Zielstellung und Rahmenbedingungen

Die vorliegende Dokumentation fasst die Ergebnisse der im Praktikum „Angriffe auf AMQP-Messagebroker“ erarbeiteten Angriffe zusammen. Das Praktikum war Bestandteil des kooperativ gelehrten Modules „Betriebliche Informationssysteme“ an der HTWK Leipzig und Uni Leipzig. Ziel des Praktikums war die Ermittlung von potenziellen Angriffsvektoren auf AMQP-Messagebroker - im speziellen RabbitMQ -, die Entwicklung der dafür benötigten Software, die Durchführung der Angriffe und Auswertung der Auswirkungen auf den Server selbst sowie andere unbeteiligte Clients.

Die Implementierung der Angriffe erfolgte in der Programmiersprache Java und nutzt dabei die offizielle RabbitMQ Java-Bibliothek. Mit veränderten Parametern richtet sich die Suche gezielt nach Faktoren die das System negativ beeinflussen und so die Verfügbarkeit stören, fokussiert auf „Denial of Service“. DoS kann dabei auf verschiedene Ressourcen bezogen sein (wie CPU, Arbeitsspeicher, Netzwerkbandbreite etc.).

Um die Begrenzungen ausfindig zu machen, gehört es zu Beginn zu unsere Aufgabe angemessene Werkzeuge zur Beobachtung der Ressourcen zu finden. Nur dann ist es möglich die Angriffe zu bewerten und ihre tiefere Auswirkung zu untersuchen.

Ferner sollen Vorschläge zur Schadensbegrenzung gegeben werden. Dazu gehört die Angabe auf welcher Ebene (Netzwerkebene, Protokollebene etc.) sich die Gefahren beseitigen lassen.

Verwendete bzw. Erstellte Programme

RabbitMQ bietet bereits ein Management-Oberfläche, mit denen sich die einige Kennzahlen zur Leistungsfähigkeit des Server messen lassen. Weiterhin enthält der Java-Client unterhalb von *rabbitmq-client-tests.jar* bereits eine Vielzahl an zahlreichen kleine Beispielprogramme für das Benchmarking der Funktionalität des eigenen Servers.

VirtualBox: Der von uns verwendete Server wurde durch die Virtualisierungslösung VirtualBox realisiert. Neben der einfachen Installation neuer Gast-Systeme besteht die Möglichkeit Sicherheitspunkte zu erstellen. Bei einem Ausfall des System kann so der ursprüngliche Sicherungspunkt wiederhergestellt werden. Weiterhin kann die Bereitstellung der Ressourcen und der Anbindung an das Netzwerk beliebig eingestellt werden.

Glances: Das in Python geschriebene Glances ist ein System-Monitoring-Tool, welches zahlreiche Informationen auf engstem Raum präsentiert. Im einfachsten Fall werden die gesammelten System-Informationen auf der Konsole ausgeben. Im vorliegenden Fall wird Glances für den Export der Messwerte in eine CSV-Datei verwendet. Ferner ermöglicht Glances die Bereitstellung eines integrierten Web-Interfaces zur Fernüberwachung der Umgebung. Zu den gesammelten Informationen gehören unter anderem: CPU-Auslastung, Prozesse, Speicherverbrauch (HDD, RAM), Netzwerk- und Disk-IO sowie weitere Informationen zum System.

HTML Performance Tools: Die HTML Performance Tools ermöglichen die Definition einer Vielzahl von Anwendungsszenarios, welche systematisch abgearbeitet werden und deren gemessene Kennzahlen in einer JSON-Datei gespeichert werden. Die Besonderheit ist die integrierte grafische Visualisierung der Ergebnisse im Webbrowser. Die Tools basieren auf dem im Java-Client enthaltenen **Perftest**.

AMQPstress: Um die Auswirkungen der gewählten Angriffsvektoren testen zu können, wurde für jeden Angriff ein eigener Client implementiert. Grundlage bildet die Java-Library von RabbitMQ, die geeignete Funktionen zur Ansprache des Servers bereitstellt. Alle erstellten Clients lassen sich über bestimmte Parameter aufrufen und mit weiteren Einstellungen versehen. Welche Einstellungen zutreffen, lässt sich anhand dieses Dokuments erschließen. Auch zeigt sich, welche Parameter für einen erfolgreichen Angriff nötig sind. Hierzu gehört unter anderem die Anzahl an Producer und Consumer oder auch die gewählte Nachrichtengröße.

HTML-Reporting: Um eine anpassbare Visualisierung der Messwerte zu ermöglichen, wurde die Visualisierungkomponente der **HTML Performance Tools** angepasst und ausgebaut. Sie ermöglicht nun eine Visualisierung und Reporting der von Glances und von Perftest bereitgestellten Informationen.

Testumgebung

Als Betriebssystem für den RabbitMQ-Servers wurde ein *Ubuntu Server 14.04.2 LTS* mit Kernel 3.16.0-30 verwendet. Das Testsysten wurde dabei als Virtuelle Maschine (VM) in der Virtualisierungslösung „Virtualbox“ der Firma Oracle betrieben, um eine einfache Skalierung der Hardware, eine einfache Portabilität des Testsystem und einen reproduzierbaren Systemzustand, über die integrierte Snapshot-Funktion, bereitzustellen. Für die Konfiguration wurden die in Tabelle 1 dargestellten Einstellungen gewählt.

Host	rabbitmqserver (192.168.178.153)
Anz. CPU	2
RAM	2048 MB
Grafikspeicher	12 MB
HDD	8,00 GB
Netzwerk	1GB über Bridging durch lokales Netzwerk-Interface

Tabelle 1: Konfiguration der Test-VM

Für die reproduzierbare Einrichtung der Testumgebung wurde ein BASH-Skript erstellt, welches sich im Repository des zugehörigen GitHub-Projektes befindet. Es setzt eine frische Installation des Ubuntu Server voraus - ohne eine Vorauswahl an Softwarepacketen (z. B. LAMP). Vor der Ausführung des Skriptes sollte ein Snapshot erstellt werden. Anschließend kann es in der VM über:

```
$> wget https://raw.githubusercontent.com/philippsied/amqp-stress-test/
master/utilities/setupTestEnv.sh
```

aus dem Git-Repository heruntergeladen werden. Da das Skript für die Einstellung des Netzwerkinterfaces Root-Rechte benötigt (Sudo-Rechte genügen nicht), muss der Root-User zuvor aktivert werden. Abschließend kann das Skript mit Sudo-Rechten ausgeführt werden:

```
$> sudo passwd root
$> sudo bash setupTestEnv.sh
```

Nach Abschluss der Einrichtung erfolgt eine übersichtliche Ausgabe aller Informationen, einschließlich der RabbitMQ-Benutzerdaten und Adressen.

Beschreibung des Benchmarking

Für die Durchführung des Benchmarking wurde stets der nachfolgend beschriebene Ablauf verwendet. Hierbei kamen drei physisch unabhängige Computer als Messpunkte zum Einsatz: der RabbitMQ-Server, der Client von dem der Angriff ausgeht und der Client zur Messung des Anwendungsszenarios. Dadurch soll eine gegenseitige Beeinflussung bereits im Vorfeld ausgeschlossen werden. Da die Messungen stets für ca. 10min ausgelegt waren, wurden sämtliche Stromsparmechanismen für die ersten 10min deaktiviert.

Zu Beginn muss die VM des RabbitMQ-Servers gestartet werden. Vor jeder Messung wurde die VM dabei auf einen Snapshot zurückgesetzt, bei der sich das System einem Zustand befindet, den es 5 minuten nach dem Hochfahren der VM besitzt (Uptime ca. 5min). Für die Beobachtung des Ressourcenverbrauches auf dem RabbitMQ-Server wurde das Kommandozeilenprogramm *glances* verwendet.
Der Programmaufruf für jeden Test lautete wie folgt:

```
$> sudo glances -t 5 --disable-process --export-csv measure.csv
```

Er veranlasst, die Ansicht alle 5 Sekunden zu erneuern und die Anzeige der aktuell laufenden Prozesse abzuschalten, um den Ressourcenverbrauch durch *glances* selbst zu reduzieren. Die Messwerte werden dabei in die *measure.csv*-Datei exportiert, welche später zur Visualisierung und Auswertung verwendet wurde.

Anschließend wurde die Messung des Anwendungsszenarios gestartet. Dazu wurde das Skript **startBenchmark.sh**[1] verwendet.
Der Aufruf lautete wie folgt:

```
$> bash startBenchmark.sh "measure.json" \
                                testc:testp@192.168.178.153:5672/%2f
```

Als letztes wurde der Angriff gestartet. Dafür kamen je nach Angriff unterschiedliche Parameter zum Einsatz. Die Angriffe wurde über separate „runconfigs“[2] aus eclipse heraus gestartet.

Die Messung des Anwendungsszenario ist auf 10min festgelegt.[3] Sobald die Messung des Anwendungsszenarios abgeschlossen ist, kann glances über die Eingabe von „q“ beendet werden, der Angriff unterbrochen werden und die Ergebnisse in der *measure.csv*-Datei bzw. *measure.json*-Datei von den Computern aggregiert werden.

Abschließend gilt es die Ergebnisse grafisch aufzubereiten und auszuwerten. Dafür müssen die *measure.csv* und die *measure.json* Dateien z. B. unterhalb des reporting Ordners kopiert

[1] https://github.com/philippsied/amqp-stress-test/tree/master/perfBenchmark/runscenario
[2] https://github.com/philippsied/amqp-stress-test/tree/master/perfBenchmark/runconfigs
[3] https://github.com/philippsied/amqp-stress-test/tree/master/perfBenchmark/runscenario/publish-consume-spec.template

werden[4] und mittels des **transformToJson.sh** BASH-Skriptes in eine „measure.json“ transformiert werden:

```
$> bash transformToJson.sh "DoS" measure.csv measure.json
```

Über die **Results.html**-Datei können die Ergebnisse nun betrachtet und ausgewertet werden. Sofern die eingestellten Skalierungen für die Achsen nicht genügen, können diese über die div-Attribute in der HTML-Datei angepasst werden.

[4] https://github.com/philippsied/amqp-stress-test/tree/master/perfBenchmark/reporting

Beschreibung der Anwendungsszenarien

S1	**Leerlauf**
Beschreibung	Der RabbitMQ-Server befindet sich im Leerlauf, d. h. ohne irgendeine Form von Last.
Parameter	Keine
Aktion	`Keine`
Ressourcenbedarf	CPU: 0.5% RAM: 120MB (Verwendet) HDD: 785MB (Frei) NET: RCX 3Kbit/s TRX 197Kbit/s
Anmerkungen	Keine

S2	**Anwendungsszenario**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten, die von einem Consumer kontinuierlich korrekt entnommen werden. Es finden keine weiteren Anfragen bzw. Zugriffe auf den RabbitMQ-Server statt.
Parameter	Messdauer: 600s (10min) 1 Consumer 1 Producer Nachrichtengröße: 1024 Byte Max. 50 Nachrichten/s (Producer)
Aktion	`bash startBenchmark.sh testc:testp@192.168.178.153:5672/%2f`
Ressourcenbedarf	CPU: 2% RAM: 127MB (Verwendet) HDD: 785MB (Frei) NET: RCX 299Kbit/s TRX 370Kbit/s
Anmerkungen	Die Sender/Empfangsrate sowie die Latenz sind wie erwartet weitgehend konstant. Die Latenz liegt durchschnittlich bei 1.3 ms. (Siehe Abb. 1, 2, 3)

