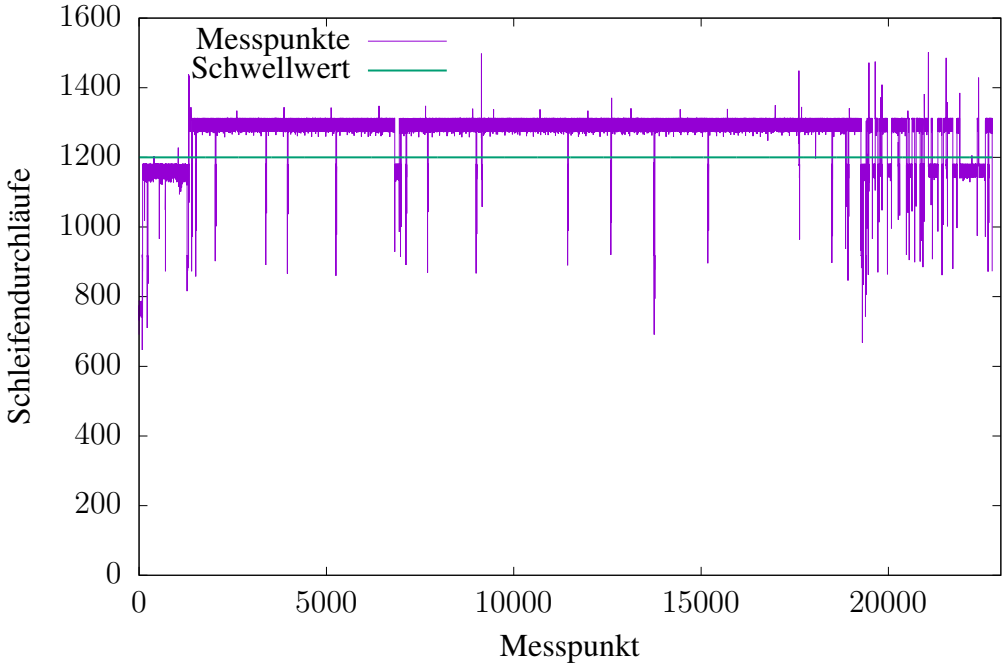


Abbildung 4.3: Darstellung der gesamten Aufzeichnung einer Datenübertragung von 10 Bytes an Daten.

In dem Zeitraum, in dem der Sender schläft, sind mehrere Stellen zu erkennen, in denen der Messpunkt unter dem Schwellwert liegt. Die einzelnen Messpunkten, die unterhalb des Schwellwerts liegen, resultieren aus Unterbrechungen des Empfängers durch den Scheduler. Liegen mehrere Messpunkte unterhalb des Schwellwerts, wie in Abbildung 4.4 zu erkennen, liegt dies daran, dass ein weiterer CPU-Kern aktiv ist, welcher zu einer Absenkung der Turbofrequenz führt. Neben diesen Störungen sind noch weitere einzelne Messpunkte zu sehen, welche niedriger als die anderen sind. Diese entstehen durch Speicherzugriffe beim Loggen.

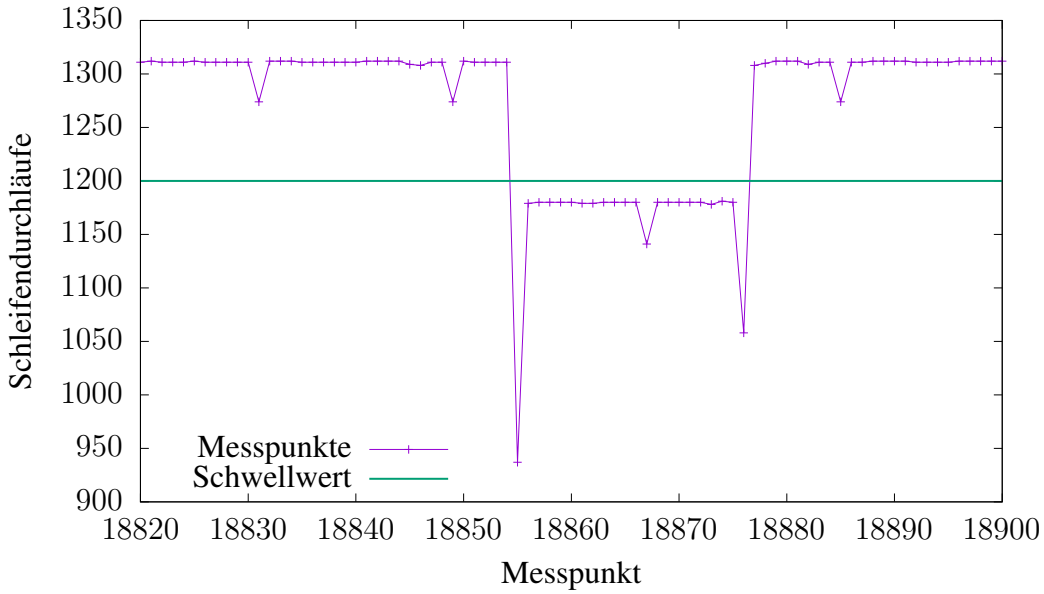


Abbildung 4.4: Störung durch einen anderen Prozess, welcher einen anderen CPU-Kern aktiviert hat.

Durch die in Kapitel 4.2 beschriebene Implementierung der Frequenzerkennung haben fast alle Messpunkte den gleichen Abstand zueinander. Pro Millisekunde werden dabei immer eine feste Anzahl an Messpunkten erzeugt. Durch die Information über die Sendedauer eines Signals, welche beim Programmstart angegeben werden muss, lässt sich anschließend feststellen, wie lange ein entsprechendes Signal anlag. Liegen mehr als die Hälfte alle Messpunkte über dem Schwellwert, so wurde eine Null übertragen, ansonsten eine Eins.

Aufgrund von Messungenauigkeiten, Unterbrechungen durch den Scheduler und das Fehlen eines Synchronisationssignal, welches den Anfang einer Übertragung und den Beginn eines Bits signalisiert, reicht das Abzählen der einzelnen

Messpunkte nicht aus. Daher wurde zusätzlich über eine Kantenerkennung versucht die Signalwechsel zu erkennen.

In Abbildung 4.5 ist eine einfache Datenübertragung mit Magic-Number und Daten sowie dem erkannten Signal dargestellt. Ein erkanntes Signal bei Datenpunkt 400 bedeutet dabei, dass eine Null erkannt wurde und bei 700 wurde entsprechend eine Eins erkannt. Damit ergibt sich Folgendes erkanntes Signal:

```
1  0011 0010 1000 0111 0000 0100 0001 0110 0100 0101 0001
2  1011 1100 0100 1000 0111 0011 1111 1111 1110 1111 1000 00
```

Durch Entfernen der ersten zwei Null-Bit ergibt sich die komplette Übertragung mit der Magic-Number `0xCA` am Anfang und anschließenden die Daten.

```
1  1100 1010 0001 1100 0001 0000 0101 1001 0001 0100 0110
2    C    A    1    C    1    0    5    9    1    4    6
3
4  1111 0001 0010 0001 1100 1111 1111 1111 1011 1110 0000
5    F    1    2    1    C    F    F    F    B    E    0
```

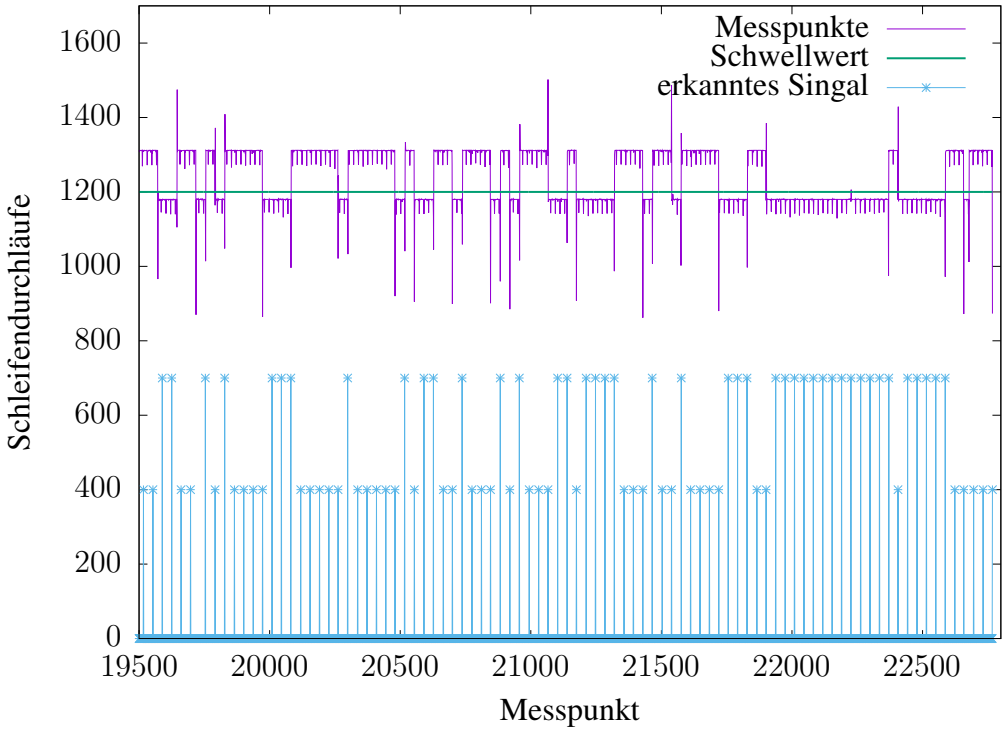


Abbildung 4.5: Ausschnitt aus der Abbildung 4.3 der eigentlichen Datenübertragung. In Blau ist dabei das erkannte Signal dargestellt. Wobei ein Punkt bei 400 eine erkannte Null und bei 700 eine erkannte Eins bedeutet.

```
-m t -t 5 -a 10 -p 4 -f 40B.img -l sender.log

> Das Programm wird als Sender gestartet
> Fünf Millisekunden Sendedauer pro Bit
> 10 Byte an Nutzdaten pro Datenpaket
> 4 Paritätsbytes
> Die Datei 40B.img wird gesendet
> Die Logginginformationen werden in die Daten sender.log
   geschrieben
```

Abbildung 4.6: Für den Sender werden meist die oben gezeigte Parameter verwendet.

```
-m r -t 5 -a 10 -p 4 -f out.img -l empfaenger.log

> Das Programm wird als Empfänger gestartet
> Fünf Millisekunden Sendedauer pro Bit
> 10 Byte an Nutzdaten pro Datenpaket
> 4 Paritätsbytes
> Die Empfangenen Daten werden in die Datei \glq out.img\grq{}
   geschrieben
> Die Logginginformationen werden in die Daten \glq empfaenger.
   log\grq{} geschrieben
```

Abbildung 4.7: Für den Empfänger werden meist die oben gezeigte Parameter verwendet.

4.4 Kommandozeilenparameter

Zur einfachen Evaluierung unterschiedlicher Optionen und Betriebsarten des Programms besitzt es Kommandozeilenparameter, über welche sich mehrere Optionen einstellen lassen. In Tabelle 4.1 sind alle existierenden Kommandozeilenparameter aufgelistet mit ihren Auswirkungen auf das Verhalten von Sender und Empfänger, wobei die Abbildung 4.6 und 4.7 jeweils einen typischen Aufruf eines Senders beziehungsweise Empfängers zeigen.

