Zielstellung und Rahmenbedingungen

Diese Dokumentation beschäftigt sich mit dem Thema „Angriffe auf AMQP-Messagebroker“. Begleitend zur Vorlesung „Betriebliche Informationssysteme“ galt es von uns in einem Praktikum zu identifizieren, welche potenziellen Angriffsvektoren existieren. Eingeschlossen ist hier auch das System „RabbitMQ“ welches das zu untersuchende Protokoll einschließt.

Ziel ist es bestimmt Angriffsvektoren aufzuspüren und zu bewerten. Durch Implementierung von einzelnen Clients auf Basis der RabbitMQ Java-Bibliothek sollen die Angriffe veranschaulicht werden. 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.

Testumgebung

Als Betriebssystem für die Bereistellung des 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 Netzwerkinterface 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 RabbitMQ-Benutzerdaten und Adressen.

Verwendete bzw. Erstellte Programme

RabbitMQ bietet einige Anwendungen, mit denen sich die Leistungsfähigkeit der Server messen lässt. Alle Hilfsprogramme sind innerhalb der *rabbitmq-client-tests.jar* vorzufinden. Diese JAR-Datei enthält weiterhin zahlreiche kleine Beispielprogramme für das Testen 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. Als Grundlage dient die Servervariante von Ubuntu 14.04 LTS, welche durch die fehlende grafische Oberfläche zum einsparen von Ressourcen dient.

Glances Glances ist ein System-Monitoring-Tool, das zahlreiche Informationen auf engstem Raum präsentiert. Im einfachsten Fall werden die gesammelten System-Informationen auf der Konsole ausgeben. Ferner ermöglicht ein integriertes Web-Interface eine Fernüberwachung der Umgebung. Zu den gesammelten Informationen gehören unter anderem: CPU-Auslastung, Kernel-Version, Speicherverbrauch (HDD, RAM) und laufende Prozesse. Hierbei werden alle Informationen im Sekundentakt aktualisiert.

Perftest ist ein Performance-Test-Tool, welches beliebig viele Producer und Consumer erstellt und die Sende-/Empfangsrate misst. Zusammen mit der Latenzzeit werden alle Angaben auf der Konsole ausgeben.

HTML Performance Tools bieten ebenfalls ein breites Spektrum an Funktionalität. Mit einer Reihe von Tools und Unterstützung von perftest lassen sich automatisierte Benchmarks erstellen. Die gelieferten Daten dienen dem Vergleich der Systems vor und während des Angriffs. Alle Ergebnisse werden in einer JSON-Datei gesichert und ansprechend in einer HTML-Seite dargestellt

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.

Beschreibung des Benchmarking

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 eine CSV-Datei exportiert, welche später zur Visualisierung und Auswertung verwendet wurde. Vor jeder Messung wurde die VM auf einen Snapshot zurückgesetzt, bei der sich das System in einem Zustand befindet, den es 5 minuten nach dem Hochfahren der VM besitzt.

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)