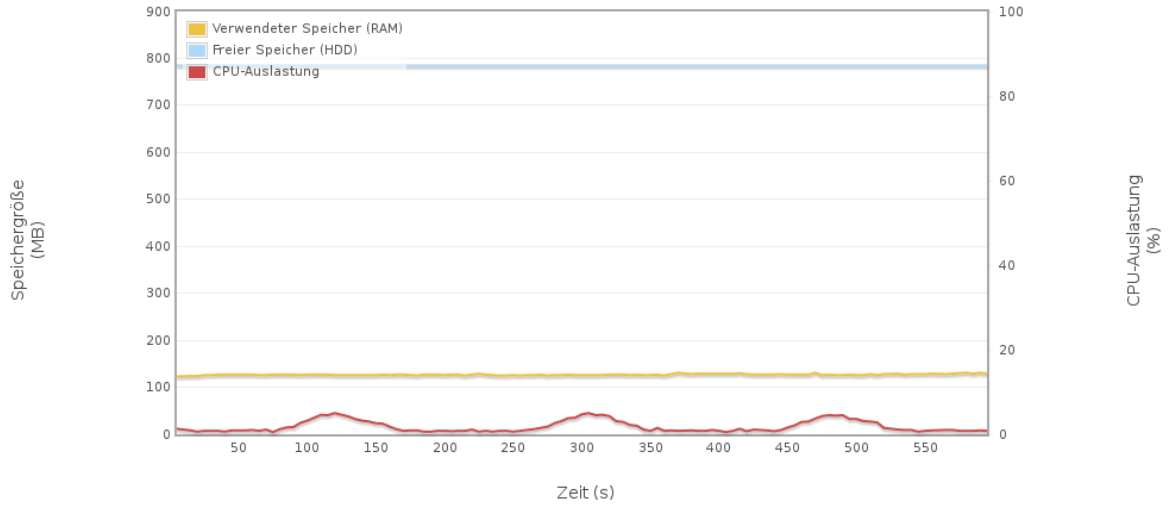

Abbildung 1: Ohne Angriff - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

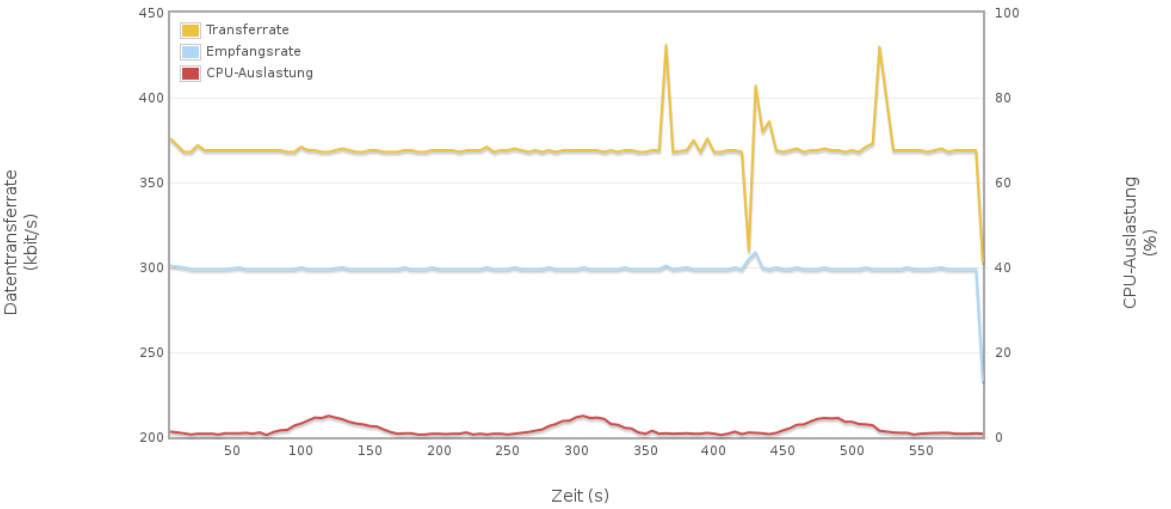

Abbildung 2: Ohne Angriff - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

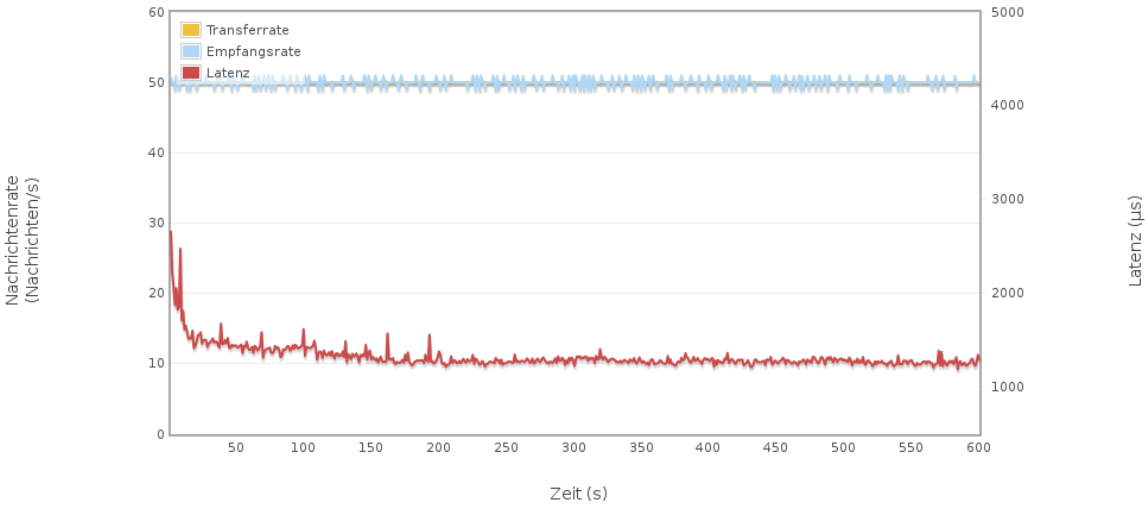

Abbildung 3: Ohne Angriff - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Beschreibung der Angriffe

A1	**Referenzangriff - Kooperativer Client**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten festgelegter Größe mit zufälligem Inhalt, die über einen „fanout“-Exchange an 5 Consumer verteilt werden. Die Consumer quittieren dem RabbitMQ-Server dabei den Erhalt der Nachricht. Dieser Angriff dient zur Ermittlung einer Leistungsreferenz.
Testparameter	-dm ACK (Aufrufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 10240 (Nachrichtengröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard) Prefetching: Max. 512MB (Standard)
Befehlszeile	`Amqpstress -dm ACK -c 5 -i 10 -ms 10240 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A2	**Ignorieren von Nachrichten**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten festgelegter Größe mit zufälligem Inhalt, die von einem oder mehreren Consumern empfangen, aber nicht quittiert werden. Der RabbitMQ-Server ist somit gezwungen, die Nachrichten in der Queue zwischenzuspeichern.
Testparameter	-dm NO (Aufrufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 10240 (Nachrichtengröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard) Prefetching: Max. 512MB (Standard)
Befehlszeile	`Amqpstress -dm NO -c 5 -i 10 -ms 10240 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A3	**Sofortiges Abweisen von Nachrichten**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten festgelegter Größe mit zufälligem Inhalt, die von einem oder mehreren Consumern empfangen, aber sofort abgewiesen (basic.Reject) werden. Der RabbitMQ-Server ist somit gezwungen, die Nachrichten in der Queue zwischenzuspeichern und erneut an den Consumer zu senden.
Testparameter	-dm REJECT (Aufrufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 10240 (Nachrichtengröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard) Prefetching: Max. 512MB (Standard)
Befehlszeile	`Amqpstress -dm REJECT -c 5 -i 10 -ms 10240 -mp -u amqp://testc:testp@192.168.178.153:5672/%2f`

A4	**Gebündeltes Abweisen von Nachrichten**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten festgelegter Größe mit zufälligem Inhalt, die von einem oder mehreren Consumer empfangen und zunächst ignoriert werden. Erreicht die Anzahl der ignorierten Nachrichten einen definierten Schwellwertes, werden diese gebündelt abgewiesen (basic.NACK). Dadurch ist der RabbitMQ-Server gezwungen alle Nachrichten zwischenzuspeichern und stoßweise alle Nachrichten bis zu der aktuellen Sequenznummer erneut zuzustellen. Die Implementierung von basic.NACK stellt eine Besonderheit von RabbitMQ dar, welche nicht Bestandteil von AMQP-0-9-1 ist.
Testparameter	-dm NACK (Aufrufparameter für Angriff) Schwellwert: 1000 Nachrichten (Standard) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 10240 (Nachrichtengröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard) Prefetching: Max. 512MB (Standard)
Befehlszeile	`Amqpstress -dm NACK -c 5 -i 10 -ms 10240 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A5	Queue-Churning
Beschreibung	Ein Client erzeugt bis zu einem Schwellwert Queues und befüllt Sie optional mit einer zufälligen Nachricht gegebener Größe. Wenn die Anzahl der erzeugten Queues einen definierten Schwellwert erreicht, werden alle Queues ohne zu Warten gelöscht und der Zyklus beginnt erneut. Während bereits neue Queues angelegt und ggf. befüllt werden, muss der RabbitMQ-Server die Überreste der Queues im RAM bzw. der Festplatte bereinigen.
Testparameter	-dq NO (Aufrufparameter für Angriff -Ohne Nachrichten) -i 1 (1ms Pause zwischen Queue-Erzeugung) -pc 100000 (Schwellwert: 100000 Queues) -u <uri> (URI für Verbindung mit Server) 1 Client - Consumer und Producer zugleich (Standard) Nicht-Persistente Nachrichten/Queues (Standard)
Befehlszeile	`Amqpstress -dq NO -i 1 -pc 100000 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A6	Nachrichten mit großem Header
Beschreibung	RabbitMQ bietet die Möglichkeit im Header einer Nachricht bestimmte Parameter für die Weiterleitung zu deklarieren. Dieser Test beschäftigt sich mit den Auswirkung auf den Server bzw. die Clients, die übergroße Header verursachen. Dies wird dadurch begünstigt, dass der RabbitMQ-Server gezwungen ist alle Weiterleitungsoptionen zu Prüfen, auch wenn diese keinem Ziel entsprechen.
Testparameter	-lh (Aufrufparameter für Angriff) -i 10 (10ms Pause zwischen 2 Anfragen) -ms 10000 (Nachrichtengröße: 10000 Byte) -hs 2500 (Headergröße - Anzahl Einträge in Map) -u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -lh -i 10 -ms 10000 -hs 2500 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A7	Channel-Flooding
Beschreibung	Neben einzelnen Connections können Clients zu einem RabbitMQ-Server auch mehrere Channel aufbauen. Hier stellt sich die Frage, wie der RabbitMQ-Server mit einer Vielzahl von Channeln zurecht kommt. Auf Basis einer einzelnen Connection wird das System so ausgelastet und beobachtet.
Testparameter	-mc (Aufrufparameter für Angriff) -p 100 (100 Producer) -c 10 (10 Consumer) -ms 10000 (Nachrichtengröße: 10000 Byte) -u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -mc -p 100 -c 10 ms 10000 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A8	Ausbleiben von Commits im Transaktionsmodus
Beschreibung	Im Transaktionsmodus ist es möglich mehrere Nachrichten als Folge zu übertragen, die, analog zu Transaktionen einer Datenbanken, als Einheit betrachtet werden. Nur durch die Commit()-Funktion wird der neue Zustand angenommen und die Nachrichten im System zur Verfügung gestellt. Ferner kann mit der Rollback()-Funktion der ursprüngliche Zustand wiederhergestellt werden. Zu untersuchen wäre die Situation, wenn permanent Nachrichten ohne commit gesendet werden - der Server muss die Nachrichten dann im Speicher belassen.
Testparameter	-tx (Aufrufparameter für Angriff) -p 5 (5 Producer) -ms 10240 (Nachrichtengröße: 10KB) -mct 100000 (Nachrichtenanzahl pro Producer: 100000) -u <uri> (URI für Verbindung mit Server) Commit auslassen (Standard)
Befehlszeile	`Amqpstress -tx -p 5 ms 10240 -mct 100000 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A9	**Handshake-Trickle**
Beschreibung	Der Handshake zum Aufbau einer Connection zwischen Client und RabbitMQ-Server erfolgt in 9 Schritten. Ziel des Angriffes ist es, gezielt Pausen in den Protokollablauf einzubauen und die Auswirkungen auf Server sowie andere Clients zu beobachten.
Testparameter	-sh (Aufrufparameter für Angriff) -i 0 (Keine Pause zwischen 2 Handshakes) -cl 20 (20 Clients) -u <uri> (URI für Verbindung mit Server) 1s Pause in jedem Schritt (Standard)
Befehlszeile	`Amqpstress -sh -i 0 -cl 20 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A10	**Heartbeat-Flooding**
Beschreibung	Damit ein RabbitMQ-Server clientseitig geschlossene TCP-Verbindungen erkennen kann (sowie vice versa), werden periodisch Heartbeats an die Clients gesendet. Dabei sendet der Server bei der Hälfte des eingestellten Timeouts einen Heartbeat an den Client. Sofern er keine Rückmeldung erhält sendet er einen zweiten Heartbeat bei erreichen des Timeouts und schließt ggf. die TCP-Verbindung. Ziel diese Angriffes ist Reduzierung des Heartbeat-Timeouts auf ein Minimum, um anschließend die Auswirkungen auf Server und Clients zu beobachten.
Testparameter	-hb (Aufrufparameter für Angriff) -i 1 (1s Heartbeat Timeout) -cl 20 (20 Clients) -u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -hb -i 1 -cl 20-u amqp://testc:testp@192.168.178.153:5672/%2f`

A11	**TCP-Connection-Dropping**
Beschreibung	Im Vergleich mit den restlichen Angriffen, setzt dieser Angriff weiter unten im Netzwerkstack an und versucht direkt auf TCP-Ebene die Connections zu schließen. Hierbei wird der TCP-Socket zu einer Connection ermittelt und stillschweigend geschlossen, sodass der RabbitMQ-Server erst nach Erreichen des Heartbeat-Timeout die Connection schließen kann. Die Connections werden in hoher Frequenz aufgebaut geschlossen, sodass der RabbitMQ-Server durch die hängenden Connections belastet ist. Anmerkung: Beim Schließen des TCP-Sockets wird ein RST-Packet an den Server gesendet, was sich mit Java-Bordmitteln nicht unterbinden lässt, deshalb ist die Einrichtung einer Firewall-Richtlinie zum Blocken der RST-Packeten notwendig (dropRST.sh).
Testparameter	-dc (Aufrufparameter für Angriff) -i 100 (100ms Pause zwischen Connection Erzeugung) -u <uri> (URI für Verbindung mit Server)

Befehlszeile	`Amqpstress -dc -i 100 -u amqp://`
	`testc:testp@192.168.178.153:5672/%2f`
	`sudo bash dropRST.sh -i eth0 192.168.178.153 5672`