Option	Sender	Empfänger
-m <Modus>	‚s‘ Starten als Sender	‚r‘ Starten als Empfänger
-f <Datei>	Datei zum senden	Zieldatei
-s <Bytes>	-	Anzahl an zu empfangenen Bytes
-d <Datei>	-	liest Messpunkte aus Datei
-t <MS>	Millisekunden Sendezeit pro Bit	
-c <Code>	Fehlerkorrekturcode [rs (Reed-Solomon),...]	
-e <Encoding>	Bit-Encoding [NRZ,...]	
-l <Datei>	Logge in <datei>	
-a <Bytes>	Größe der Nutzdaten pro Paket in Byte	
-p <Bytes>	Anzahl Paritätsbytes pro Paket	
-x	Zusätzlicher Thread, welcher einen CPU-Kern auslastet	

Tabelle 4.1: Die Kommandozeilenparameter und ihre Bedeutung beim Ausführen des Programms als Sender oder Empfänger.

Kapitel 5

Gegenmaßnahmen

Der im Kapitel 3 beschriebene Covert Channel basiert darauf, dass durch Frequenzänderungen Informationen zwischen CPU-Kernen übertragen werden können. Neben dem eigentlichen Covert Channel ist es wichtig, den Fokus auch auf mögliche Gegenmaßnahmen zu legen, um den Angriff zu verhindern oder wenigstens zu erschweren. Im Folgenden wird darauf eingegangen, welche Hardware- und Softwareänderungen es hierfür gibt und wie die Covert Channel erkannt werden kann, um die Gegenmaßnahmen gezielter einsetzen zu können.

5.1 Hardwaregegenmaßnahmen

Die offensichtlichste Hardwaremaßnahme zur Verhinderung des Covert Channels, ist das Abschalten der Turbo-Boost Funktion. Dabei gehen jedoch die im Durchschnitt durch Turbo-Boost erzielte 6% Ersparnis an Ausführungszeit verloren [14]. Ist dies nicht hinnehmbar, so bleibt nur eine Anpassung der Hardware übrig, welche entsprechend vom Hersteller vorgenommen werden muss.

Ein erster Ansatzpunkt für Hardwareänderungen wäre es die Metrik anzupassen, aufgrund deren Turbo-Boost die Turbofrequenz festlegt. Das Entfernen der Abhängigkeit, dass die Anzahl der aktiven CPU-Kerne einen Einfluss auf die Turbofrequenz hat, macht den hier beschriebenen Covert Channel unmöglich. Neben der Anzahl der aktiven CPU-Kerne gibt es noch weitere Metriken wie beispielsweise die Temperatur oder den Energieverbrauch, von denen die Turbofrequenz abhängt. Das Design des hier beschriebenen Covert Channel ist bewusst so gehalten, dass die Metrik, welche die CPU-Frequenz beeinflusst, austauschbar ist. Daher ist das Entfernen einer Metrik keine geeignete Gegenmaßname zur Verhinderung des Covert Channels, da diese durch andere ersetzt werden kann. Es würde das Ausnutzen jedoch erheblich erschweren, weil es schwerer ist die übrigen Metriken auszunutzen, um eine Frequenzänderung hervorzurufen.

Hardwareanpassungen zur Verhinderung dieses Covert Channels sind nur mit viel Aufwand umsetzbar und garantieren nicht, dass kein weiterer Turbo-Boost Covert Channel existiert oder erzeugt wird. Das Ausschalten der Turbo-Boost Funktion ist deshalb der effektivste Hardwareschutz vor einem Covert Channel dieser Art.

5.2 Softwaregegenmaßnahmen

Nicht nur Hardwaremaßnahmen können den Covert Channel verhindern, sondern auch Softwaremaßnahmen. Das Betriebssystem ist dabei die zentrale Komponente, welches die Möglichkeit hat, Maßnahmen gegen den Covert Channel durchzusetzen. Da beim Design darauf geachtet wurde, dass der Covert Channels nicht auf Schnittstellen des Betriebssystems, zum Auslesen der Turbofrequenz oder ähnliches angewiesen ist, erschwert dies die Implementierung von möglichen Gegenmaßnahmen.

Das Betriebssystem kann auf mehrere Arten versuchen den Covert Channel zu verhindern. Durch gezielte Störungen lässt sich die Kommunikation stark einschränken sowie durch Anpassen von Energiesparfunktionen und Unterbinden des Bestimmens der aktuellen CPU-Frequenz auch ganz verhindern.

Im Nachfolgenden wird auf diese Punkte eingegangen und darauf geachtet, die Eignung gegen eine ideale Umsetzung des in Kapitel 3 beschriebenen Covert Channels zu prüfen. Dabei werden die in Kapitel 4 beschriebenen Implementierungseinschränkungen außen vor gelassen. Da sich die Implementierung leicht ändern lässt, das zugrundeliegende Prinzip des Covert Channels jedoch nicht.

5.2.1 Verhindern des Bestimmens der aktuellen Turbofrequenz

Der hier beschriebene Covert Channel basiert darauf, dass die aktuelle Turbofrequenz bestimmt werden kann. Wie in Kapitel 4.2 beschrieben, kann dies z.B. durch Auslesen der CPU-Frequenz über die vom Betriebssystem bereitgestellten Wege erfolgen. Aufgrund des Zieles der Unabhängigkeit vom Betriebssystem und der niedrigen Aktualisierungsrate dieser Werte wurde dieser Ansatz nicht gewählt. Das Ermitteln der aktuellen Turbofrequenz erfolgt stattdessen durch Messen der Schleifendurchläufe innerhalb eines vorgegebenen Zeitraums. Hierfür wird eine genaue Zeitmessung benötigt.

Der in Kapitel 4.2 vorgestellte Weg der Frequenzbestimmung benutzt den `time-stamp counter` (TSC) zur genauen Zeitbestimmung. Die vorgestellte Instruktion zum Auslesen des TSC, ist RDTSC. Die RDTSC-Instruktion überprüft, ob das `time stamp disable` (TSD) Flag gesetzt ist und liefert entsprechend die Werte des TSC zurück oder erzeugt eine Illegal Instruction Exception [21].

Das TSD-Flag kann hierbei dafür sorgen, dass RDTSC zu einer privilegierte Instruktion wird. Zusätzlich kann, durch das Setzen des `RDTSC exiting` Flags, veranlasst werden, dass ein Aufruf innerhalb einer virtuellen Maschine (VM) von RDTSC zu einer Exception führt [21]. Hierdurch hat das Betriebssystem die Möglichkeit die RDTSC-Instruktion abzufangen und dabei den TSC-Wert zu manipulieren.

Auf Systemen, bei denen RDTSC-Instruktionen nicht existieren oder erlaubt sind, kann die POSIX Funktion `clock_gettime` benutzt werden. Im Vergleich zu RDTSC hat `clock_gettime` eine geringere Genauigkeit [11].

Sowohl RDTSC-Instruktionen als auch die `clock_gettime` Funktion sind vom Betriebssystem manipulierbar. Eine geringere Auflösung der TSC-Werte oder zufällige größere Sprünge würden den Covert Channel jedoch nur erschweren und nicht verhindern. Wie im Kapitel 4.3 beschrieben, wurde bei der Implementierung darauf geachtet, dass leichte Zeitsprünge nicht zu Fehlern führen. Bei einer geringeren Auflösung der Zeiteinheit von TSC würde entsprechend die Bandbreite sinken, der Covert Channel jedoch nicht verhindert werden. Um das Signal des Senders zu rekonstruieren, muss lediglich das Nyquist-Shannon-Abtasttheorem erfüllt sein, was bedeutet, dass die Sendedauer eines Bits doppelt so hoch sein muss wie der kleinste durch die Zeitquelle unterscheidbare Zeitunterschied [39].

Die Manipulierung der TSC-Werte ist zudem mit Vorsicht zu betrachten, da dies unerwünschte Nebenwirkungen haben kann. Beispielsweise basiert bei Linux die Implementierung von `clock_gettime` sowie die Implementierungen des Advanced Programmable Interrupt Controller (APIC) auf dem TSC-Wert [26]. Wird der TSC-Wert vom Host-System manipuliert, bekommen virtuelle Maschinen dies nicht mit und nehmen diese Werte als gegeben an, was entsprechend zu verspäteten Interrupts oder falschen Zeitmessungen führen kann.

5.2.2 Energiesparfunktionen deaktivieren

Der in dieser Arbeit beschriebene Covert Channel basiert darauf, dass die Anzahl der aktiven CPU-Kerne Einfluss auf die Turbofrequenz hat. Ein CPU-Kern gilt als aktiv, wenn er in einem kleineren State als C3 ist. Wie in Kapitel 2.3.3 beschrieben, gilt ein CPU-Kern bei einem C-State von C3 oder höher als schlafend. Das Betriebssystem ist dabei in der Lage den maximalen C-State zu setzen.

Durch das generelle Verhindern, dass ein CPU-Kern in einen C3 oder höheren State betritt, lässt sich der Covert Channel verhindern, ohne dabei alle Vorteile von Turbo-Boost zu verlieren. Für Turbo-Boost sind hierbei alle CPU-Kerne aktiv, was entsprechend zu der Wahl der Turbofrequenz führt, welche für die maximale Anzahl der aktiven Kerne hinterlegt ist. Da die Turbofrequenz höher ist als die Basis-Frequenz, ergibt sich gegenüber der kompletten Abschaltung von Turbo-Boost ein Performancevorteil für Prozesse, die von einer höheren CPU-Frequenz

Anzahl ausgelasteter Kerne	alle C-States	maximal C2-State
idle	14,5 W	23,1 W
1	31,6 W	31,4 W
2	37,1 W	34,6 W
3	39,8 W	36,3 W
4	43,6 W	39,4 W
5	40,9 W	41,8 W
6	43,5 W	44,2 W
7	47,1 W	47,2 W
8	49,4 W	49,4 W

Tabelle 5.1: Messungen des Energieverbrauchs des Package mit RAPL (Kapitel 2.3.4). Zeigt den Unterschied des Energieverbrauches zwischen dem Normalzustand und nach dem Verbieten von C3- und höheren States.

profitieren. Wie im folgenden Abschnitt zu sehen, führt das im Leerlauf aktiv halten von CPU-Kernen zu einem höheren Energieverbrauch.

Analyse Energieverbrauch

Da heutige Rechenzentren meist nur eine durchschnittliche Auslastung von 6% erreichen, befinden sich viele Systeme die meiste Zeit im Leerlauf [23]. CPU-Kerne, die sich im Leerlauf befinden, werden normalerweise schlafen gelegt, um Energie zu sparen. Werden diese jedoch nicht schlafen gelegt, wird wie in Tabelle 5.1 zu sehen mehr Energie verbraucht.

Bei dem direkten Vergleich zwischen dem Energieverbrauch bei allen erlaubten C-States und wenn alle CPU-Kerne aktiv sind, zeigt sich, dass der Energieverbrauch bei bis zu 4 ausgelasteten CPU-Kernen mit allen erlaubten C-States höher ist, als wenn durchgehend alle CPU-Kerne aktiv sind. Dies hängt mit der Turbofrequenz zusammen. Wie in Kapitel 2.3.2 beschrieben, besitzt der hier eingesetzte Prozessor mehrere Turbofrequenzstufen, die erste bei bis zu zwei aktiven CPU-Kernen, die zweite bei bis zu vier aktiven CPU-Kernen und die letzte Stufe bei allen weiteren aktiven CPU-Kernen. Die unterschiedlichen Frequenzen sorgen entsprechend für einen anderen Stromvebrauch. So braucht eine höhere CPU-Frequenz mehr Energie als eine niedrigere [33].

5.2.3 Stören des Covert Channels

Neben dem Verhindern des Covert Channels kann auch durch das gezielte Stören die Nutzung des Covert Channels erschwert werden. Das Betriebssystem hat dabei zwei Möglichkeiten den Covert Channel zu stören. So kann es mithilfe des Schedulers den Sender und Empfangsprozess stören oder über durch Anpassen der Energiesparfunktionen Einfluss auf die Übertragung nehmen.