Auswirkungen der Angriffe

Angriff	**Referenzangriff - Kooperativer Client**
Messwerte	CPU: 13% RAM: 134MB (Verwendet) HDD: 785MB (Frei) NET: RCX 5Mbit/s TRX 42.8Mbit/s
Beobachtungen	Es ist keine Beeinträchtigung erkennbar. Die Nachrichtenraten liegen jeweils konstant bei 50 Nachrichten/s und die Latenz liegt bei durchschnittlich 1.3 ms. (Siehe Abb. 6).
Anmerkungen	Keine

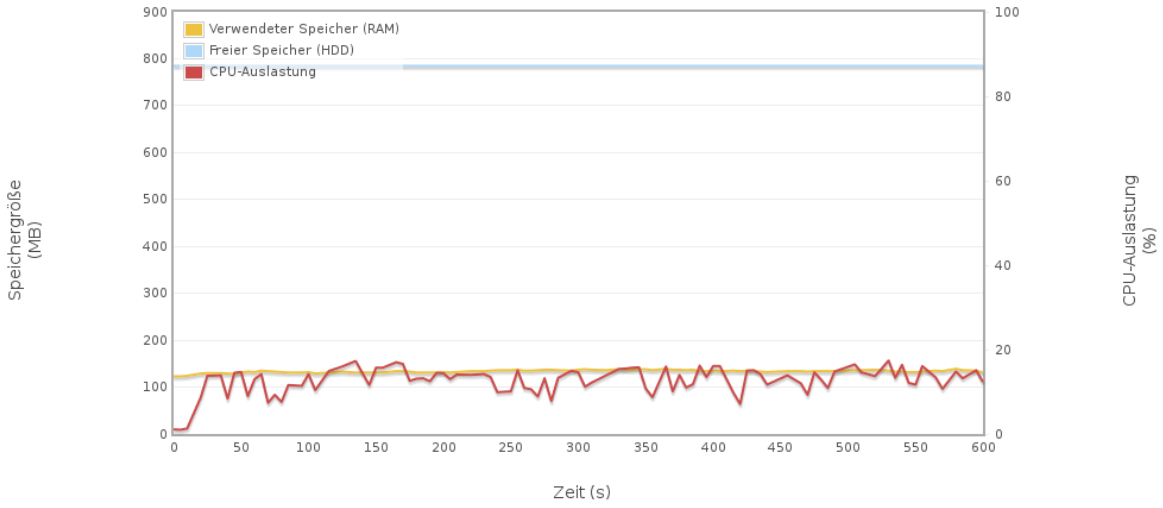

Abbildung 4: Kooperativer Client - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

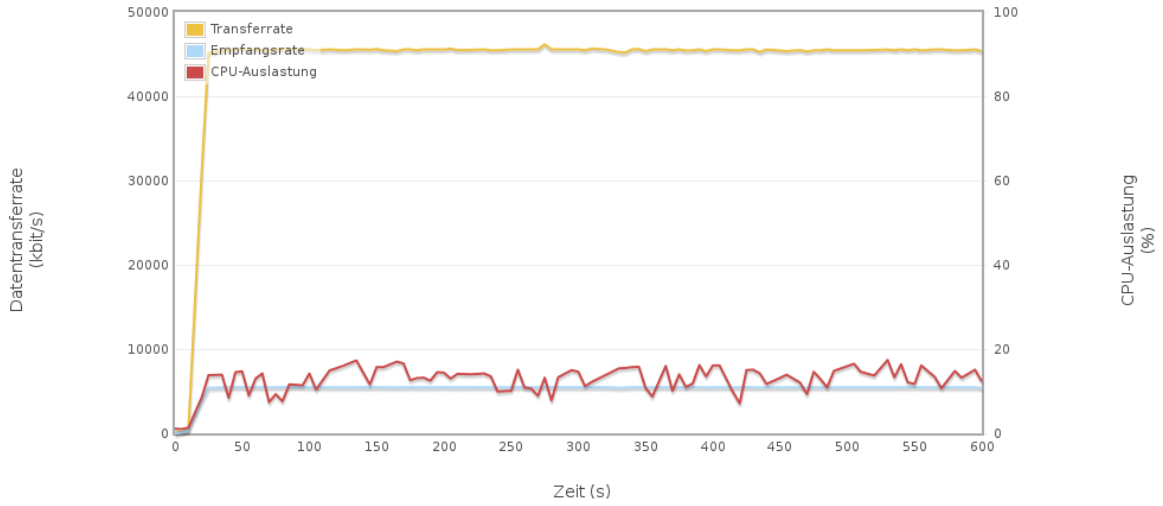

Abbildung 5: Kooperativer Client - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

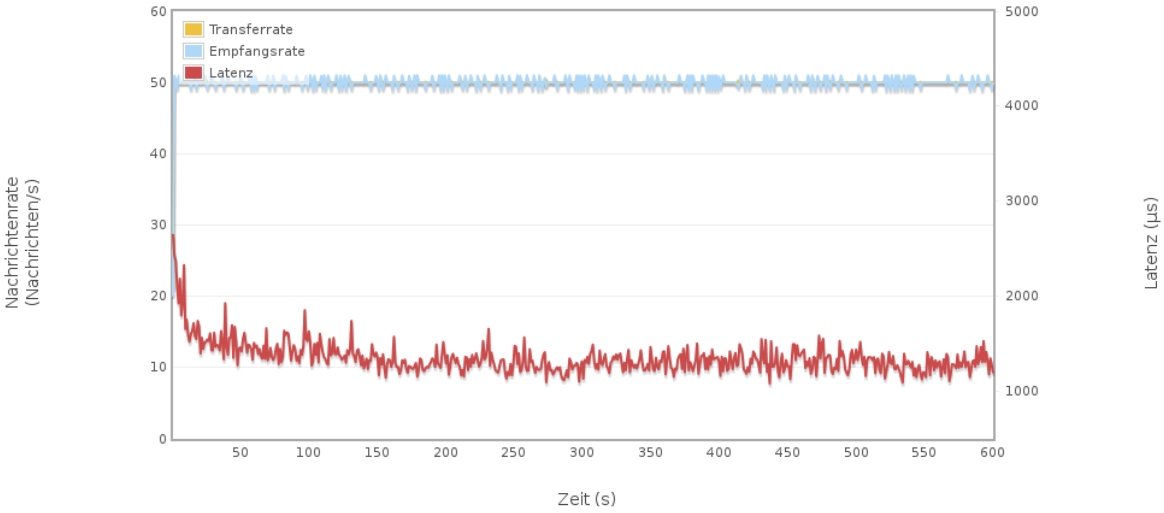

Abbildung 6: Kooperativer Client - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Ignorieren von Nachrichten**
Messwerte	CPU: 9% RAM: 362MB (Verwendet) HDD: 593MB (Frei) NET: RCX 3.8Mbit/s TRX 33.8Mbit/s
Beobachtungen	Die Nachrichten werden bis zur Hälfte des eingestellten Limits (hier 500MB) im RAM gehalten, bei überschreiten der Grenze lagert der RabbitMQ-Server die Nachrichten im Hintergrund auf die Festplatte aus. Hierbei entsteht eine CPU-Lastspitze, welche in Abb. 7 deutlich bei ca. 260s zu sehen ist und sich direkt auf die Latenz im Anwendungsszenario auswirkt (Siehe Abb. 9). Durch die Freigabe des RAM-Speichers entsteht ein Sägezahnmuster. Dieser Vorgang wiederholt sich bis der standardmäßig gesetzte Schwellwert erreicht ist und der Server alle Verbindungen schließt und in diesem Zustand verweilt. Der Ressourcenverbrauch sinkt auf den Leerlauf-Zustand. Davor lag die Latenz bei durchschnittlich ca. 1.4ms. Die Übertragungsraten (Daten/Nachrichten) decken sich weitgehend mit denen des Referenzszenarios.
Anmerkungen	In der aktuellen Implementierung des Angriffes hat der Client ebenfalls einen hohen Bedarf an Arbeitsspeicher, da die Client-Bibliothek die empfangene Nachricht für jeden Consumer separat zwischenspeichert. Dies lässt sich durch den Umfang des Prefetching einstellen.

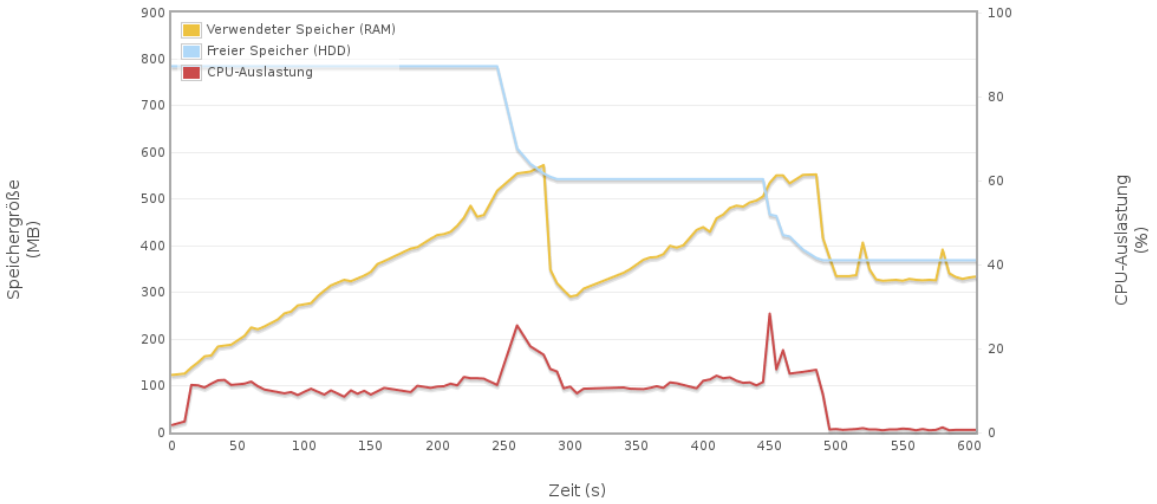

Abbildung 7: Ignorieren von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

Abbildung 8: Ignorieren von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

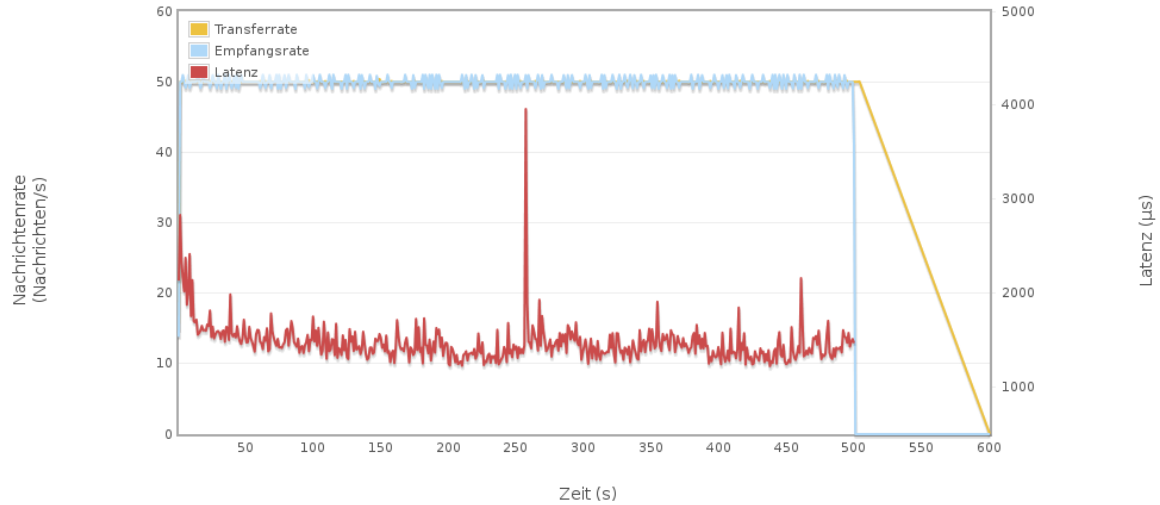

Abbildung 9: Ignorieren von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Sofortiges Abweisen von Nachrichten**
Messwerte	CPU: 24% RAM: 467MB (Verwendet) HDD: 603MB (Frei) NET: RCX 4.2Mbit/s TRX 28.2Mbit/s
Beobachtungen	Der Beginn ähnelt den Beobachtungen beim Ignorieren der Nachrichten. So wird zunächt der RAM gefüllt bis die eingestellte Grenze erreicht wird, um die Daten auf die Festplatte auszulagern (Siehe Abb. 10, 11). Die CPU-Auslastung ist hierbei jedoch höher. Zudem sind die Übertragungsraten im Vergleich zum Referenzszenario „unruhiger“. Die Latenz ist insgesamt leicht erhöht und steigt nach dem Beginn des Auslagerns auf die Festplatte weiter an. Sobald der verfügbare Festplattenspeicher erschöpft ist, bricht der RabbitMQ-Server die Verbindungen ab (Siehe Abb. 12). Weshalb der Speicherbedarf über die Grenze hinauswächst ist unklar (Siehe Abb. 10, ca. 510s). Die Autoren vermuten, dass für abgewiesen Nachrichten andere Puffer verwendet werden, welche nicht auf die Festplatte ausgelagert werden können bzw. dürfen.
Anmerkungen	In der aktuellen Implementierung des Angriffes hat der Client ebenfalls einen hohen Bedarf an Arbeitsspeicher, da die Client-Bibliothek die empfangene Nachricht für jeden Consumer separat zwischenspeichert. Dies lässt sich durch den Umfang des Prefetching einstellen.