Stören mithilfe des Scheduler

Durch die Möglichkeit der Unterbrechung oder Wechselns des CPU-Kerns eines Prozesses ist der Scheduler in der Lage den Sender- und Empfängerprozess zu stören. Häufigere Unterbrechungen oder CPU-Kern Wechsel können beim Empfänger zu einem Synchronisationsverlust führen. Daneben kann auch durch gezieltes Verwalten der Prozesse ein schlafen legen des CPU-Kerns verhindern oder hinauszögern werden. Zusätzlich kann durch häufiges aufwecken unterschiedlicher Prozesse ein Rauschen erzeugt werden, was die Übertragung stört. Hierfür kann die Ausführung von Prozessen hinausgezögert oder zusätzliche Prozesse erzeugt werden, die nur zum Stören des Covert Channels existieren.

Stören mithilfe der Energiesparfunktion des Prozessors

Die C-States lassen sich nicht nur wie in Kapitel 5.2.2 beschrieben ausnutzen, um alle CPU-Kerne aktiv zu halten, sondern auch zum Stören der Kommunikation des Covert Channels. Dazu wird die Dauer des Wechsels eines C-States verlängert, um so längere Zeit einen CPU-Kern aktiv zu halten, wodurch die Sendezeit eines Bits verlängert werden muss, um ein Signal zu erhalten. Zusätzlich kann noch die Eigenschaft der unterschiedlichen Frequenzstufen benutzt werden, um die Kommunikation zu stören. Dazu wird versucht so lange wie möglich innerhalb einer Frequenzstufe zu bleiben, was bedeutet, dass das Aufwecken und schlafen legen von CPU-Kerne hinausgezögert wird.

5.3 Erkennen des Covert Channels

Neben dem Verhindern oder Stören des Covert Channel kann es auch sinnvoll sein ihn zu erkennen, um die zuvor beschriebenen Gegenmaßnahmen gezielt einsetzen zu können. Ein solches Verfahren zu Erkennung von frequenzbasierten Covert Channel wurde schon von Wu et al. vorgestellt [50]. Dabei erweiterten sie den Xen Hypervisor, um gezielt Events aufzuzeichnen. Anschließend werden diese mit vorherigen Aufzeichnungen verglichen, die als Referenzmuster für normales

Verhalten verwendet werden. Diese Methode zur Erkennung von frequenzbasierten Covert Channel funktioniert auch bei dem hier vorgestellten Covert Channel. Da ein Sendevorgang ziemlich von einem normalen Verhalten eines Programms abweicht. So pausiert sich der Sender ständig während eines Sendevorgangs, nur um kurz darauf aufzuwachen und einen kompletten CPU-Kern auszulasten. Im Vergleich zu den meisten Programmen, die eine Arbeit erledigen und anschließend auf ein Signal von Außen warten um mit ihrer Arbeit weiter fortzufahren. Dabei muss jedoch bedacht werden, dass die CPU-Frequenz mit entsprechend hoher Auflösung aufgezeichnet werden muss, ansonsten wird ein Sendevorgang nur als ein durchgehend ausgelasteter CPU-Kern erkannt, wodurch das Verhalten sehr stark einem normalen Programm ähnelt.

Kapitel 6

Evaluation

Das Ziel der Evaluation ist es, zu zeigen, dass die Implementierung des Covert Channels funktioniert und auch im Vergleich zu zuvor beschriebenen frequenzbasierten Covert Channel, eine hohe Übertragungsrate erreicht wird. Zu Beginn wird in Abschnitt 6.3 analysiert, welche Übertragungsrate Hardware- und Softwarebedingt maximal möglich ist und ob diese auch real erreicht werden kann. Selbst unter optimalen Bedingungen, bei denen versucht wurde, alle Störungen zu minimieren, kommt es zu Übertragungsfehlern aufgrund von Störungen durch andere Prozesse. Daher wird in Kapitel 6.4 untersucht wie viele Übertragungsfehler unter normalen Bedingungen auftreten. Um die dabei auftretenden Übertragungsfehler zu kompensieren und so mögliche erneute Übertragungen eines Datenpaketes zu verhindern, wurde das in Kapitel 3 beschrieben Übertragungsprotokoll mit entsprechender Fehlererkennung und Korrektur eingeführt. Anschließend wird in Kapitel 6.5 untersucht wie viele Paritätsinformationen benötigt werden, um Übertragungsfehler zu vermeiden. Da nicht nur kleine Daten über den Covert Channel übertragen werden sollen, wird im Kapitel 6.6 untersucht, mit welchen Nutzdatenraten bei größeren Datenmengen zu rechnen ist. Aufgrund der vielen Parameter, die sich zur Optimierung einstellen lassen und der Unberechenbarkeit von Fehlern, wird hier nicht versucht die maximal mögliche Nutzdatenrate zu erreichen, sondern eine Nutzdatenrate, die auch über eine längere Übertragung stabil ist. Am Ende wird noch in Kapitel 6.7 auf die Möglichkeit eingegangen, dass dieser Covert Channel auch zwischen virtuellen Maschinen eingesetzt werden kann. Dabei wird auch hier untersucht, was für eine maximale Übertragungsrate erreicht werden kann. Als Grundlage für die Versuche wird in Kapitel 6.1 die verwendete Methodik beschrieben und anschließend in Kapitel 6.2 den verwendeten Versuchsaufbau eingegangen.

6.1 Methodik

Die in der Auswertung verwendeten Messdaten wurden alle vom Empfänger oder Sender selbst aufgezeichnet. Beide speichern während des Empfangsvorgangs alle gemessenen Messpunkte. Zusätzlich stoppt der Sender die Übertragungszeit, um entsprechend die Nutzdatenrate zu erhalten, daneben wird noch aufgezeichnet wie viele Sendungswiederholungen es gab. Der Empfänger speichert neben den Messpunkten auch die Anzahl der fehlerhaft empfangenen Nachrichten. Empfänger sowohl als auch Sender schreiben am Ende der Übertragung alles in eine Datei, aus der anschließend die Werte der Auswertung entnommen wurden.

Bei der Auswertung werden dabei unterschiedliche Metriken verwendet. Am häufigsten wird dabei nicht die Übertragungsrate sondern die Nutzdatenrate verwendet. Dies hat den Hintergrund, dass es, bedingt durch das Übertragungsprotokoll, zu Sendungswiederholungen kommen kann und damit die reine Übertragungsrate keine Aussage darüber gibt, wie lange es dauert, Nutzdaten zu übertragen. Diese und die weiteren verwendeten Metriken werden im Folgenden beschrieben.

Übertragungsrate oder Bitrate Die Bitrate oder auch Übertragungsrate gibt an, wie viele Bits pro Sekunde übertragen werden. Dabei wird kein Unterschied gemacht, ob es sich bei den übertragenen Bits um Nutzdaten oder Verwaltungsdaten des Übertragungsprotokolls handelt.

Symbolrate Die Symbolrate gibt an, wie viele Symbole pro Sekunde Übertragen werden. Da bei diesem Covert Channel jedes Symbol nur ein Bit an Information trägt, gilt hier, dass die Symbolrate gleich der Bitrate ist. Die Einheit der Symbolrate ist Baud, abgekürzt Bd.

Sendedauer pro Bit Die Sendedauer pro Bit ist abgeleitet von der Symbolrate. Hierbei wird angegeben, wie lange ein Signal anliegt, um ein Bit zu senden.

Nutzdatenrate Bei der Nutzdatenrate wird ermittelt, wie viele Bits an Nutzdaten im Durchschnitt innerhalb einer Sekunde übertragen werden. Dargestellt in der Einheit bit/s. In den Versuchen wird zum Erhalten der Nutzdatenrate die Anzahl der übertragenen Nutzdatenbits durch die Dauer der Übertragung geteilt.

Kanalkapazität Die Kanalkapazität gibt die höchste Bitrate in bit/s an, mit der Informationen fehlerfrei übertragen werden können.

6.2 Versuchsaufbau

Das in den Versuchen verwendete Testsystem besitzt einen Intel® Xeon® Silver 4108 Prozessor mit einer Basisfrequenz von 1,6 GHz und einer maximalen Turbofrequenz von 3,0 GHz [20]. Bei dem Prozessor handelt es sich um einen 64-bit 8-Kern-Prozessor mit 16 Threads und einer Thermal Design Power von 85 W. Der Prozessor unterstützt Intel® Turbo-Boost 2.0 mit den in Tabelle 2.1 angegebenen Turbofrequenzen und basiert auf der Skylake Mikroarchitektur, welche in Kapitel 2.3.1 näher beschrieben wurde. Zudem befindet sich das Testsystem in einem klimatisierten Raum.

Auf dem Host sowie in den virtuellen Maschinen kommt die Server Edition von Fedora 28 als Betriebssystem zum Einsatz. Auf dem Host-System sowie in den virtuellen Maschinen wurden neben OpenSSH keine weiteren Dienste installiert und auch keine sonstigen Änderungen vorgenommen, welche dazu führen könnten, dass Störungen reduziert werden oder die den Covert Channel auf sonstige Weise begünstigen könnten. Alle Programme des Covert Channels werden von einem Benutzer ohne Administratorrechte ausgeführt.

Für die Virtualisierung der virtuellen Maschinen wird die Kernel-based Virtual Machine (KVM) [3] eingesetzt. Auch hier wurden keine weiteren Modifikationen vorgenommen. Bei Versuchen mit virtuellen Maschinen erhält jede Maschine jeweils zwei GiB Arbeitsspeicher und zwei virtuelle CPU-Kerne.

Bei allen Versuchen ist die Hyper-Threading Funktion des Prozessors deaktiviert. Dies verhindert, dass in der Implementierung nicht auf den Sonderfall eingegangen werden muss, bei dem Sender und Empfänger auf dem gleichen physischen CPU-Kern laufen. Mithilfe von Hyper-Threading lässt sich auch ein Covert Channel erzeugen, welcher jedoch auf einem anderen Konzept beruht, als der in dieser Arbeit beschriebene. Bei einem auf Hyper-Threading basierenden Covert Channel wird die Eigenschaft ausgenutzt, dass sich, durch Berechnungen des Senders, die Anzahl der Schleifendurchläufe innerhalb eines Zeitraumes ändert. Hierbei basiert die Änderung nicht auf einer Frequenzänderung, sondern auf der Auslastung des Rechenwerks. Von Wang und Lee wurden schon mehrere Covert Channel untersucht, die unter anderem auf Hyper-Threading basieren [45]. Durch die gestiegenen Sicherheitsbedenken bei dem Einsatz von Hyper-Threading hat sich OpenBSD Mitte 2018 dazu entschlossen Hyper-Threading zu deaktivieren [27].

6.3 Maximale Übertragungsrate

Das Ziel der Bestimmung der maximalen Übertragungsrate ist es, herauszufinden, mit welcher Übertragungsrate Daten zumindest teilweise und unter optimalen

Bedingungen fehlerfrei Übertragen werden können. Zuerst wird dabei bestimmt, welche limitierenden Faktoren es Hardware- und Softwareseitig gibt. Dabei wird die Änderungsrate der Turbofrequenz, die Samplingrate der CPU-Frequenz und die Anzahl der Störungen bedacht, da diese Punkte Einfluss auf die Übertragungsrate haben.

Wie in Kapitel 2.3.3 beschrieben, überprüft Turbo-Boost jede Millisekunde die Bedingungen, welche die Turbofrequenz beeinflussen und passt diese gegebenenfalls für die nächste Millisekunde an. Dies ist auch der begrenzende Faktor für die maximale Übertragungsrate, da die Samplingrate der CPU-Frequenz mithilfe des TSC-Werts um einiges höher ist. Beispielsweise werden in der in dieser Arbeit beschriebenen Implementierung 10 Messpunkte pro Millisekunde erhoben. Mit dem limitierenden Faktor der Änderungsrate der Turbofrequenz ergibt sich eine theoretisch maximale Übertragungsrate von 1000 bit/s.