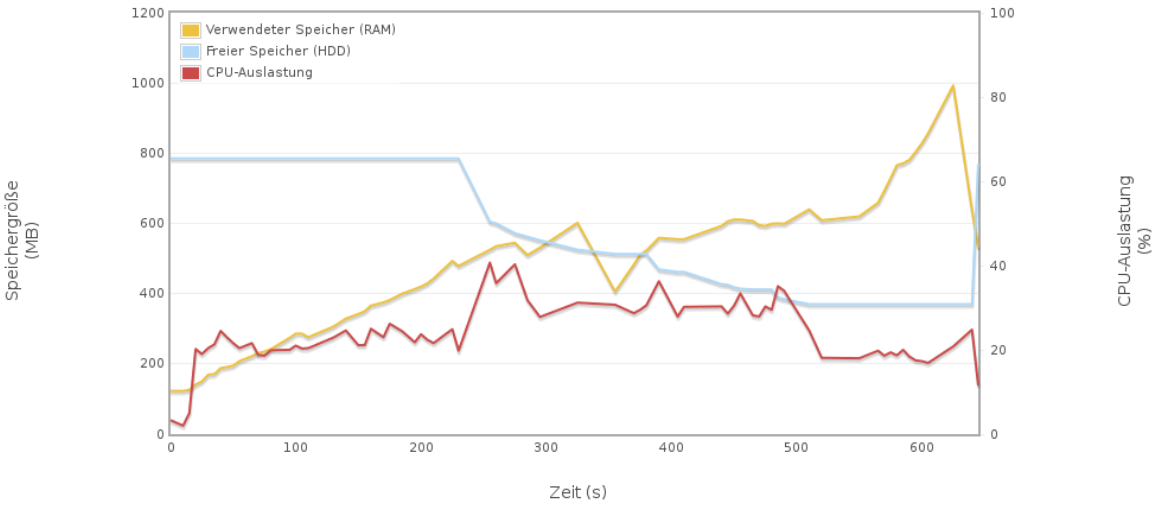

Abbildung 10: Sofortiges Abweisen von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

Abbildung 11: Sofortiges Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

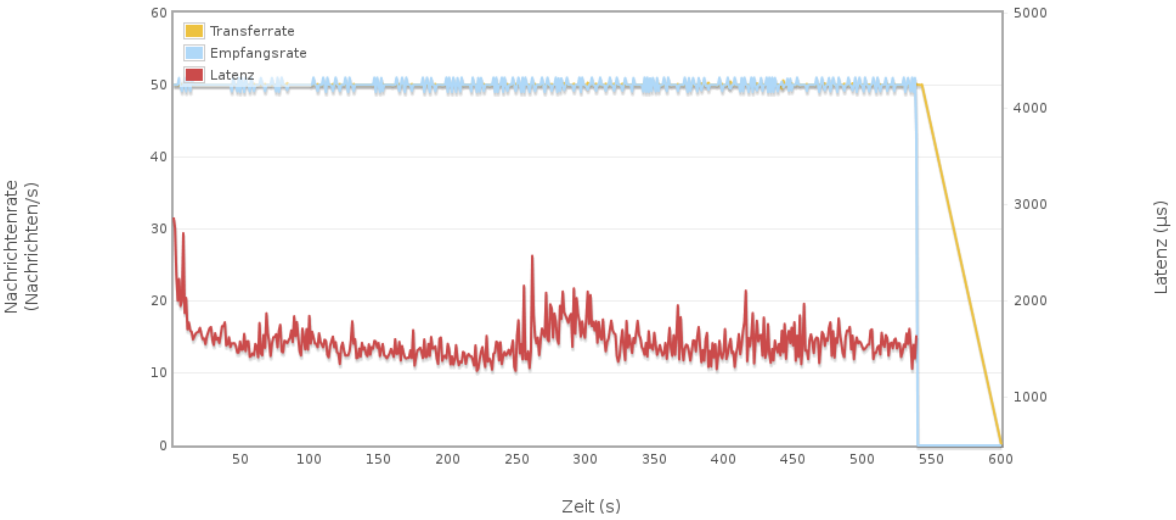

Abbildung 12: Sofortiges Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Gebündeltes Abweisen von Nachrichten**
Messwerte	CPU: 15% RAM: 534MB (Verwendet) HDD: 572MB (Frei) NET: RCX 4.2Mbit/s TRX 81.8Mbit/s
Beobachtungen	Wie in den vorangegangen Scenarien, werden die Nachrichten bis zum Schwellwert im RAM gehalten, wobei sie bei Erreichen des Schwellwertes auf die Festplatte ausgelagert werden. Zeitgleich werden kontinuierlich Nachrichten vom RabbitMQ-Server an die Consumer gesendet, die diese zunächst Ignorieren. Das gebündelte Abweisen seites der Consumer kann der Abb. 14 und den Peaks im Verlauf der Latenz in Abb. 15 sehr gut entnommen werden. Nachdem der Server keinen freien Festplattenspeicher mehr zur Verfügung hat, schließt er die Verbindungen, wobei der Bedarf an RAM weiter steigt. Weshalb der Speicherbedarf über die Grenze hinauswächst ist unklar (Siehe Abb. 13, ca. 510s). Die Autoren vermuten, dass für abgewiesen Nachrichten andere Puffer verwendet werden, welche nicht auf die Festplatte ausgelagert werden können bzw. dürfen.
Anmerkungen	In der aktuellen Implementierung des Angriffes hat der Client ebenfalls einen hohen Bedarf an Arbeitsspeicher, da die Client-Bibliothek die empfangene Nachricht für jeden Consumer separat zwischenspeichert. Dies lässt sich durch den Umfang des Prefetching einstellen.

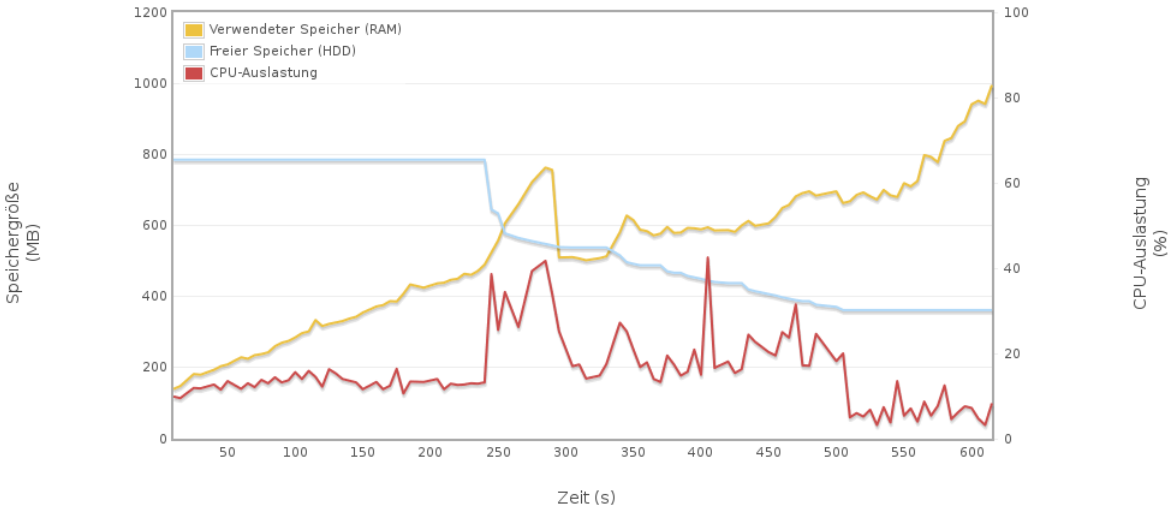

Abbildung 13: Gebündeltes Abweisen von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

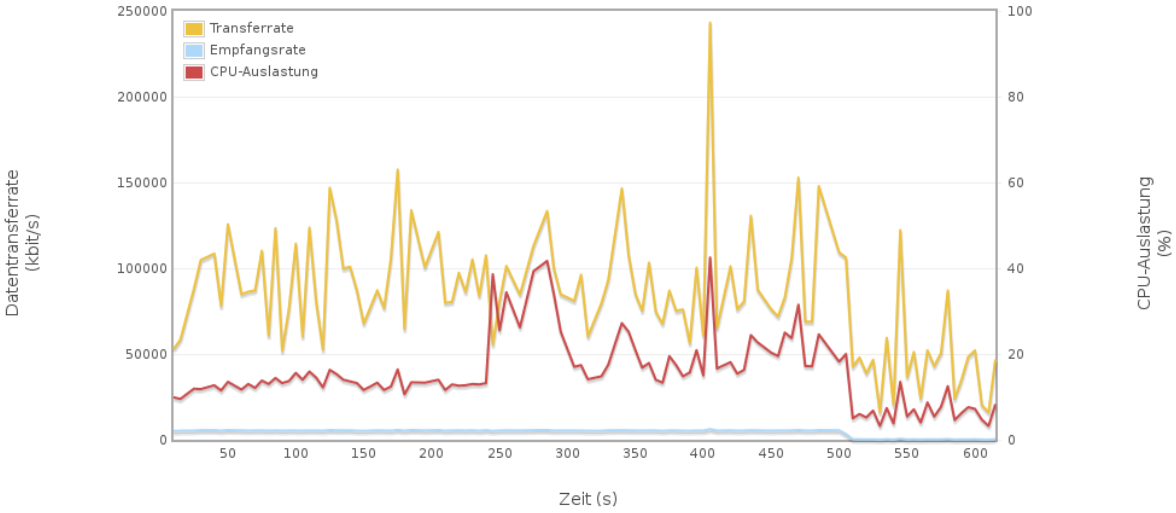

Abbildung 14: Gebündeltes Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

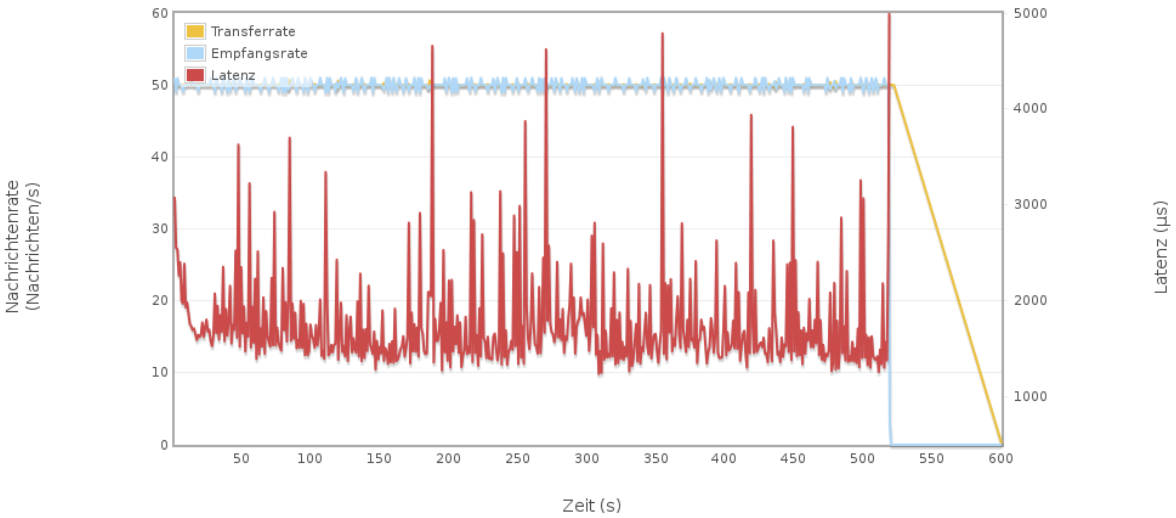

Abbildung 15: Gebündeltes Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Queue-Churning**
Messwerte	CPU: 44% RAM: 1542MB (Verwendet) HDD: 784MB (Frei) NET: RCX 265kbit/s TRX 358kbit/s
Beobachtungen	Direkt zu Beginn steigen sowohl der Bedarf an RAM als auch die CPU-Auslastung sprunghaft an. Auffallend ist, dass der für das Speichern von Nachrichten gesetzte Schwellwert bei ca. 500MB, nicht für die Speicherung von Queues gilt und kein Festplattenspeicher verwendet wird. Der RAM-Bedarf steigt somit kontinuierlich an, bis er an das physisch (bzw. virtuell) gesetzte Limit stößt. Bereits bei Überschreiten der 1GB RAM-Grenze schließt der RabbitMQ-Server alle Verbindungen und verweilt in diesem Zustand, wobei weitere Kommunikation abgelehnt wird (Siehe Abb.24).
Anmerkungen	Der Angriff wurde einige Sekunden zu früh manuell beendet, was in dem Abfallen des Ressourcenverbrauches und den Peaks in der 560s in Abbildungen 16, 17 und 24 zu sehen ist.

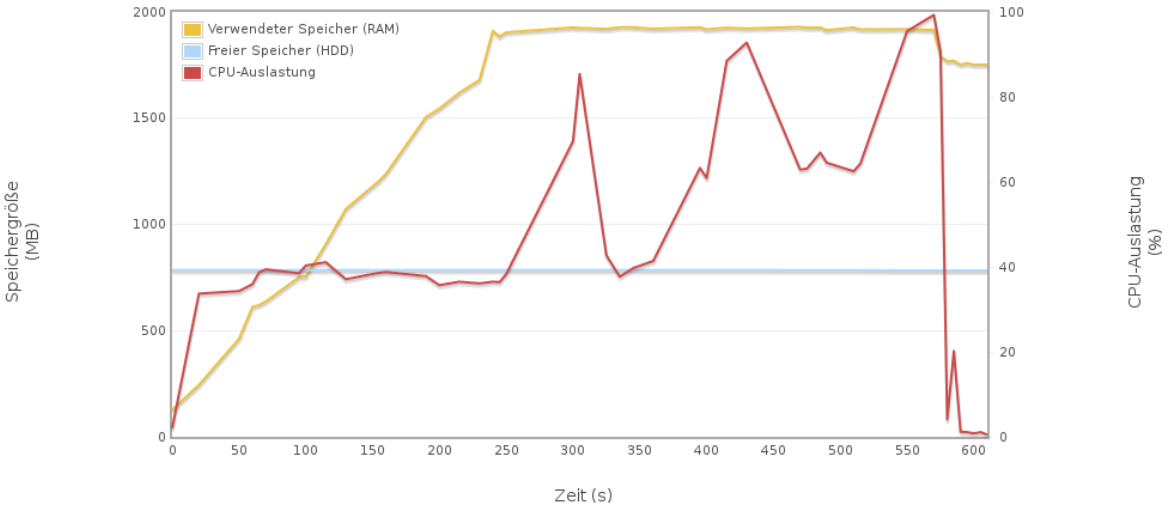

Abbildung 16: Queue-Churning - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

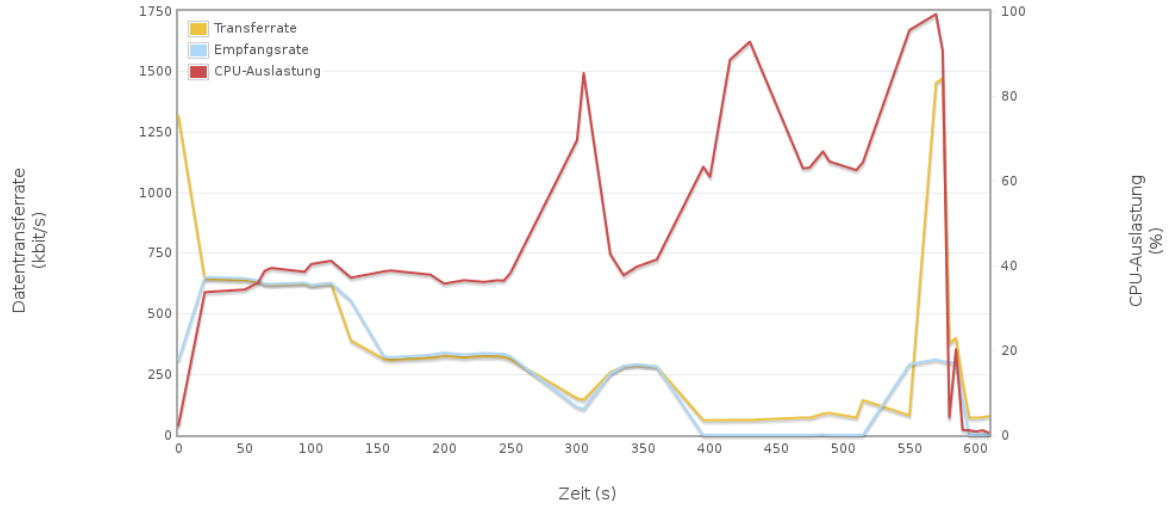

Abbildung 17: Queue-Churning - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

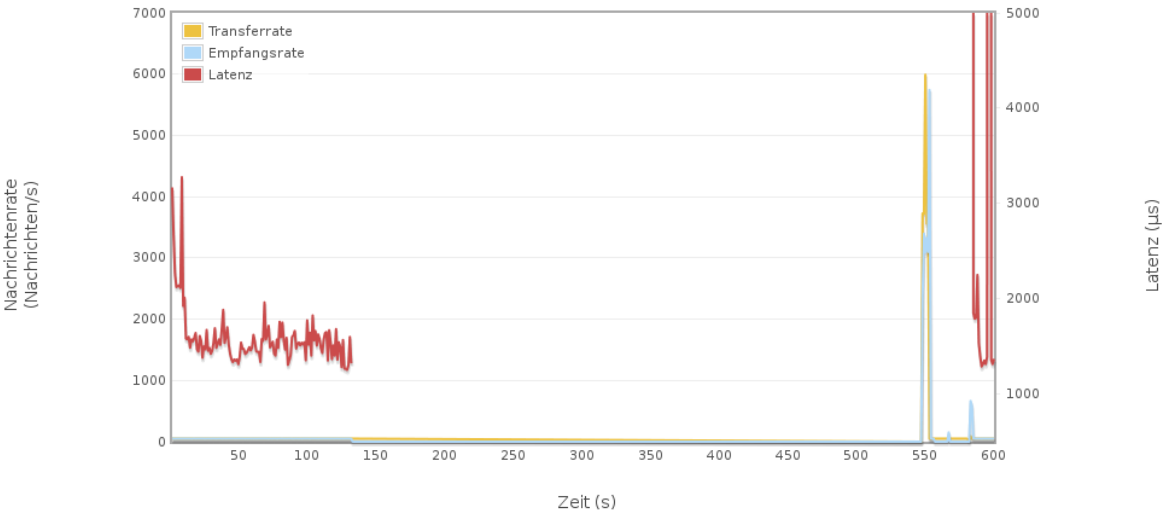

Abbildung 18: Queue-Churning - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Ausbleiben von Commits im Transaktionsmodus**
Messwerte	CPU: 2% RAM: 955MB (Verwendet) HDD: 785MB (Frei) NET: RCX 924kbit/s TRX 83bit/s
Beobachtungen	Durch Auslassen des Commits steigt der Speicherverbrauch stark an. Nach wenigen Sekunden ist die Speichergrenze erreicht und es werden keine weiteren Nachrichten übertragen. Daraufhin verharren alle Producer bis der Speicher wieder freigegeben wird (Falls Consumer vorhanden).
Anmerkungen	Die Messung des Andwendungsszenario konnte bei diesem Angriff nicht durchgeführt werden, da das dafür verwendete Tool „HTML Performance Tools“ bei der Messung reproduzierbar einfriert. Sporadisch friert der RabbitMQ-Server bei diesem Versuch ein. Ein Aufbau einer neuen Verbindung schlägt fehl und die Weboberfläche ist nicht mehr erreichbar. Nach längerer Zeit (etwa 10 Minuten) baut der Server aber wieder die Verbindungen ab und ein Zugriff ist wieder möglich. Dennoch bleibt der Speicherverbrauch bestehen.

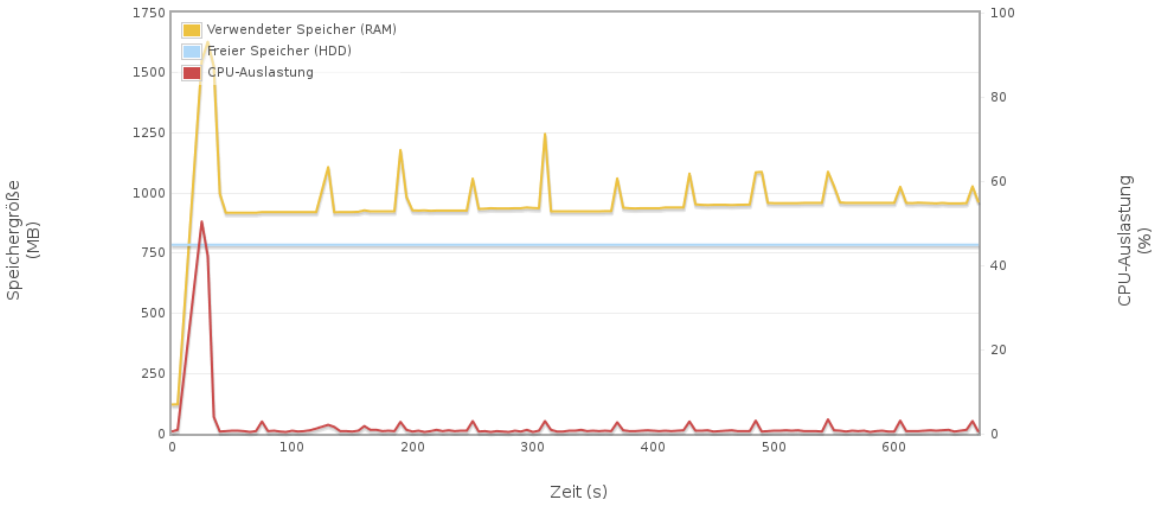

Abbildung 19: Ausbleiben von Commits im Transaktionsmodus - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

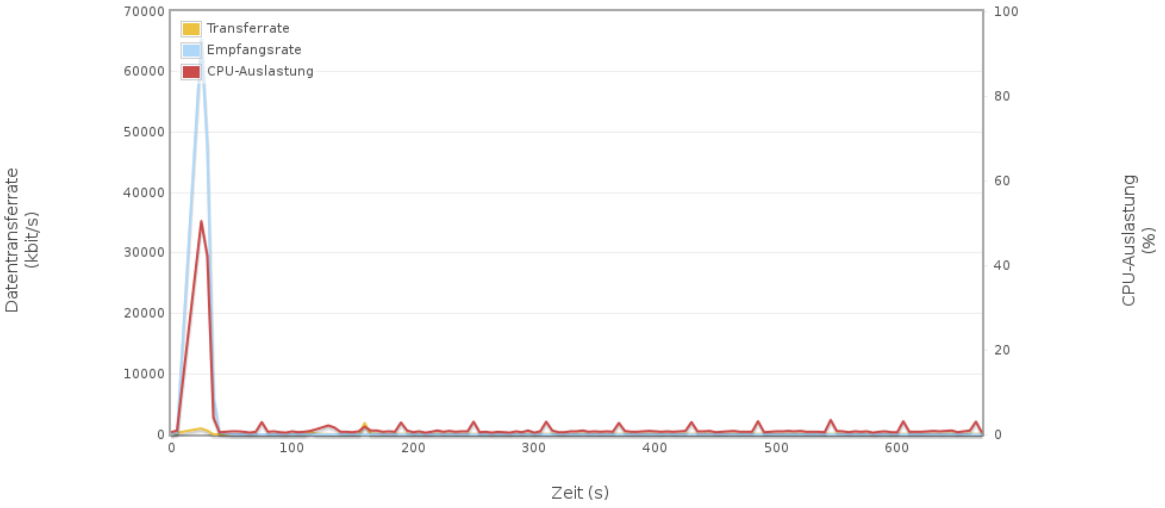

Abbildung 20: Ausbleiben von Commits im Transaktionsmodus - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

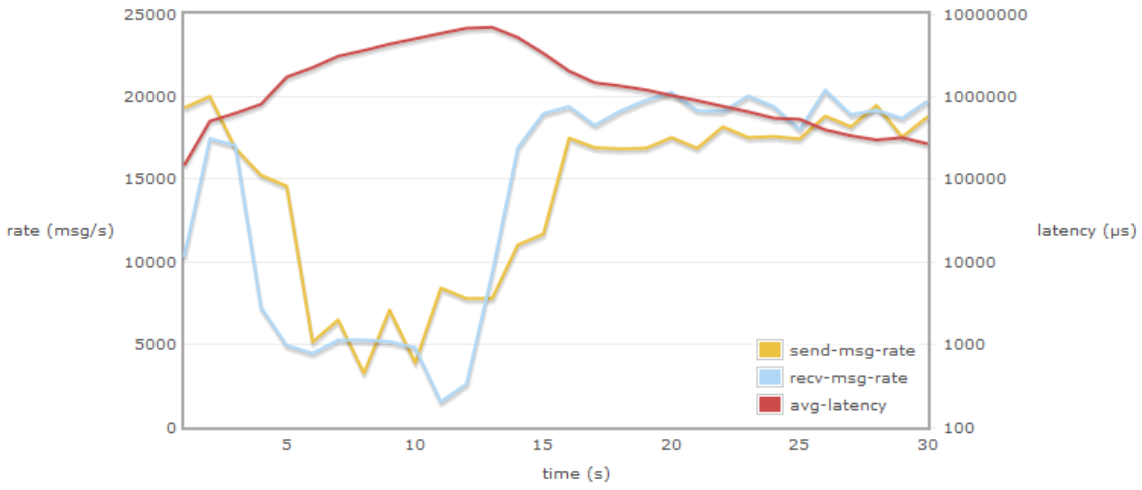

Abbildung 21: Ausbleiben von Commits im Transaktionsmodus - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario bei geringer Testzeit

Angriff	**Nachrichten mit großem Header**
Messwerte	CPU: 2% RAM: 468MB (Verwendet) HDD: 391MB (Frei) NET: RCX 2.8Mbit/s TRX 33kbit/s
Beobachtungen	Die Anwendung generiert zu Beginn 2500 Weiterleitungsoptionen (Bestehend aus 8 Byte Key und 36 Byte Value = 44 Byte pro Eintrag) und schreibt sie in den Header jeder Nachricht. Hierdurch ist der RabbitMQ-Server blitzartig massiv ausgelastet, versucht jedoch die Nachrichten auf die Festplatte auszulagern. Die Datenübertragungsrate sinkt dabei auf 20-30 Nachrichten/s. Hierbei macht es keinen Unterschied ob der Key oder Value vergrößert wird, die geringe Übertragungsrate bleibt bestehen. Der Value wird daher nicht ausgewertet. Die Headergröße ist bei etwa 2500 Weiterleitungsoptionen begrenzt. Wird diese überschritten lässt das System die Verbindung fallen, aufgrund einer zu großen Framegröße.
Anmerkungen	Die Framegröße lässt sich beim Aufbau der Verbindung angeben, kann aber die bereits voreingestellten eingestellten 128 Byte nicht überschreiten.