Um zu zeigen, dass es möglich ist diese Übertragungsrate auch zu erreichen, wird der in Kapitel 3.3 beschriebene Entwurf etwas angepasst. Der Entwurf ist darauf ausgelegt Übertragungsfehler zu erkennen und zu beheben. Der dafür benötigte Protokoll-Overhead ist für diesen Versuch unnötig und verursacht nur eine Erhöhung der Sendezeit. Für diesen Versuch werden dementsprechend die Paket-ID sowie die Paritätsbytes am Ende entfernt und auch die Bestätigung eines erfolgreich empfangenen Pakets entfällt. Der Sender sendet somit alle Nutzdaten in einem Datenpaket und geht davon aus, dass der Empfänger alles erfolgreich empfängt.

Ohne ein entsprechendes Betriebssystem, welches alle Hintergrundaktivitäten verhindert, lassen sich Störungen nicht verhindern. Um den Einfluss von Störungen dennoch zu minimieren, wird mithilfe des beim Kernel-Start angegebenen Parameters `isolated_cores` dem Scheduler das Verwalten der angegebenen CPU-Kerne untersagt [1]. Für diesen Versuch werden die CPU-Kerne zwei, drei und vier isoliert. Wie in Tabelle 2.1 zu sehen, ist der erste Turbofrequenzsprung bei drei aktiven CPU-Kernen. Daher muss neben dem Sender und Empfänger noch ein weiterer CPU-Kern durchgehend aktiv gehalten werden. Wie in Kapitel 4.2 beschrieben, startet normalerweise der Empfänger einen weiteren Thread, welcher vom Scheduler automatisch auf einen anderen Kern verschoben wird. Da der Scheduler hier den CPU-Kern nicht verwalten darf, wird der Thread nicht auf einen weiteren CPU-Kern verschoben und läuft somit parallel zum Empfängerprozess auf dem gleichen CPU-Kern. Dies stört nur den Empfängerprozess, sorgt jedoch nicht dafür, dass ein weiterer CPU-Kern ausgelastet wird. Aus diesem Grund wird das Programm `stress` [9] verwendet, um einen zusätzlichen CPU-Kern auszulasten und damit aktiv zu halten.

Bei diesem Versuch wurden 10 Byte an Daten mit dem zuvor beschriebenen abgeänderten Nachrichtenaufbau gesendet. Dabei enthält das Paket nur die Magic-Number sowie die zu sendenden Daten. Der Versuch wurde dabei 20 mal

Anzahl betroffene Nachrichten	Fehlerart
7	keine Fehler
2	Einzelbitfehler in Magic-Number
6	ein Einzelbitfehler
1	zwei Einzelbitfehler
2	Einzelbitfehler und Bündelfehler. 4 Bit beschädigt
1	Bündelfehler über 5 Bits
1	Auswertungsfehler

Tabelle 6.1: Auswertung der aufgetretenen Übertragungsfehler beim Senden von 20 Nachrichten à 10 Byte mit einer Übertragungsrate von 1000 bit/s.

Übertragungsrate	Fehlerrate
1000 bit/s	13,5 %
500 bit/s	7,25 %
333 bit/s	0,19 %
250 bit/s	0,44%

Tabelle 6.2: Fehlerrate in Abhängigkeit der Übertragungsrate. Durch Senken der Übertragungsrate verringert sich auch die Fehlerrate.

wiederholt und die aufgetretenen Übertragungsfehler analysiert, welche in Tabelle 6.1 dargestellt sind. Am schwerwiegendsten sind dabei die Einzelbitfehler innerhalb der Magic-Number sowie der eine aufgetretene Auswertungsfehler, da diese nicht mithilfe eines Error-Correction-Codes behoben werden können und alle nachfolgenden Bits beschädigt sind, oder die Nachricht nicht erkannt wird. Unter einem Auswertungsfehler ist hierbei ein Fehler zu verstehen, bei dem ein Bit von der Auswertung nicht erkannt wurde. Dies kann beispielsweise passieren, wenn der Empfängerprozess kurzzeitig unterbrochen wird und dadurch die erwartete Signaldauer nicht mehr mit der gemessenen Zeit übereinstimmt. Werden die Übertragungen mit Einzelbitfehler in der Magic-Number und der eine Auswertungsfehler als komplett fehlgeschlagene Übertragung gezählt und somit alle übertragenen Bits als falsch empfangen gewertet, so ergibt sich eine Fehlerrate von 13,5%. In Tabelle 6.2 sind die Fehlerraten mehrerer Übertragungsraten aufgelistet. Trotz Übertragungsfehlern konnten 7 Nachrichten fehlerfrei übertragen werden und es konnte somit gezeigt werden, dass unter optimierten Bedingungen die theoretische maximale Übertragungsrate von 1000 bit/s auch erreicht werden kann.

6.4 Fehlerrate bei unterschiedlicher Übertragungsraten

Bei den vorherigen Messungen in Kapitel 6.3 hat sich gezeigt, dass die theoretische maximale Übertragungsrate von 1000 bit/s erreicht werden kann. Dabei muss jedoch mit vielen Übertragungsfehler gerechnet werden. Durch Senken der Übertragungsrate hat sich gezeigt, dass die Anzahl der Übertragungsfehler abnimmt. Daher ist das Ziel dieser Untersuchung aufzuzeigen, welche Arten von Fehlern bei unterschiedlichen Übertragungsraten unter realitätsnahen Bedingungen auftreten. Um die realitätsnahen Bedingungen zu bieten, wurde auf jegliche Optimierung zur Unterdrückung von Störfaktoren verzichtet. Der Versuchsaufbau wurde analog zu Kapitel 6.2 aufgebaut, jedoch mit Ausnahme des Kommunikationsprotokolls. Dieses wurde, wie im vorherigen Versuch, angepasst, um die unverfälschten Übertragungsfehler zu erhalten. Würde das in Kapitel 3.3 beschriebene Protokoll eingesetzt werden, so wird der Error-Correction-Code einzelne Bitfehler korrigieren und fehlerhaft empfangene Pakete erneut vom Sender anfordern, was bedeutet, dass sich am Ende keine Übertragungsfehler mehr in den empfangenen Daten befinden.

Wie im vorherigen Versuch, wurde ebenfalls pro Durchlauf 10 Byte an Daten übertragen und jede Übertragung 40 mal wiederholt, um den Einfluss durch kurzzeitige Hintergrundaktivitäten, wie Beispielsweise Cronjobs, zu minimieren. Die 10 Byte an Nutzdaten wurden gewählt, weil sich bei dieser Größe schon viele Fehler gezeigt haben und die Übertragungszeit pro Nachricht noch gering ist. Je länger eine Nachricht ist, desto wahrscheinlicher ist es, dass es zu irreparablen Fehlern kommt. Während die Magic-Number-Fehler sowieso unabhängig von der Nachrichtenlänge sind, sind die Bitfehler sowie die Auswertungsfehler jeweils abhängig von der Nachrichtenlänge. Je länger ein Sendevorgang, desto wahrscheinlicher ist es, dass es zu Störungen kommt, die einen Übertragungsfehler verursachen.

In Abbildung 6.1 ist die Summe der Fehler dargestellt, die während der 40 Wiederholungen pro Sendedauer aufgetreten sind. Im Bereich von fünf bis 16 Millisekunden Sendedauer pro Bit ist deutlich zu erkennen, dass über die Hälfte der empfangenen Datenpakete keine Übertragungsfehler beinhaltet. Innerhalb dieses Bereiches treten auch am wenigsten Fehler, verursacht durch die fehlerhafte Erkennung der Magic-Number oder durch die Auswertung, auf. Diese Fehler sind am schwerwiegendsten, da sich diese nicht über einen Error-Correction-Code beheben lassen. Im Gegensatz zu den im Versuch aufgetretenen Bitfehler, welche sich durch einen Error-Correction-Code beheben lassen.

Mit steigender Sendedauer pro Bit steigt auch die Anzahl an Fehlern durch die Auswertung drastisch an. Besonders stark zu erkennen ist dies in Abbildung

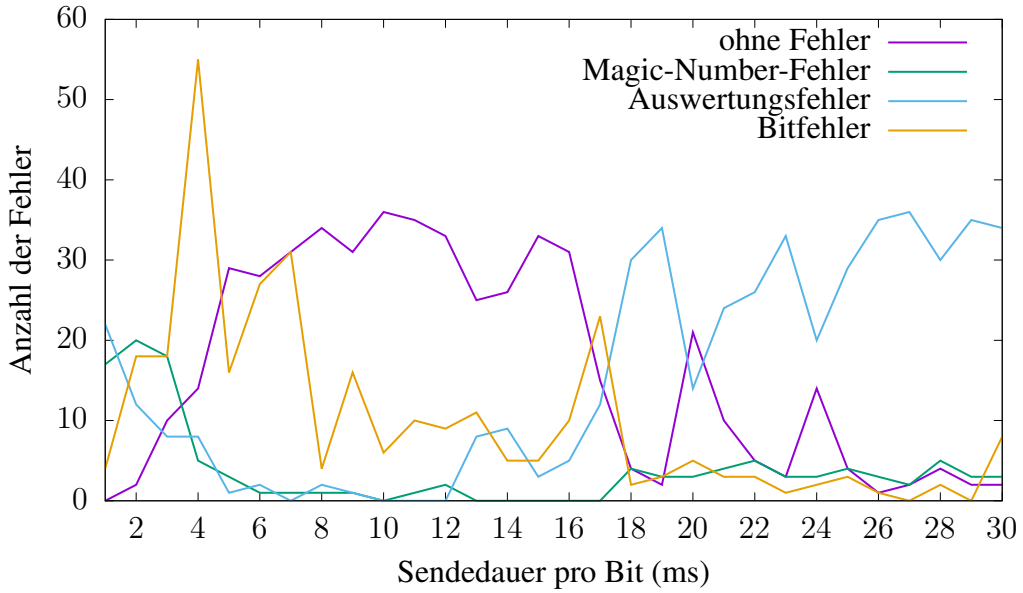


Abbildung 6.1: Darstellung der Fehler einer 40 mal wiederholten Übertragung von 10 Byte an Nutzdaten mit unterschiedlicher Sendedauer pro Bit. Im Bereich von fünf bis 16 Millisekunden Sendedauer pro Bit treten am wenigsten Fehler auf.

6.2, welche die Fehlerrate pro Millisekunden Sendedauer pro Bit zeigt. Dabei wurden Magic-Number-Fehler und Auswertungsfehler als kompletter Verlust der Nachricht gewertet, da sich diese selbst mit einem Error-Correction-Code nicht mehr beheben lassen.

Der starke Anstieg an Fehlern durch die Auswertung liegt daran, dass diese auf den Bereich von zwei bis 10 Millisekunden Sendedauer pro Bit optimiert ist. Je länger die Sendedauer eines Bits, desto mehr Störungen können innerhalb dieser auftreten. Hinzukommend steigt auch die Wahrscheinlichkeit, dass Synchronisationsfehler durch Unterbrechungen des Schedulers auftreten. Besonders auffällig sind die zwei Einbrüche der Fehlerrate bei 20 und 24 Millisekunden Sendedauer pro Bit. Da an dem System während des gesamten Versuchs keine Änderung vorgenommen wurde und auch Hintergrundaktivitäten einen gegenteiligen Effekt haben, müssen diese Anomalien mit der Signalerkennung zusammen hängen.

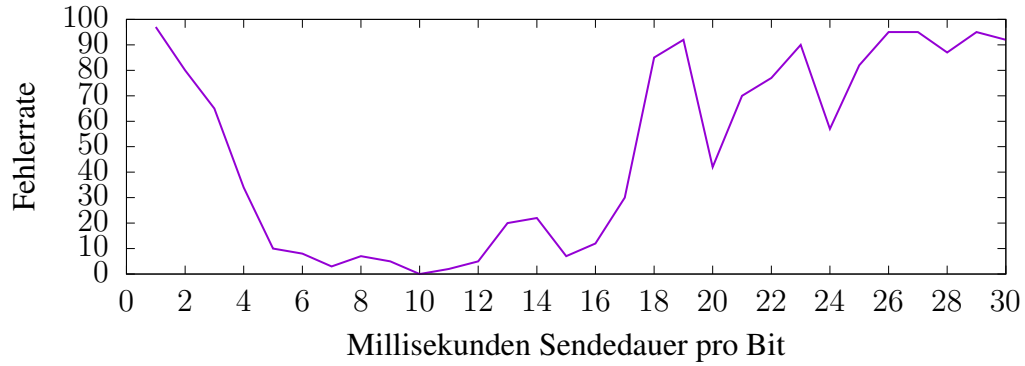


Abbildung 6.2: Auswertung der Fehlerrate aus Abbildung 6.1. Datenpakete mit Magic-Number-Fehlern und Pakete mit Auswertungsfehler zählen dabei als komplett Fehlerhaft empfangen, da diese Übertragung irreparabel beschädigt sind und auch von einem Error-Correction-Code nicht wiederhergestellt werden können.

6.5 Anzahl an Paritätsbytes

Wie in Abbildung 6.1 zu sehen, sind viele Übertragungen innerhalb der fünf bis 16 Millisekunden Sendedauer pro Bit fehlerfrei, jedoch nicht alle. Daher muss bei jeder Übertragung mit Fehlern gerechnet werden. Für diesen Fall wurde das in Kapitel 3.3 beschriebene Kommunikationsprotokoll ausgelegt. Durch den Einsatz eines Error-Correction-Codes können Übertragungsfehler erkannt und falls möglich auch gleich behoben werden. Das Ziel dabei ist es die Anzahl an Datenpaketen, die Aufgrund von Übertragungsfehlern mehrfach gesendet werden, zu minimieren. Hierfür ist die Anzahl der vorhandenen Paritätsbytes entscheidend. Der eingesetzte Reed-Solomon-Code schreibt keine feste Anzahl an Paritätsbytes vor. Abhängig von der Anzahl der eingesetzter Paritätsbytes ändert sich die Anzahl der als fehlerhaft erkannten und behebbaren Bytes. Dabei können genau so viele Fehler erkannt werden wie Paritätsbytes vorhanden sind und davon die Hälfte korrigiert werden. Aus diesem Grund wird hier untersucht,welchen Einfluss die Anzahl der Paritätsbytes auf Sendungswiederholungen und auf die vom Error-Correction-Code fälschlicherweise als korrekt eingestuften Daten hat.

Für diesen Versuch wurde dabei das im Entwurf beschriebene Übertragungsprotokoll verwendet. Wie im Abschnitt 6.4 gezeigt Versuch, gibt es bei einer Sendedauer von 10 Millisekunden pro Bit keine Magic-Number-Fehler oder Auswertungsfehler, die beide nicht durch einen Error-Correction-Code behoben werden können, weil dabei entweder gar keine Nachrichten erkannt werden oder meist ab der Hälfte der Nachricht alle folgenden Daten falsch sind. Um diese Fehler zu minimieren wurde eine Sendedauer von 10 ms/bit gewählt. Die Größe an Nutzdaten

pro Paket ist 10 Byte, welche ebenfalls dem vorhergehenden Versuch entspricht. Damit auch einige Datenpakete gesendet werden, werden bei diesem Versuch insgesamt 1000 Byte an Nutzdaten übertragen.

Die Anzahl der untersuchten Paritätsbytes wurde auf den Bereich von zwei bis 20 Bytes eingegrenzt. Bei zwei Paritätsbytes können Fehler in bis zu zwei Bytes erkannt und ein Byte korrigiert werden. Um sämtlich mögliche Fehler innerhalb der 10 Byte Nutzdaten reparieren zu können, werden 20 Byte an Parität benötigt, weswegen dies als obere Grenze gewählt wurde.

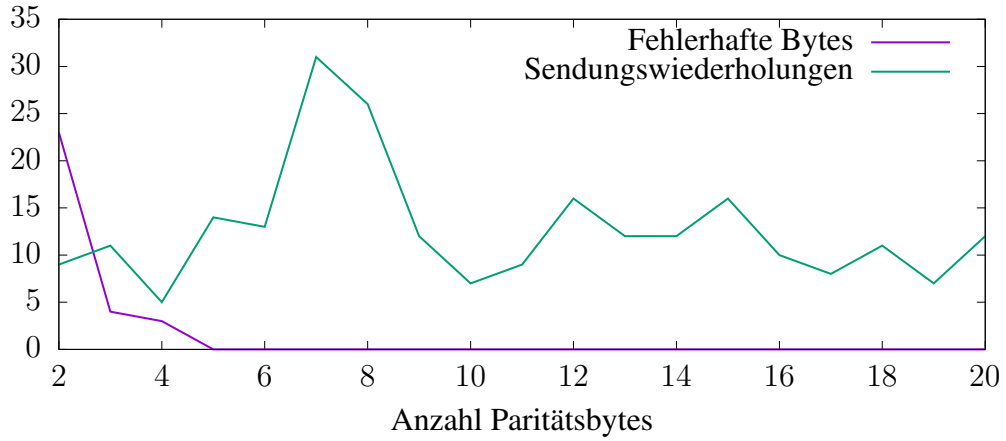


Abbildung 6.3: Darstellung der Anzahl an Fehlern nach der Anwendung des Reed-Solomon Error-Correction-Code in Abhängigkeit der Anzahl verwendeter Paritätsbytes. Ab 5 verwendeten Paritätsbytes erkennt der Reed-Solomon-Code alle Übertragungsfehler.

In Abbildung 6.3 sind die Ergebnisse dargestellt. Hierbei ist zu erkennen, dass ab 5 verwendeten Paritätsbytes der Reed-Solomon-Code alle aufgetretenen Übertragungsfehler erkannt und, falls eine Reparatur möglich war, diese auch repariert hat. Falls eine Reparatur nicht mehr möglich war, wurde das Datenpaket neu versendet. Die Anzahl der Sendungswiederholungen ist ebenfalls in der Abbildung zu sehen, welche durch vom Empfänger nicht erkannte Nachrichten oder aufgrund von anderen Übertragungsfehlern entstehen. Der starke Anstieg an Sendungswiederholungen bei 7 oder 8 verwendeten Paritätsbytes, liegt Wahrscheinlich an einem erhöhten Aufkommen an Störungen während des Versuches und den, wie in Abschnitt 6.6 vorgestellten, resultierenden Folgefehlern. Ein protokollieren aller Aktivitäten des Schedulers während der Versuche ist leider nicht möglich, ohne dabei Einfluss auf den Versuch selbst zu nehmen, da diese Protokollierung wiederum selbst CPU-Zeit benötigt und somit die Ergebnisse verfälscht.

6.6 Nutzdatenrate bei größeren Datenmengen

Wie schon in Kapitel 3.3 erwähnt, steigt die Wahrscheinlichkeit für einen irreparablen Fehler mit der Dauer einer Übertragung. Um dieses Problem bei größeren Nutzdaten zu umgehen, wurden diese in einzelne Datenpakete aufgeteilt. Das Ziel bei diesem Versuch ist es, zuerst mit kleineren Daten von 100 Byte die optimale Übertragungsrate zu ermitteln und anschließend Übertragungen von 512 Byte zu testen, was der Größe eines 4096 Bit RSA-Schlüssels entspricht. Zudem wird dabei untersucht, welche Fehler hierbei auftreten.

Um einen ersten Eindruck zu bekommen, wie sich die Nutzdatenrate in Abhängigkeit von Paketgrößen verhält, wurde in der ersten Messreihe eine 100 Byte große Datei übertragen, welche im Voraus aus Zufallszahlen generiert wurde. Die Dateigröße von 100 Byte wurde dabei gewählt, weil hierbei mehrere einzelne Nachrichten gesendet werden müssen und so auch mögliche unerwünschte Effekte der Implementierung des Übertragungsprotokolls auftreten können.

Um kleine Ausreißer auszugleichen, wurde jede Messung 10 mal wiederholt und der Durchschnitt der Messung gebildet. Ausgewertet wurde dabei die Datenübertragungsrate des Senders. Dazu wurde die Zeit gemessen, die benötigt wird, um die Daten komplett vom Sender zum Empfänger zu übertragen, und vom Empfänger bestätigt zu bekommen. Um die Datenübertragungsrate zu erhalten, wurden anschließend die 100 Byte durch die für die Übertragung benötigte Zeit geteilt. Mögliche Fehler, die auf fehlerhaften Erkennungen des Error-Correction-Codes beruhen, wurden hier nicht berücksichtigt. Zudem wurde die Anzahl an Paritätsbytes auf vier festgelegt.

In Abbildung 6.4 sind die Ergebnisse der Untersuchungen dargestellt. Eine Nutzdatenrate von 63,4 bit/s wird dabei bei einer Datengröße von 14 Byte pro Nachricht mit einer Übertragungsrate von 250 bit/s erreicht. In der Grafik zeigt sich, dass sich die Bandbreite von 250 bit/s für den hier getesteten Fall optimal eignet. Eine höhere Übertragungsrate hat die gleiche Anzahl an Sendungswiederholungen, jedoch ist die Sendedauer eines Bits kürzer. Vor allem bei niedrigeren Übertragungsraten kommt es zu vielen Sendungswiederholungen, wie in Abbildung 6.5 zu sehen ist. Erwartungsgemäß steigt die Nutzdatenrate mit der Größe der Daten pro Nachricht, solange es zu keinen Sendungswiederholungen kommt, da durch erneute Übertragungen die durch eine größere Paketgröße erlangten Geschwindigkeitsverbesserungen zunichte gemacht werden.

Für den anschließenden Versuch mit 512 Byte an Nutzdaten wurden, basierend auf den Ergebnissen aus Abbildung 6.4 eine Datengröße von 14 und 16 Byte gewählt sowie als Übertragungsrate 250, 200 und 166 bit/s. Die höchste Nutzdatenrate von 56 bit/s hat sich dabei bei einer Übertragungsrate von 166 bit/s und einer Datengröße von 16 Byte ergeben.

In den Versuchen hat sich gezeigt, dass die Zeit, die ein CPU-Kern zum umschalten zwischen Basis- und Turbofrequenz benötigt, teil stark variiert. Während des Sendevorgangs reicht es aus, wenn der CPU-Kern des Senders nur die Basisfrequenz nutzt, da dessen CPU-Kern lediglich aktiv gehalten werden muss. Für den Empfänger ist es jedoch wichtig, dass dessen CPU-Kern die Turbo-Frequenz nutzt, da er diese misst. Der Empfänger ist darauf angewiesen, dass der Sender erst Daten sendet, wenn er die Turbofrequenzen nutzt. Während es in den Ergebnissen teilweise zu Verzögerungen von bis zu 300 Millisekunden kam, hat sich in extra Messungen gezeigt, dass die Verzögerung im Durchschnitt bei 10 Millisekunden liegt. Eine längere Zeit für den Wechsel zu Turbofrequenz führt in Folge dazu, dass der Empfänger den Anfang des Datenpaketes verpasst und so die komplette Übertragung mit der entsprechenden Timeout-Zeit des Senders warten muss, bis er die nächste Nachricht erkennt.