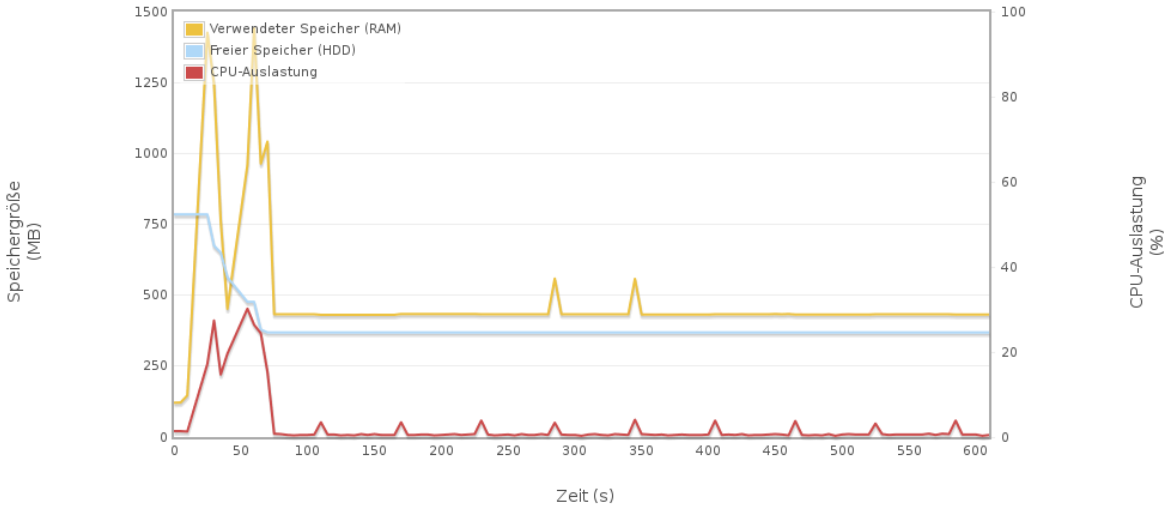

Abbildung 22: Nachrichten mit großem Header - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

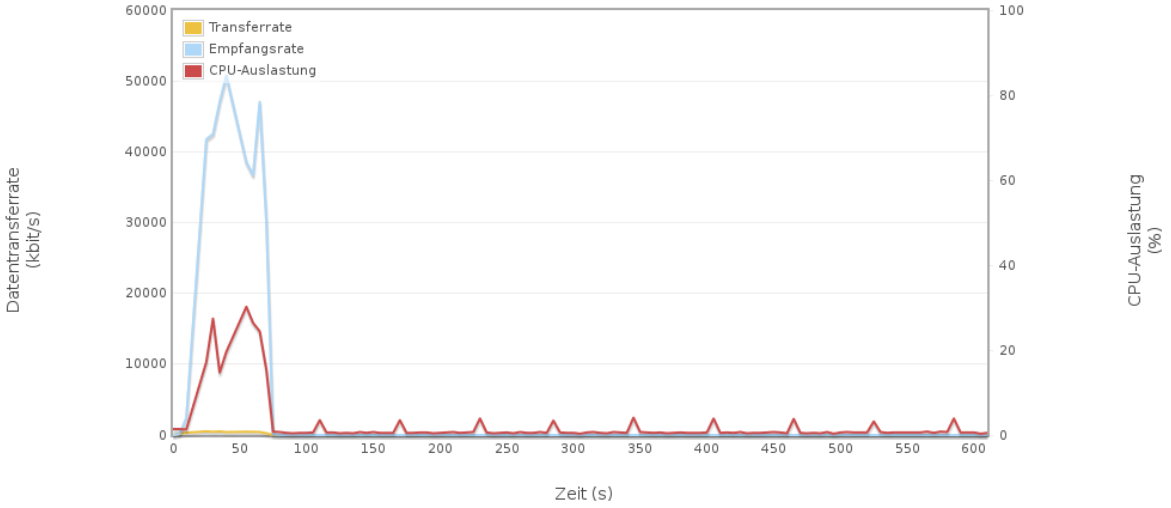

Abbildung 23: Nachrichten mit großem Header - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

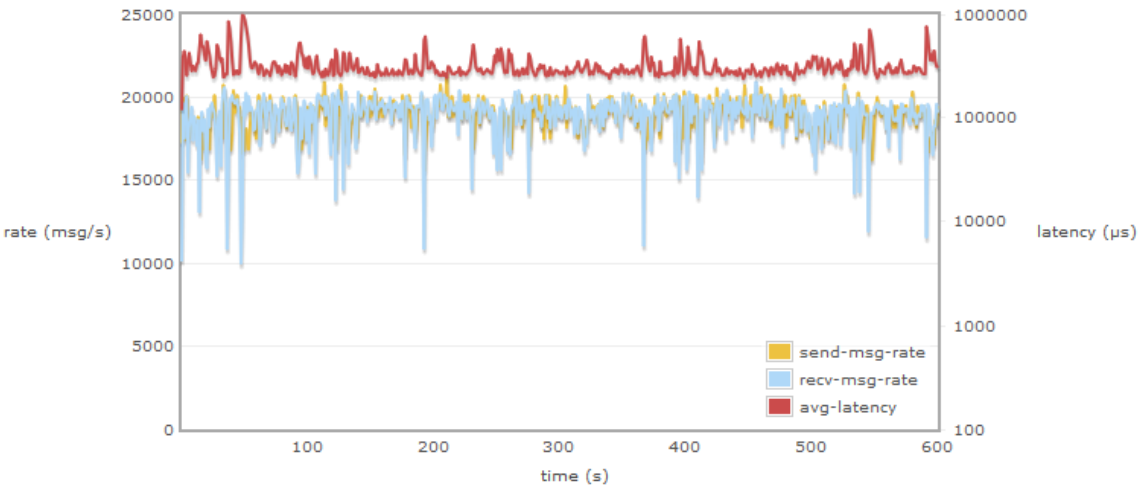

Abbildung 24: Nachrichten mit großem Header - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Übertragungsrate nach Nachrichtengröße			Headergröße
10.000 Byte	1.000 Byte	100 Byte	Einträge (Byte)
140m/s	260m/s	350m/s	200 (8.800)
80m/s	120m/s	180m/s	500 (22.000)
30m/s	50m/s	70m/s	1.000 (44.000)
20m/s	30m/s	40m/s	2.000 (88.000)
10m/s	10m/s	20m/s	2.500 (110.000)
270m/s	3.000m/s	10.000m/s	kein Eintrag

Tabelle 2: Vergleich Headergröße

Angriff	**Channel-Flooding**
Messwerte	CPU: 76% RAM: 298MB (Verwendet) HDD: 785MB (Frei) NET: RCX 116.7Mbit/s TRX 205.0Mbit/s
Beobachtungen	Das System ist stark ausgelastet, ähnelt aber der Auslastung unter der Erstellung mehrerer Verbindungen. Allerdings beansprucht der Aufbau der Channel verhältnismäßig viel Zeit. Die Datenübertragungsrate steigt mit der Anzahl der Channel, wobei die CPU-Auslastung ebenfalls steigt (Siehe Abb. 25,26). Bedingt durch die hohe CPU-Beanspruchung, verdoppelt sich die Latenz im Anwendungsszenario, wobei vereinzelt Peaks entstehen (Siehe Abb. 27).
Anmerkungen	Zeit für Aufbau der Channel hängt stark von der Anzahl von Producer und Consumer ab. Nachfolgend zeigt sich ein Vergleich von mehreren Kanälen sowie mehreren Verbindungen.

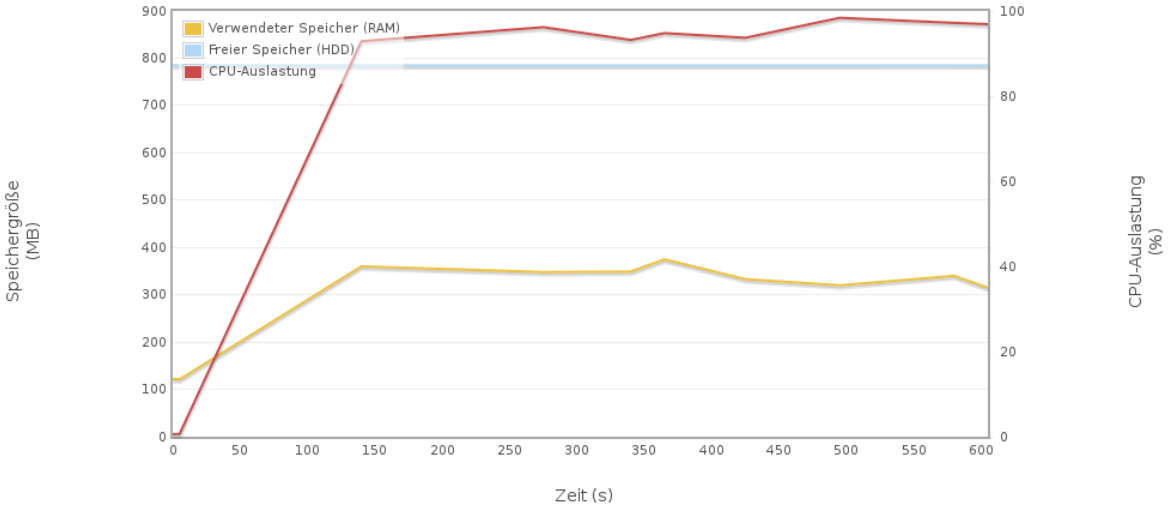

Abbildung 25: Channel-Flooding - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

Übertragungsrate (schreiben)	Producer	Consumer	Nachrichtengröße (Byte)
7.000m/s	100	10	100
800m/s	100	10	1.000
100m/s	100	10	10.000

Tabelle 3: Mehrere Channel

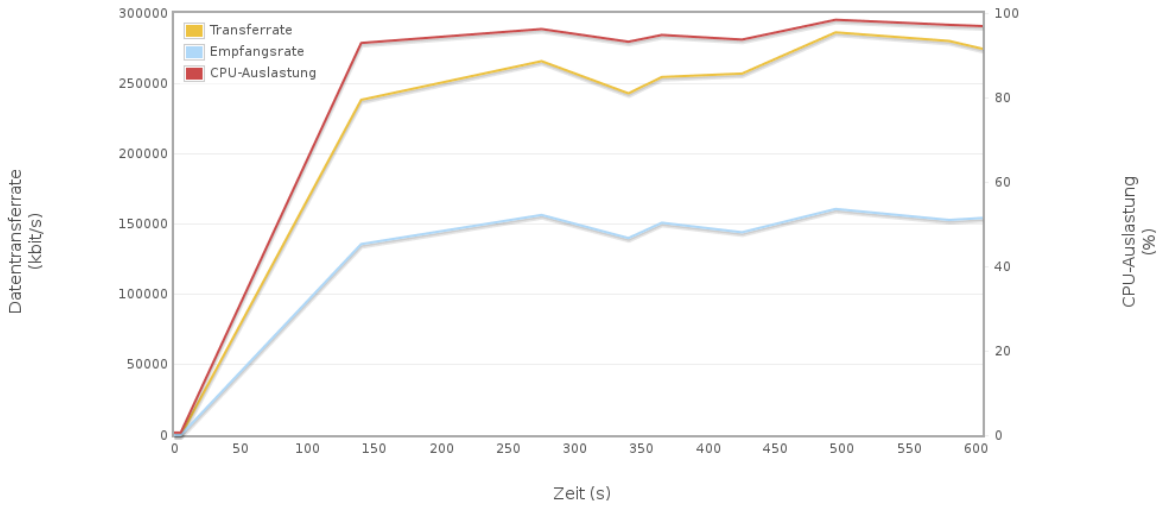

Abbildung 26: Channel-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

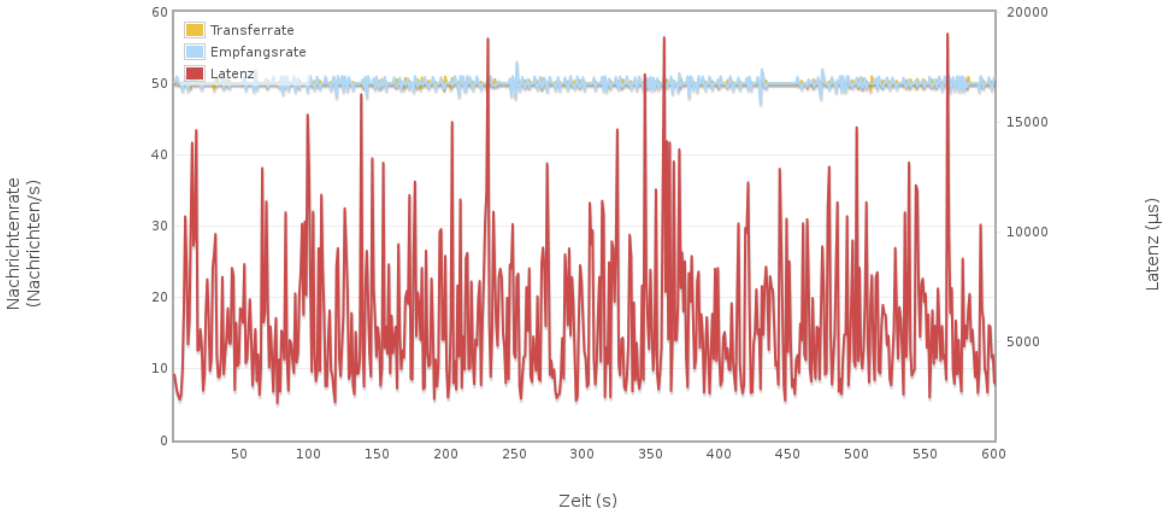

Abbildung 27: Channel-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Übertragungsrate (schreiben)	Producer	Consumer	Nachrichtengröße (Byte)
24.000m/s	100	10	100
2.000m/s	100	10	1.000
100m/s	100	10	10.000