Bei diesem Versuch zeigte es sich, dass diese Timeout-Zeiten nicht richtig dimensioniert waren, was besonders zu Folgefehlern führt, wenn der Empfänger in der Hälfte, der von Sender gesendeten Nachricht, den Beginn einer Nachricht erkennt und von dieser Stelle an das Datenpaket einliest. Dies führt dazu, dass der Sender teils schon mit dem wiederholten Senden des Datenpaketes begonnen hat, bevor der Empfänger empfangsbereit ist. Dies führt zu einem Kreislauf, in dem der Empfänger ein Phantompaket empfängt und dies nicht erkennt. Währenddessen erwartet der Sender eine Bestätigung des gesendeten Paketes und leitet eine Sendungswiederholung ein, nachdem diese nicht empfangen worden ist. Um dieses Problem zu beheben, kann nach beispielsweise 10 Sendungswiederholungen der Sender extra lange warten damit der Empfänger auch sicher beim nächsten Sendevorgang im Empfangsmodus ist.

Ein weiterer Fall, bei dem sich Sender und Empfänger in einer Endlosschleife von Sendungswiederholungen wiederfinden tritt auf, wenn eine Empfangsbestätigung verloren geht und das erneute versendete Paket derart beschädigt wird, dass die Paket-ID nach der Fehlerkorrektur der nächsten Paket-ID entspricht. Dies führt dazu, dass der Empfänger auf Paket N+2 wartet, der Sender allerdings immer noch die Bestätigung von Paket N erwartet und dies daher endlos wiederholt. Aus diesem Fall können sich Sender und Empfänger nicht selbst befreien. Die Übertragung muss von neuem gestartet werden. Analog dazu ergibt sich der umgekehrte Fall, bei dem der Sender eine Nachricht mit einer höheren Paket-ID sendet, als der Empfänger erwartet, was durch einen Übertragungsfehler innerhalb der ACK-Nachricht entstehen kann.

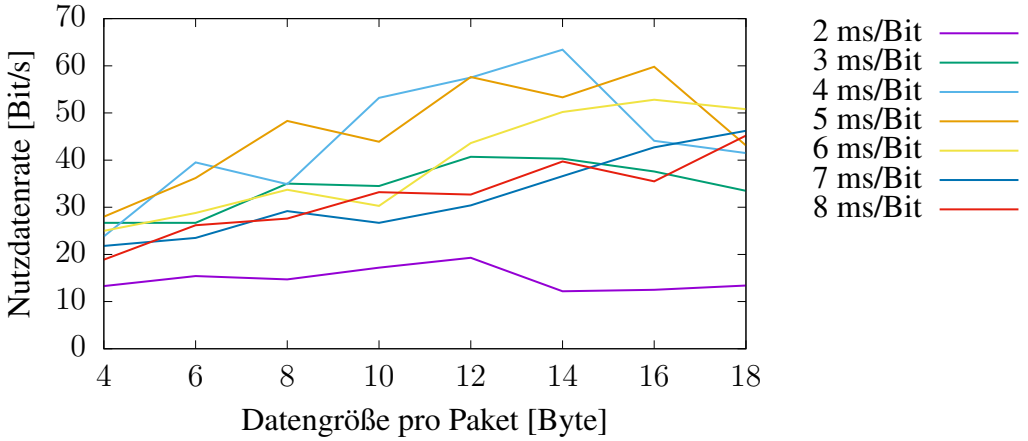


Abbildung 6.4: Die Ergebnisse der Nutzdatenrate bei einer Übertragung von 100 Byte an Daten.

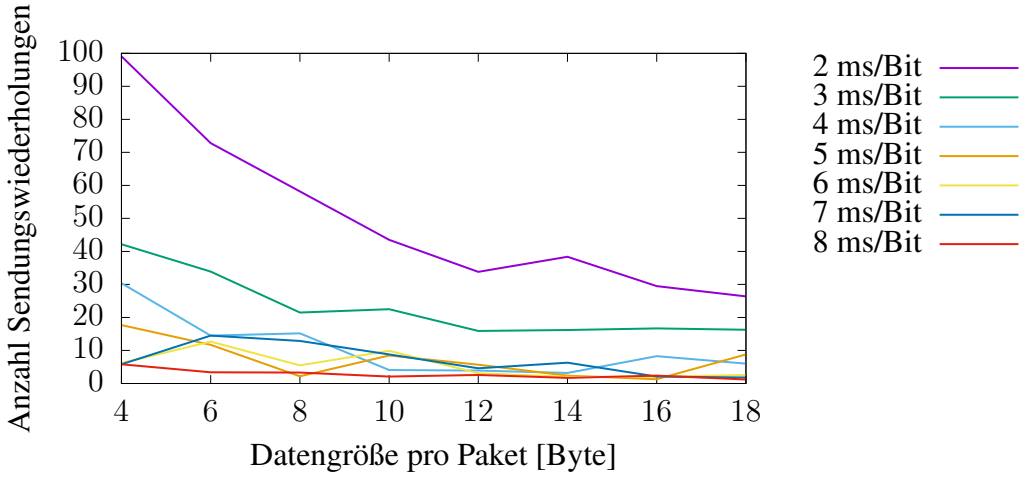


Abbildung 6.5: Dargestellt ist die durchschnittliche Anzahl an Paketen, die mehr als einmal gesendet wurden in Abhängigkeit der Sendezeit pro Bit bei einer Übertragung von 100 Byte an Daten.

6.7 Covert Channel zwischen virtuellen Maschinen

Durch die Beschaffenheit des Covert Channels ergibt sich nicht nur ein Kommunikationskanal zwischen zwei Prozessen auf einem Host, sondern auch ein Kommunikationskanal zwischen zwei virtuellen Maschinen. Bei den ersten Versuchen des Covert Channels zwischen zwei virtuellen Maschinen hat sich gezeigt, dass mehrere kleine Anpassungen der Einstellungen des Covert Channels benötigt werden, damit dieser funktioniert. Wie in Abbildung 6.6 zu sehen werden hier weniger Schleifendurchläufe innerhalb des vorgegebenen Zeitramens erreicht als direkt auf dem Hostsystem. Dies liegt an den längeren Verarbeitungszeiten für die CPUID- und RDTSC-Instruktionen, weil die entsprechend vom Hypervisor abgefangen werden und dieser sie selbst beantwortet. Zudem ist zu erkennen, dass es sehr lange dauert, bis die maximale Turbofrequenz erreicht wird und somit eine Kommunikation möglich ist. Daher wurde beim Empfänger die Wartezeit zwischen Ende der empfangenen Übertragung und Beginn des Sendevorgangs der ACK-Nachricht auf die Sendedauer eines Datenpaketes erhöht. Beim Sender wurde entsprechend der Timeout des Empfangsvorgangs auf die dreifache Sendedauer eines Datenpaketes angepasst.

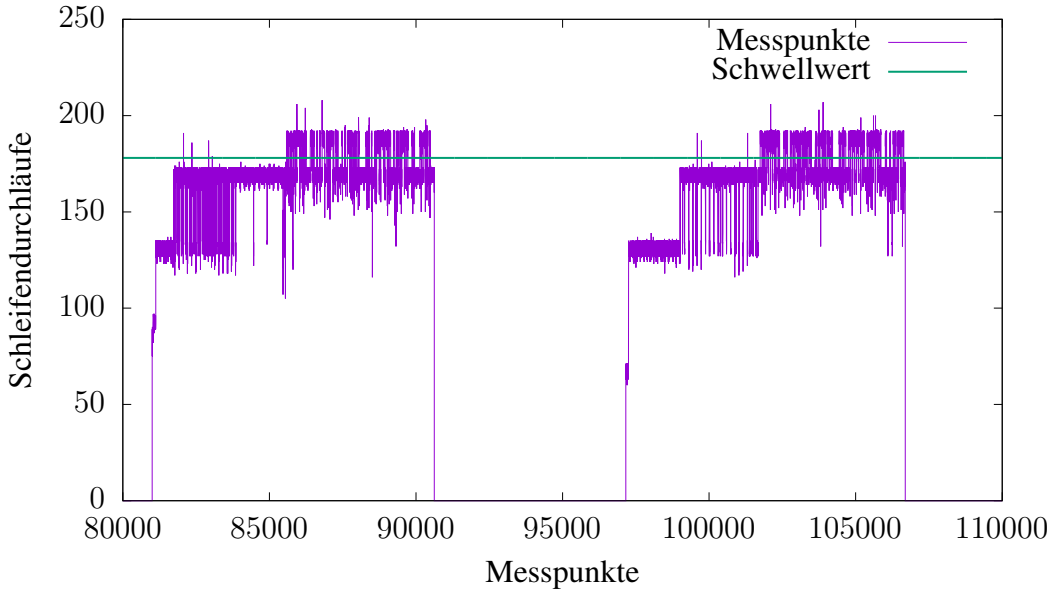


Abbildung 6.6: Ein Ausschnitt der erfassten Datenpunkte des Empfängers. Das dargestellte Schwellwert ist die Grenze zwischen der maximalen und der nächst niedrigeren Turbofrequenz.

Die Ergebnisse der ersten Untersuchungen des Covert Channels zwischen zwei virtuellen Maschinen sind in Abbildung 6.7 und 6.8 zu sehen. Die virtuellen Maschinen haben jeweils zwei dedizierte CPU-Kerne um das Wechseln auf andere CPU-Kerne zu vermeiden sowie zu verhindern, dass Sender und Empfänger auf dem gleichen CPU-Kern landen. Bei diesem Versuchsaufbau wurde eine feste Datengröße von 8 Byte pro Paket festgelegt und insgesamt 40 Byte an Daten pro Testlauf übertragen. Jeder Versuch wurde 10 mal wiederholt und die Werte am Ende gemittelt.

Durch Änderungen der Wartezeiten ergibt sich auch eine andere theoretisch maximale Datenübertragungsrate als im Kapitel 6.6 berechnet. Hier ergibt sich bei einer Bandbreite von 200 bit/s und der gegebenen Datengröße von 8 Byte pro Datenpaket eine theoretisch maximale Datenübertragungsrate von 38,1 bit/s. Wie in Abbildung 6.7 zu erkennen wird die theoretisch maximale Datenübertragungsrate nicht erreicht. Dies liegt an den vielen Sendungswiederholungen.

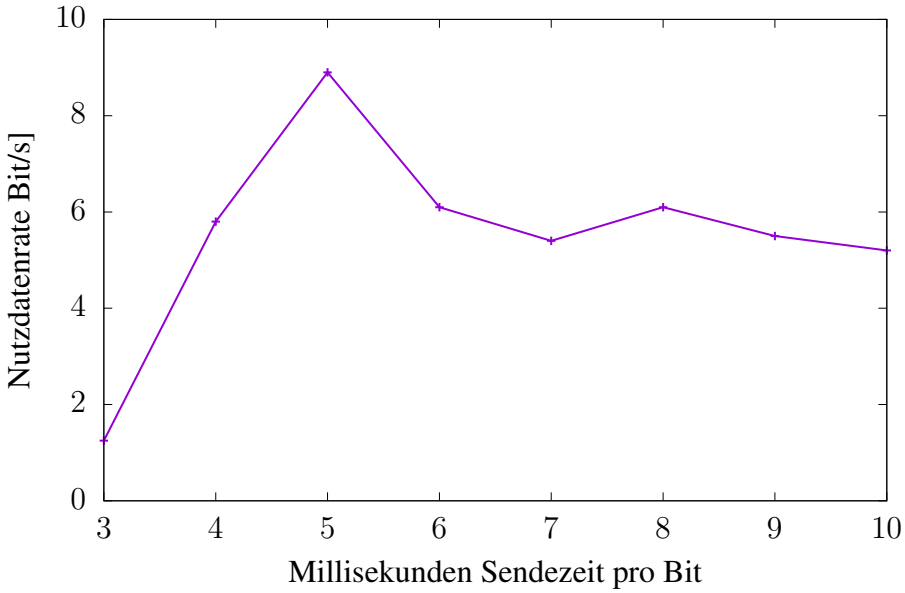


Abbildung 6.7: Dargestellt ist die Nutzdatenrate in Abhängigkeit der Sendedauer pro Bit bei einer Übertragung zwischen virtuellen Maschinen. Der beste Wert wurde bei einer Sendedauer von fünf Millisekunden pro Bit erreicht.