Tabelle 4: Mehrere Verbindungen

Angriff	**Handshake-Trickle**
Messwerte	CPU: 2% RAM: 125MB (Verwendet) HDD: 785MB (Frei) NET: RCX 293kbit/s TRX 355kbit/s
Beobachtungen	Die Auswirkungen des Angriffes sind kaum wahrnehmbar und unterscheiden sich nur marginal von der alleinigen Ausführung des Anwendungsszenarios. Selbst bei 20 parallel laufenden Handshakes ist die Auswirkung kaum verändert. Der Angriff könnte jedoch, in Analogie zum „Sync-Flooding“ bei TCP, dazu umgebaut werden, das der Handshake lediglich gestartet wird, um ihn anschließend ohne Rückmeldung zu schließen. Dabei wird für mindestens 10s ein TCP-Socket des Servers blockiert, was den Server schnell an die Grenze der zur Verfügung stehenden Socketdeskriptoren bringt.
Anmerkungen	Da im RabbitMQ-Server standardmäßig ein Handshake-Timeout von 10s gesetzt ist, verwendet der vorliegende Angriff jeweils 9 Pause zu je 1s. Dabei wurde ebenfalls versucht den Timeout auf 30s hochzusetzen, was vom Server jedoch ignoriert wurde.

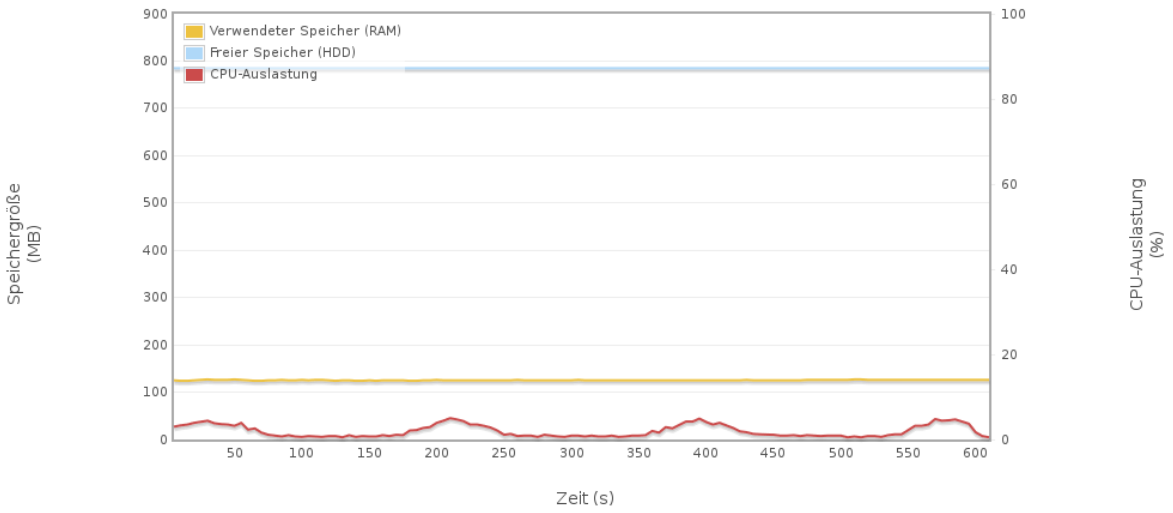

Abbildung 28: Handshake-Trickle - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

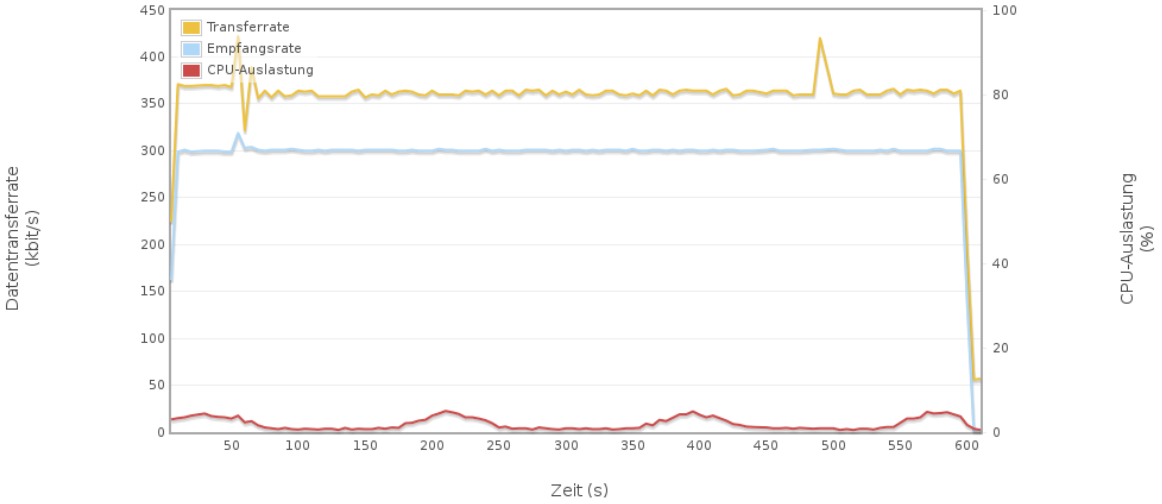

Abbildung 29: Handshake-Trickle - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

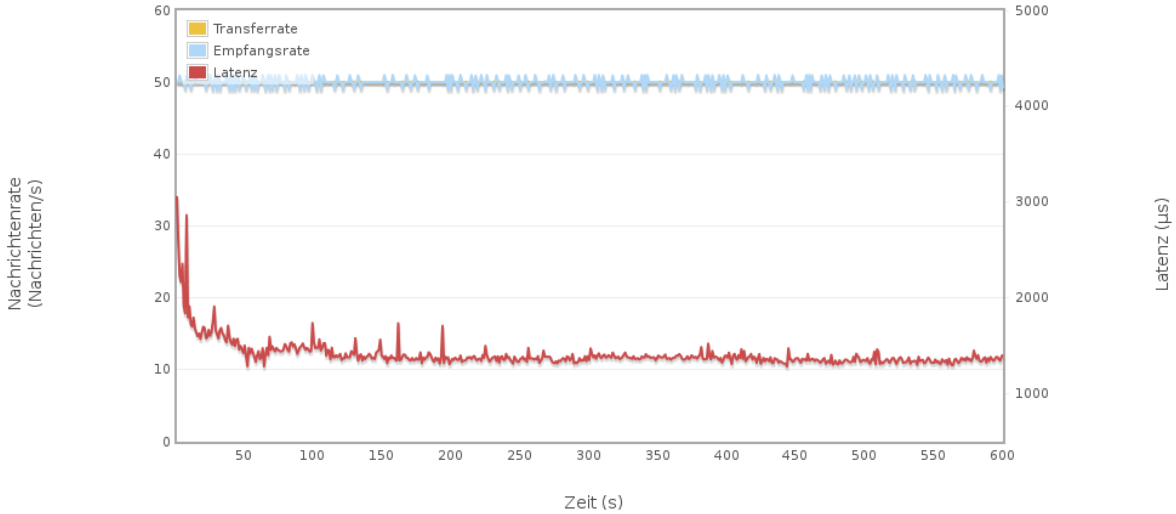

Abbildung 30: Handshake-Trickle - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Heartbeat-Flooding**
Messwerte	CPU: 4% RAM: 134MB (Verwendet) HDD: 785MB (Frei) NET: RCX 318kbit/s TRX 386kbit/s
Beobachtungen	Die Auswirkungen des Angriffes sind kaum wahrnehmbar und unterscheiden sich nur marginal von der alleinigen Ausführung des Anwendungsszenarios. So ist die CPU-Auslastung und die Netzwerkauslastung etwas erhöht, wobei der Peak zu Beginn der Messung durch die Erzeugung der 20 Connections verursacht wurde. Die Latenz sowie die Nachrichtenraten sind im Anwendungsszenario weitgehen konstant. Ergänzung: Es wurde eine Vergleichsmessung mit dem Standard-Heartbeat von 580s durchgeführt, was zu einer Halbierung der CPU-Auslastung (2%) und einer minimalen Verbesserung der Latenz geführt hat. Die Netzwerkauslastung hat sich erwartungsgemäß ebenfalls verringert.
Anmerkungen	Während der Tests wurde versucht, den Heartbeat auf den maximal möglichen Wert zu setzen. Der RabbitMQ-Server ignoriert jedoch alle Anfragen, die darauf Abzielen den Heartbeat auszusetzen oder ihn höher als auf 580s zu setzen. Somit ist der Heartbeat stets ein Wert zwischen 1 und 580s.

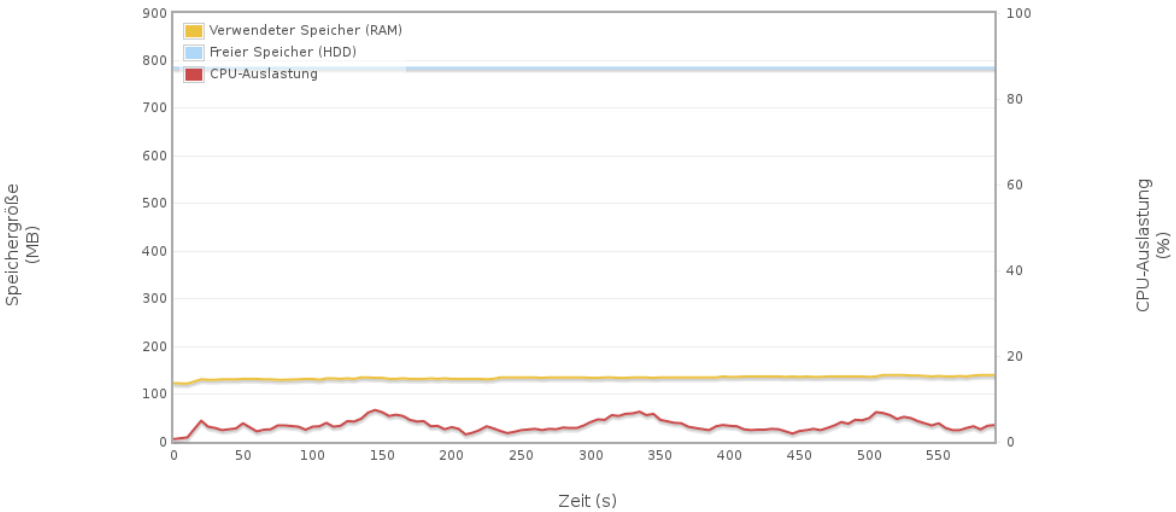

Abbildung 31: Heartbeat-Flooding - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