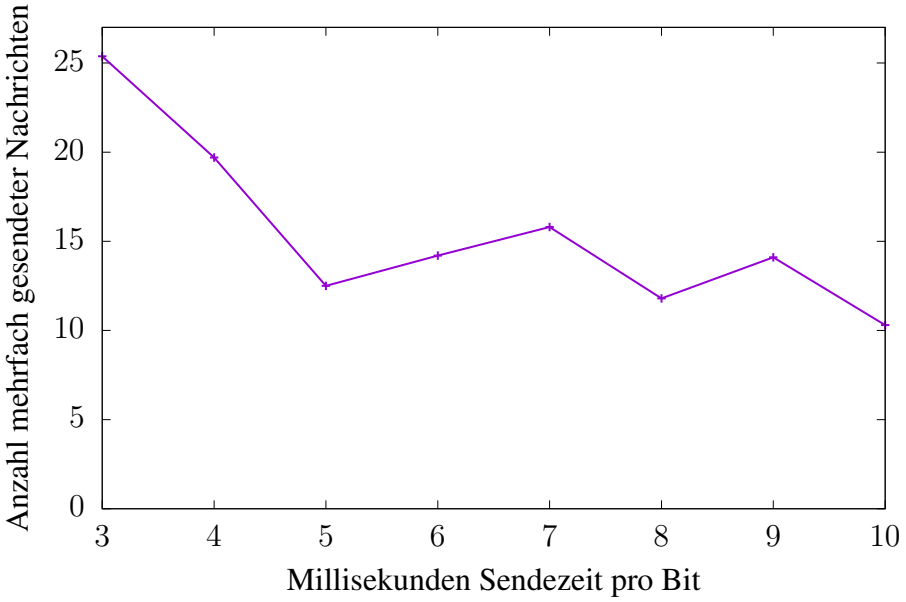


Abbildung 6.8: Dargestellt ist die durchschnittliche Anzahl an Paketen, die mehr als einmal gesendet wurden in Abhängigkeit der Sendezeit pro Bit für den Versuch in Abschnitt 6.7.

Kapitel 7

Diskussion

Der in dieser Arbeit beschriebene Covert Channel ist der erste frequenzbasierte Covert Channel, welcher mithilfe von Turbo-Boost Informationen überträgt. Der Ansatz, Turbo-Boost zu verwenden, bietet einige Vorteile gegenüber bisherigen frequenzbasierten Covert Channel. So können Sender und Empfänger unabhängig der Mithilfe des Betriebssystems Informationen übertragen und dies auch auf aktuellen Prozessoren. Da der in dieser Arbeit beschrieben Covert Channel Einschränkungen bezüglich des Einsatzszenarios unterliegt, wird zuerst in Kapitel 7.1 diskutiert, wann und ob ein Einsatz praktikabel ist. Anschließend in wird in Kapitel 7.2 dieser Covert Channel mit den bisherigen frequenzbasierten Covert Channel verglichen und aufgezeigt, in welchen Punkten sich der in dieser Arbeit beschriebene Ansatz von bisherigen abhebt und unter welchen Gegebenheiten ein Einsatz sinnvoll ist. In Kapitel 7.3 wird kurz angerissen, ob sich aus dem Covert Channel auch ein Side Channel konstruieren lässt. Abschließend wird in Kapitel 7.4 noch auf die Möglichkeit des Einsatzes dieses Covert Channels auf Prozessoren des Herstellers AMD eingegangen.

7.1 Einsatzfähigkeit

Der in dieser Arbeit beschriebene Covert Channel besitzt einen einfachen Aufbau und lässt sich auch leicht einsetzen. Er besitzt jedoch Einschränkungen, die es schwer machen, diesen in realen Systemen zu verwenden. Eine der Hauptbedingungen ist es, dass die CPU-Auslastung sehr niedrig oder während der Übertragung stabil ist. Es gibt teils sehr große Unterschiede, was die Auslastung eines Rechenzentrums angeht. Dies ist hauptsächlich der Art geschuldet, wie die Server eingesetzt werden. So erreicht beispielsweise die Google-Cloud im Schnitt eine CPU-Auslastung von 40% [17]. Dies liegt daran, dass die Server jeweils dynamisch Aufgaben zugewiesen bekommen und dadurch eine höhere Auslastung er-

reichen als Rechenzentren, in denen Kunden Server für ihre Aufgaben mieten können. Hierbei wird im Tagesdurchschnitt nur eine Auslastung von 6% erreicht [23].

Bei dem hier eingesetzten Prozessor führt eine CPU-Auslastung von 40% dazu, dass die zweite von drei Turbostufe erreicht wird. Startet zusätzlich noch ein Empfängerprozess, so wird ein weiterer CPU-Kern ausgelastet und die letzte Turbostufe ist erreicht. Damit ist dieser Cover-Channel nicht mehr möglich. Die Google-Cloud ist nicht der erwartete Einsatzort, jedoch zeigt dies deutlich, dass der Einsatz dieses Covert Channels nicht überall sinnvoll ist.

Selbst wenn man die Auslastung des Systems außer Acht lässt, ergeben sich weitere Schwierigkeiten. So muss ein isoliertes Programm oder virtuelle Maschine infiziert werden, um an die vertraulichen Daten zu gelangen. Machbar wäre dies durch Kompromittieren von Systemanwendungen, Zusatzprogrammen oder Bibliotheken, wie es beispielsweise Ende 2018 bei dem Paket `EventStream` geschehen ist, in welches Code zum Stehlen von Bitcoins eingefügt wurde [35]. Alternativ kann auch schon in der Entwicklung des Programms Code eingeschleust werden. Dies ist alles mit einem großen Aufwand verbunden und daher nur praktikabel bei einem gezielten und im Voraus geplanten Angriff.

Ein Angreifer, der diesen Aufwand eingeht, um den Covert Channel zu benutzen, wird auch warten, bis das System wenig Last hat und alle nötigen Daten übertragen sind. Ansonsten könnte der Angreifer einen andere Covert Channel wählen, welcher eine höhere Übertragungsrate bietet. So hat Beispielsweise der von Maurice et al. beschriebene Covert Channel, welcher den Cache als Kommunikationskanal benutzt, eine Übertragungsrate von mehr als 45 kB/s [28].

Daraus zeigt sich, dass der hier beschriebene Covert Channel ein sehr eingeschränktes Einsatzgebiet hat und in den meisten Fällen ein anderer Covert Channel mit höherer Übertragungsrate bevorzugt werden würde. Sollten andere Covert Channel auf dem Zielsystem nicht funktionieren, so bietet der hier beschriebene Covert Channel eine gute Alternative. Zudem zeigt er, dass Covert Channel in allen Komponenten vorkommen können und selbst wenn durch diese durch Hardwareänderungen behoben werden, neue entstehen können.

7.2 Vergleich zu bisherigen frequenzbasierten Covert Channel

Bisherige frequenzbasierte Covert Channels (siehe Kapitel 2.1) basieren auf dem Prinzip, dass entweder Sender und Empfänger auf dem gleichen CPU-Kern laufen oder alle CPU-Kerne die gleiche Frequenzdomäne haben. Alternativ können auch Schnittstellen des Betriebssystems benutzt werden, um Informationen über die aktuelle CPU-Frequenz anderer CPU-Kerne zu erhalten. In heutigen Prozessorarchi-

tekturen ist es nicht mehr gegeben, dass alle CPU-Kerne der gleichen Frequenzdomäne angehören und auch auf die Informationen des Betriebssystems kann kein Verlass sein, weil das Betriebssystem diese zur Verhinderung eines Covert Channels manipulieren könnte. Dadurch sind diese frequenzbasierten Covert Channel auf heutigen Prozessorarchitekturen nicht mehr möglich, ganz im Gegensatz zu dem in dieser Arbeit beschriebenen Covert Channel. Dieser funktioniert auf aktueller Hardware.

Auch die Übertragungsrate bisheriger frequenzbasierter Covert Channel war bisher eher gering. So erreicht der von Alagappan et al. beschriebe Covert Channel durch das manuelle Setzen der CPU-Frequenz eine Nutzdatenrate von 325 bit/s [8]. Durch das reine Beeinflussen des CPU-Frequenz Governor und ohne spezielle Rechte erreichten der Covert Channel eine Nutzdatenrate von 20 bit/s ohne Übertragungsfehler. In der von Miedl et al. vorgestellten Arbeit wird ein neuronales Netz benutzt, um die Signal zu dekodieren und über die Auslastung des CPU-Kerns den CPU Frequency Governor dazu zu bringen, die gewünschte CPU-Frequenz einzustellen [30]. Dabei erreichten sie eine Nutzdatenrate von bis zu 20 bit/s. Im Vergleich dazu sind bei dem in dieser Arbeit beschriebenen Covert Channel unter Idealbedingungen bis zu 1000 bit/s möglich. Unter Realbedingungen werden Nutzdatenrate von bis zu 56 bit/s erreicht. Mit Ausnahme des von Alagappan et al. beschriebenen Ansatzes, bei dem manuell die CPU-Frequenz gesetzt wurde, hat der in dieser Arbeit vorgestellte Covert Channel eine höhere Nutzdatenrate als vergleichbare Ansätze.

7.3 Turbo-Boost als Side Channel

Neben der Nutzung der Eigenschaften von Turbo-Boost als Covert Channel ist es auch denkbar, sie als Side-Channel zu verwenden. So kann mithilfe der in dieser Arbeit verwendeten Methode zum Senden von Nachrichten die Laufzeit von Prozessen bestimmt werden, wodurch weitere Rückschüsse auf dessen Aufgabe möglich sind. Bei RSA kann beispielsweise durch Analysen über die Dauer von Webseitenaufrufe Rückschlüsse auf den privaten Schlüssel des Webservers getroffen werden [41]. Denkbar wäre hier, dass dies auch über die CPU-Aktivität möglich ist, wobei die Auflösung von einer Millisekunde, mit der Turbo-Boost maximal in der Lage ist die Laufzeit eines Prozesses zu bestimmen, für eine derartige Analyse voraussichtlich zu gering ist.

7.4 AMD Turbo Core

Nicht nur Intel bietet eine Möglichkeit zur automatischen Übertaktung der Basisfrequenz einer CPU. AMD besitzt mit Turbo Core ebenfalls eine solche Technologie. Diese funktioniert jedoch etwas anders als die von Intel. Solange mehr als die Hälfte alle CPU-Kerne aktiv ist und die Thermal Design Power (TDP) noch nicht erreicht ist, kann auf allen CPU-Kernen die Frequenz gleich erhöht werden. Sind nur die Hälfte oder weniger CPU-Kerne aktiv, so kann die Frequenz noch weiter auf die maximale Turbofrequenz erhöht werden [47]. Nach der aus der Präsentation [47] zu entnehmenden Annahme über die Funktionsweise von Turbo Core, müsste dieser analog zu Turbo-Boost funktionieren. Jedoch besitzt Turbo Core nur zwei Frequenzstufen, im Gegensatz zu Intel, welche mehrere verwendet. Zudem gilt ein CPU-Kern als schlafend, wenn er im C6-State oder höher ist. Daraus ergibt sich, dass der hier beschriebene Covert Channel voraussichtlich auch auf AMD Prozessoren funktioniert.