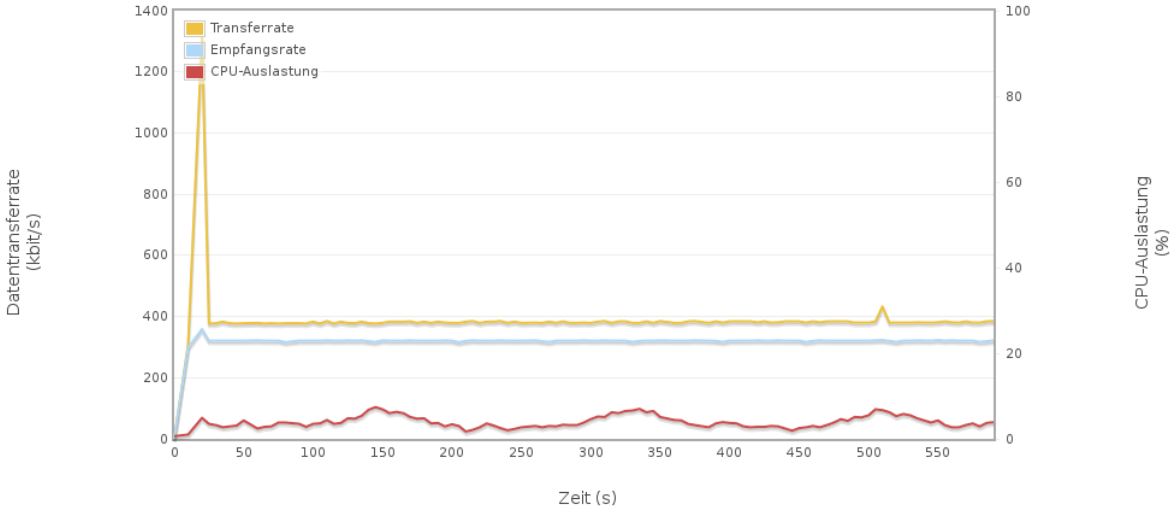

Abbildung 32: Heartbeat-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

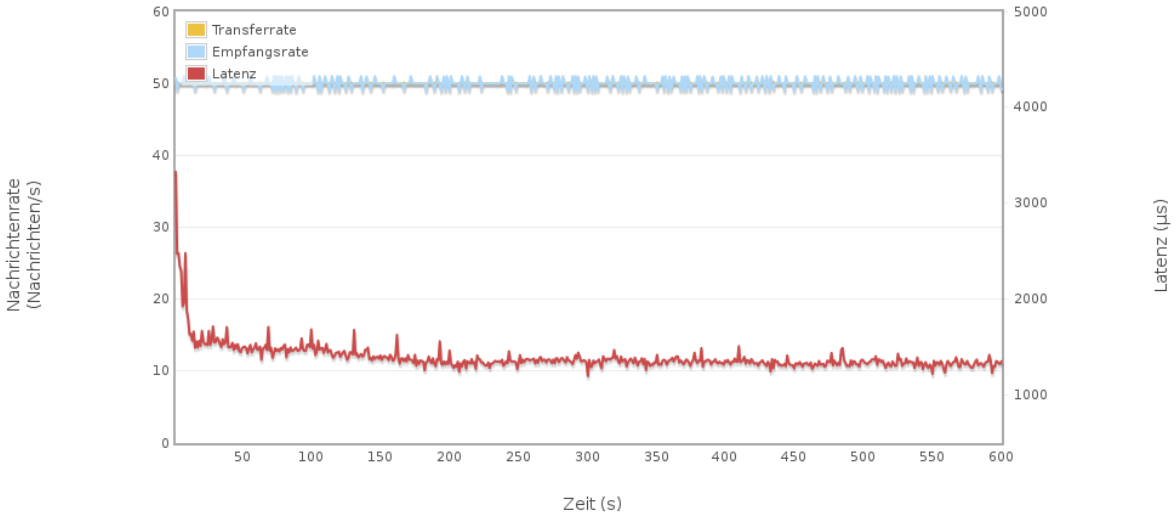

Abbildung 33: Heartbeat-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**TCP-Connection-Dropping**
Messwerte	CPU: 2% RAM: 416MB (Verwendet) HDD: 785MB (Frei) NET: RCX 303kbit/s TRX 522kbit/s
Beobachtungen	Mit steigender Anzahl an unbemerkt geschlossenen Connections, steigt auch der Bedarf an TCP-Sockets auf dem Server. Das überschreiten des Schwellwertes geht deutlich bei der 150. Sekunde aus Abb. 34 hervor. Danach werden keine neuen Connections mehr akzeptiert, weshalb der Speicherbedarf weitgehend konstant ist. Da gegen Ende der Messung die ersten Verbindungen durch den Heartbeat-Timeout geschlossen werden, können kurzzeitig neue Verbindungen aufgebaut werden. Aus Abbildung 36 geht dabei hervor, dass einzig die Latenz in dem Moment der Überschreitung des Socketdeskriptoren-Schwellwertes kurzzeitig beeinflusst wird.
Anmerkungen	In der Testumgebung standen exakt 1024 Filedeskriptoren zur Verfügung, wovon 829 für Sockets reserviert waren.

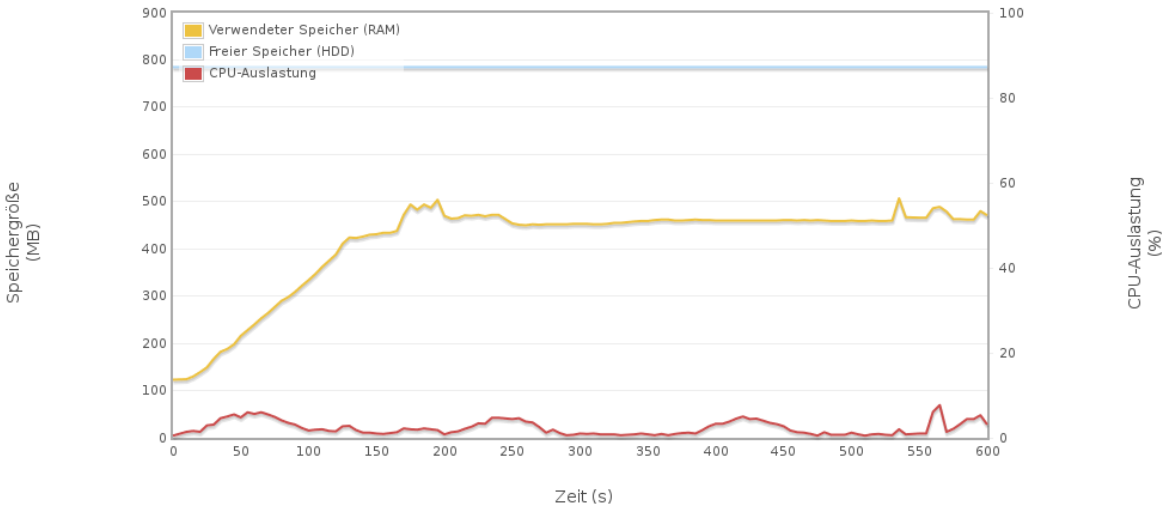

Abbildung 34: TCP-Connection-Dropping - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

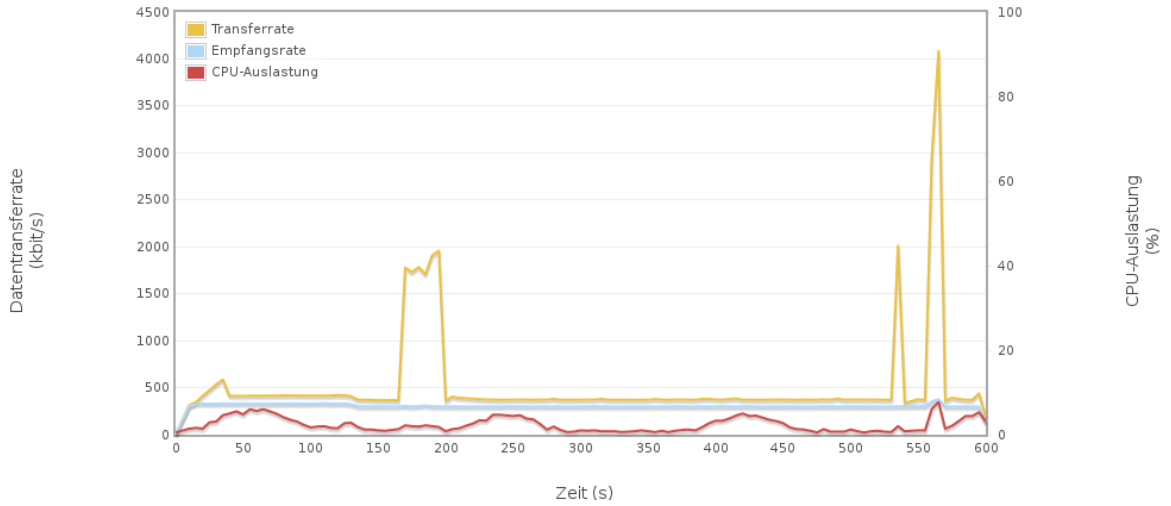

Abbildung 35: TCP-Connection-Dropping - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

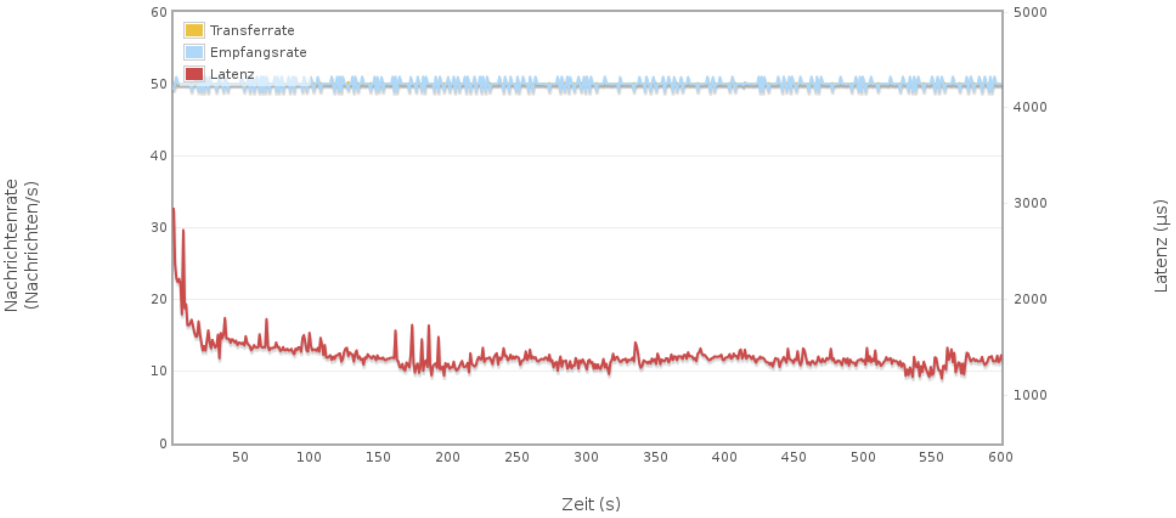

Abbildung 36: TCP-Connection-Dropping - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Ausblick

Die hier gezeigten Angriffsszenarien stellen lediglich eine kleinen Ausschnitt der denkbaren Angriffe dar. Hierbei wurden zum Teil bekannte Angriffe auf andere Protokollen adaptiert und ggf. ausgebaut. So existieren in der Literatur jedoch weitere Angriffe z. B. für das TCP. Nachfolgend wird eine kleine Auswahl der Angriffe beschrieben und deren mögliche Adaptierung auf das AMQP diskutiert.

SYN Flood-Angriff SYN-Flooding ist eines der ältesten Angriffe, dennoch aktuell und von großer Bedeutung. Bei dem Aufbau einer Verbindung zwischen Client und Server wird ein 3-Wege-Handshake durchgeführt. Die Idee hinter diesem Angriff ist es möglichst viele SYN-Pakete zu versenden, um so die Ressourcen des Servers auszulasten. Durch das Ignorieren der Rückantwort (ACK) des Angreifers, wird der Server dennoch gezwungen einige Zeit auf das entsprechende Paket zu warten.

Die SYN-Pakete sind einfach zu erzeugen und entsprechend klein. Hierdurch ist es möglich den Server schnell auszulasten und somit einen Denial-of-Service (DoS) zu bezwecken. Auch bei einem RabbitMQ-Server ist dieser Angriff möglich, denn jede offene Anfrage zum Aufbauen einer Verbindung wird für 10 Sekunden gehalten. Genügend Zeit um alle Socket-Deskriptoren zu belegen. Auf dem unseren Test-System stehen nur 1024 zur Verfügung, welche nur in wenigen Sekunden ausgeschöpft sind. Das bedeutet, dass für eine ernst gemeinte Verbindung keine Ressourcen mehr zur Verfügung stehen. Die Wartezeit lässt sich über die config-Datei „rabbitmq.config“ beliebig anpassen, bietet dennoch keinen Schutz vor diesen Angriff.

Zusammenfassung

An dieser Stelle sollen noch einmal alle gesammelten Erfahrungen zusammentragen werden. RabbitMQ ist in der getesteten Version (3.5.3) ein sehr stabiler Message-Broker und dient zur verteilten Bearbeitung von Aufgaben und Entkopplung von Applikationsteilen. Dennoch zeigten sich unter allen getesteten Angriffsvektoren teils starke veränderte Messwerte in Bezug auf die Übertragungsrate oder der Ressourcen des Servers selbst. Viele Ergebnisse zeigten, dass das Auslagern der Nachrichten vom Arbeitsspeicher auf die Festplatte Auswirkungen auf den Server hat. Können die Nachrichten nicht mehr im Arbeitsspeicher gehalten werden, konzentrieren sich die Ressourcen auf das Auslagern und die Clients werden weitgehend zurückgehalten. Somit zeichnet sich eine geringe Übertragungsrate der Nachrichten aus. Erst nach Beendigung des Schreibvorgangs steigt die Übertragungsrate wieder an. Da der Server in unserem Projekt auf einer virtuellen Maschine beruht und somit stark eingeschränkt ist, zeichnen sich die Unterschiede hier sehr stark ab.