Kapitel 8

Fazit

Covert Channels sind Kommunikationskanäle, die von keiner Sicherheitsrichtlinie erfasst werden und mit denen Prozesse in der Lage sind versteckt Informationen auszutauschen. In dieser Arbeit wurde ein frequenzbasierter Covert Channel entwickelt, welcher mithilfe der Intel Turbo-Boost Technologie in der Lage ist, Informationen zwischen zwei CPU-Kernen auszutauschen, ohne dabei von Schnittstellen des Betriebssystems abhängig zu sein. Im Vergleich zu älteren frequenzbasierten Covert Channels funktioniert dieser auch auf modernen Prozessoren des Herstellers Intel und erreicht zudem bessere Nutzdatenraten von bis zu 56 bit/s bei einer Kommunikation zwischen zwei CPU-Kernen. Zudem ist es mit diesem Covert Channel möglich mit bis zu 9 bit/s zwischen zwei virtuellen Maschinen zu kommunizieren. Um die auftretenden Übertragungsfehler zu kompensieren, wurde ein Übertragungsprotokoll entwickelt. Dies versucht auftretende Fehler mithilfe eines Error-Correction-Codes und durch Sendungswiederholungen zu korrigieren.

Die Intel Turbo-Boost Technologie ist dabei eine Funktion des Prozessors, welche das automatische Übertakten der Prozessorfrequenz erlaubt. Turbo-Boost wählt anhand unterschiedlicher Parameter die aktuelle Turbofrequenz aus, welche für alle CPU-Kerne gilt, die die Turbofrequenz nutzen wollen. Einer der Parameter ist die Anzahl der aktiven CPU-Kerne, welche in dieser Arbeit verwendet wurde, um Informationen zu übertragen. So kann, durch das Aufwecken und schlafen legen eines CPU-Kerns Einfluss auf die Turbofrequenz genommen werden, wodurch es schlussendlich möglich ist Bits zu übertragen.

Der so entstandene Covert Channel ermöglicht somit auf aktueller Hardware den Datenaustausch zwischen verschiedenen Prozessen mit einer relativ hohen Datenrate. Die effektivste Gegenmaßnahme ist das Limitieren von Turbo-Boost auf nur eine Frequenz, was die Vorteile von Turbo-Boost minimiert und daher eine genaue Abwägung der Risiken und Kosten wichtiger macht. Der Covert Channel benötigt jedoch einen erheblichen Aufwand, um auf einem System eingerichtet zu werden, und wird somit nur für spezielle Nutzer zu einem erwähnenswerten

Risiko. Dennoch sollte diese Möglichkeit eines Covert Channel beim Entwurf eines Systems, welches mit vertraulichen Daten arbeitet, nicht außer Acht gelassen werden.

8.1 Ausblick

Bei der Evaluierung der Implementierung hat sich gezeigt, dass es aufgrund von Fehlern innerhalb der Signalerkennung viele Sendungswiederholungen gibt. Um die Nutzdatenrate weiter zu steigern, sollte in zukünftigen Arbeiten die Signalerkennung verfeinert werden und vielleicht dabei der Ansatz von Miedl et al. [30] in Betracht gezogen werden, hierfür ein neuronales Netz einzusetzen. Zudem kann nicht nur die Signalerkennung verbessert werden, auch der Sender kann durch Verwenden mehrerer CPU-Kerne ein deutlicheres Signal senden, wodurch eine Verbesserung der Nutzdatenrate möglich wäre. Bei der Implementierung wurden zudem Wartezeiten verwendet, die mögliche Fehler durch einen falsch erkannten Nachrichtenanfang und deren Folgefehler verhindern sollten. Diese Wartezeiten lassen sich auf eine Übertragungsrate optimieren und so die Nutzdatenrate noch weiter steigern.

Literaturverzeichnis

[1] 3.13. Isolating CPUs Using tuned-profiles-realtime. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/isolating_cpus_using_tuned-profiles-realtime.

[2] gnuplot homepage. http://www.gnuplot.info/.

[3] KVM. https://www.linux-kvm.org/page/Main_Page.

[4] Power Management States: P-States, C-States, and Package C-States | Intel® Software. https://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states.

[5] RFC 793 - Transmission Control Protocol, September 1981. https://tools.ietf.org/html/rfc793.

[6] usleep(3) - Linux manual page, September 2017. http://man7.org/linux/man-pages/man3/usleep.3.html.

[7] Turbo Boost Technology (TBT) - Intel - WikiChip, January 2018. https://en.wikichip.org/wiki/intel/turbo_boost_technology.

[8] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani. DFS covert channels on multi-core platforms. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, October 2017.

[9] Amos Waterland. stress project page, July 2014. https://people.seas.harvard.edu/~apw/stress/.

[10] Davide B. Bartolini, Philipp Miedl, and Lothar Thiele. On the Capacity of Thermal Covert Channels in Multicores. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 24:1–24:16,

New York, NY, USA, 2016. ACM. http://doi.acm.org/10.1145/2901318.2901322.

[11] Bill Torpey. Measuring Latency in Linux - Confessions of a Wall Street Programmer, February 2014. https://btorpey.github.io/blog/2014/02/18/clock-sources-in-linux/.

[12] Ethan Blanton and Mark Allman. TCP Congestion Control, September 2009. https://tools.ietf.org/html/rfc5681.

[13] Aniello Castiglione, Alfredo De Santis, Ugo Fiore, and Francesco Palmieri. An asynchronous covert channel using spam. *Computers & Mathematics with Applications*, 63(2):437–447, January 2012. http://www.sciencedirect.com/science/article/pii/S0898122111006432.

[14] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova. Evaluation of the Intel® Core™ i7 Turbo Boost feature. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 188–197, October 2009.

[15] J. Cioranesco, H. Ferradi, and D. Naccache. Communicating Covertly through CPU Monitoring. *IEEE Security Privacy*, 11(6):71–73, November 2013.

[16] Gabriele Paoloni. How to Benchmark Code Execution Times on Intel®IA-32 and IA-64 Instruction Set Architectures, September 2010. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf.

[17] P. Garraghan, P. Townend, and J. Xu. An Analysis of the Server Characteristics and Resource Utilization in Google Cloud. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 124–131, March 2013.

[18] Berk Gülmezoğlu, Mehmet Sinan İnci, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. A Faster and More Realistic Flush+Reload Attack on AES. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design*, Lecture Notes in Computer Science, pages 111–126. Springer International Publishing, 2015.

[19] Susan C. Hill, Joseph Jelemensky, and Mark R. Heene. Queued serial peripheral interface for use in a data processing system, March 1989. https://patents.google.com/patent/US4816996/en.

[20] Intel. Intel® Xeon® Silver 4108 Prozessor (11 MB Cache, 1,80 GHz) Produktspezifikationen, 2017. https://ark.intel.com/content/www/de/de/ark/products/123544/intel-xeon-silver-4108-processor-11m-cache-1-80-ghz.html.

[21] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d and 4, May 2018.

[22] Jacob Pan. RAPL (Running Average Power Limit) driver [LWN.net], April 2013. https://lwn.net/Articles/545745/.

[23] James M. Kaplan, William Forrest, and Noah Kindler. Revolutionizing Data Center Energy Efficiency, July 2008. https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf.

[24] L. Ji, H. Liang, Y. Song, and X. Niu. A Normal-Traffic Network Covert Channel. In *2009 International Conference on Computational Intelligence and Security*, volume 1, pages 499–503, December 2009.

[25] Eduard Jorswieck and Anne Wolf. Praktikum Fehlerreduktionssysteme / Codierungstheorie. page 5.

[26] Linux. Linux source code: arch/x86/kernel/apic/apic.c (v5.0.3) - Bootlin, March 2019. https://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/apic/apic.c#L476.

[27] Mark Kettenis. CVS: cvs.openbsd.org: src, June 2018. https://www.mail-archive.com/source-changes@openbsd.org/msg99141.html.

[28] Clementine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, Kay Roemer, and Stefan Mangard. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA, 2017. Internet Society. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/hello-other-side-ssh-over-robust-cache-covert-channels-cloud/.

[29] Mete Balci. A Minimum Complete Tutorial of CPU Power Management, C-states and P-states. https://metebalci.com/blog/a-minimum-complete-tutorial-of-cpu-power-management-c-states-and-p-states/.

[30] Philipp Miedl, Xiaoxi He, Matthias Meyer, Davide Basilio Bartolini, and Lothar Thiele. Frequency Scaling as a Security Threat on Multicore Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2018. https://ieeexplore.ieee.org/document/8412555/.

[31] Aleksandra Mileva and Boris Panajotov. Covert channels in TCP/IP protocol stack - Extended version-. *Central European Journal of Computer Science*, 4:45–66, June 2014.

[32] Vicky Miridakis. Wie Software das Rechenzentrum von morgen steuert | Virtualisierung, August 2016. https://www.it-daily.net/it-management/data-center-server-storage/13233-wie-software-das-rechenzentrum-von-morgen-steuert-virtualisierung.

[33] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, pages 35–44, New York, NY, USA, 2002. ACM. http://doi.acm.org/10.1145/514191.514200.

[34] heise online. Merkel: Daten sind Rohstoffe des 21. Jahrhunderts. https://www.heise.de/newsticker/meldung/Merkel-Daten-sind-Rohstoffe-des-21-Jahrhunderts-2867735.html.

[35] heise online. NPM-Paket EventStream mit Schadcode zum Stehlen von Bitcoins infiziert, November 2018. https://www.heise.de/security/meldung/NPM-Paket-EventStream-mit-Schadcode-zum-Stehlen-von-Bitcoins-infiziert-4233171.html.

[36] I. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960. https://epubs.siam.org/doi/abs/10.1137/0108018.

[37] Efraim Rotem. Intel® Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency. page 43, 2015.

[38] Samuel Wan. Intel Will No Longer Disclose Multi-Core Turbo Boost Frequencies | eTeknix, 2017. https://www.eteknix.com/intel-multi-core-turbo-boost/.

[39] Peter Seibt. *Algorithmic Information Theory: Mathematics of Digital Information Processing*. Signals and Communication Technology. Springer-Verlag, Berlin Heidelberg, 2006. https://www.springer.com/de/book/9783540332183.

[40] Servermeile. Intel® Turbo Boost Technology | Servermeile Technet. http://technet.servermeile.com/intel-turbo-boost-technology/.

[41] Intekhab Shaukat. Revisiting Remote Timing Attacks on OpenSSL. page 100, April 2015.

[42] United States Government Department of Defense (DoD). *Trusted Computer System Evaluation Criteria (TCSEC)*. December 1985. https://upload.wikimedia.org/wikipedia/commons/a/aa/Trusted_Computer_System_Evaluation_Criteria_DOD_5200.28-STD.pdf.

[43] Vince Weaver. Reading RAPL energy measurements from Linux, June 2016. http://web.eece.maine.edu/~vweaver/projects/rapl/.

[44] Viritium. Data Coding and Error Checking Techniques, 2014. http://www.virtium.com/wp-content/uploads/2015/11/WP011-0215-01-Data-Coding-and-Error-Checking-Techniques.pdf.

[45] Z. Wang and R. B. Lee. Covert and Side Channels Due to Processor Architecture. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 473–482, December 2006.

[46] Werner Fischer. Processor P-states and C-states - Thomas-Krenn-Wiki, January 2019. https://www.thomas-krenn.com/en/wiki/Processor_P-states_and_C-states.

[47] Sean White. High-performance power-efficient x86-64 server and desktop processors using the core codenamed "Bulldozer". In *2011 IEEE Hot Chips 23 Symposium (HCS)*, pages 1–32, Stanford, CA, USA, August 2011. IEEE. http://ieeexplore.ieee.org/document/7477512/.

[48] Stephen B. Wicker. *Reed-Solomon Codes and Their Applications*. IEEE Press, Piscataway, NJ, USA, 1994.

[49] WikiChip. Xeon Silver 4108 - Intel - WikiChip, April 2019. https://en.wikichip.org/wiki/intel/xeon_silver/4108.

[50] Jingzheng Wu, Liping Ding, Yanjun Wu, Nasro Min-Allah, Samee U. Khan, and Yongji Wang. C2detector: a covert channel detection framework in cloud computing. *Security and Communication Networks*, 7(3):544–557, March 2014. https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.754.

[51] Milan Yadav. A Brief Survey of Current Power Limiting Strategies. December 2017.