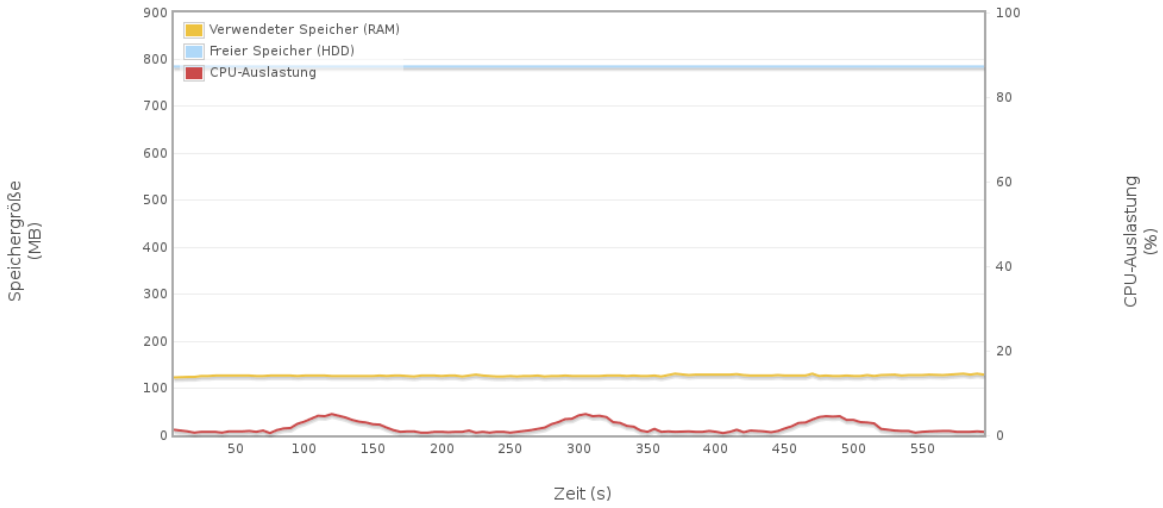

Abbildung 1: Ohne Angriff - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

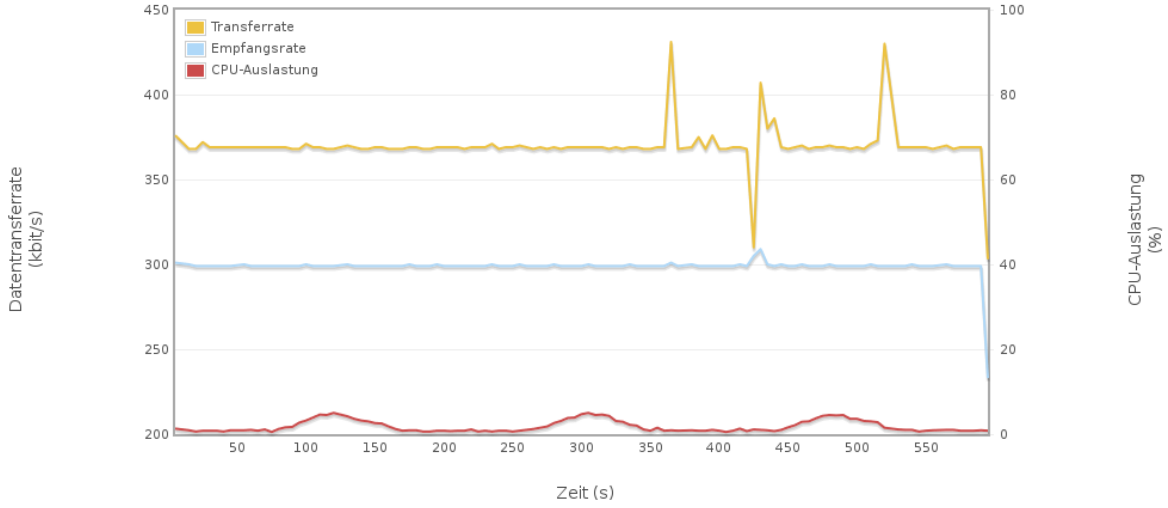

Abbildung 2: Ohne Angriff - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

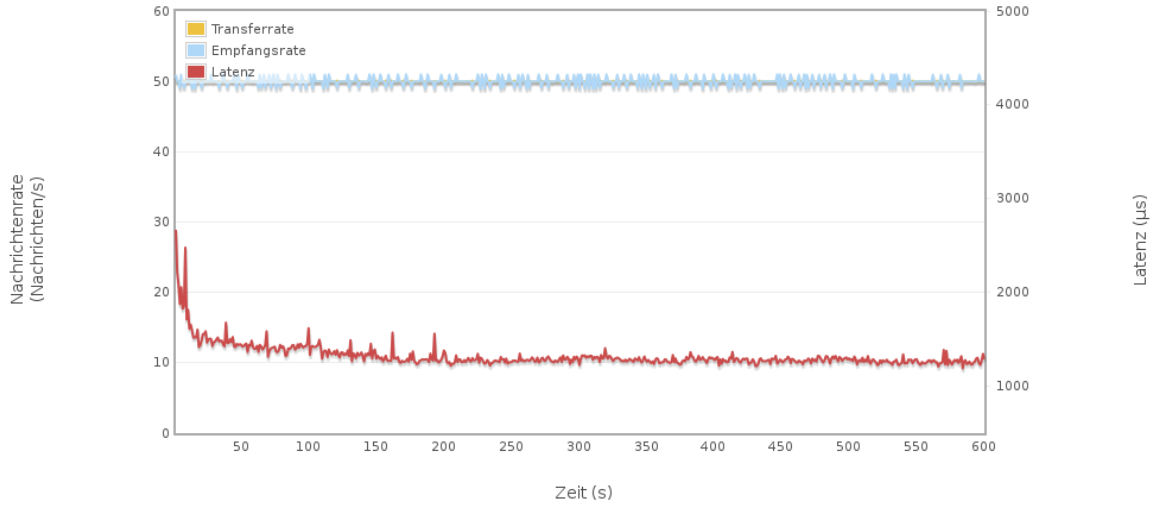

Abbildung 3: Ohne Angriff - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Beschreibung der Angriffe

A1	**Referenzangriff - Kooperativer Client**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten die über einen „fanout“-Exchange an 5 Consumer verteilt werden. Die Consumer quittieren dem RabbitMQ-Server dabei den Erhalt der Nachricht.
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)
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 gewisser 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 (Auf rufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 10240 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (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 gewisser 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 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard)
Befehlszeile	`Amqpstress -dm REJECT -c 5 -i 10 -ms 102400 -mp -u amqp://testc:testp@192.168.178.153:5672/%2f`

A4	**Gebündeltes Abweisen von Nachrichten**
Beschreibung	Ein Producer erzeugt kontinuierlich Nachrichten gewisser Größe mit zufälligem Inhalt, die von einem oder mehreren Consumer empfangen, zunächst ignoriert werden, um sie bei erreichen eines Schwellwertes gebündelt abzuweisen (basic.NACK). Dadurch ist der RabbitMQ-Server gezwungen alle Nachrichten zwischenzuspeichern und stoßweise alle Nachrichten bis zu der aktuellen Sequenznummer erneut zuzustellen.
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 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard)
Befehlszeile	`Amqpstress -dm NACK -c 5 -i 10 -ms 102400 -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 der Schwellwert an erzeugten Queues erreicht ist, werden alle Queues ohne zu Warten gelöscht und der Zyklus beginnt erneut. Der RabbitMQ-Server muss die Überreste der Queues im RAM bzw. der Festplatte bereinigen, während neue Queues angelegt und befüllt werden.
Testparameter	-dq (Aufrufparameter für Angriff) -pc 10 (Schwellwert: 10 Queues) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 0 (Keine Nachrichten senden) -u <uri> (URI für Verbindung mit Server) 1 Client - Consumer und Producer zugleich (Standard) Nicht-Persistente Nachrichten (Standard)
Befehlszeile	`Amqpstress -dq -pc 10 -i 10 -ms 0 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A6	**Nachrichten mit großem Header**
Beschreibung	RabbitMQ bietet die Möglichkeit im Header der Nachricht bestimmt Parameter für die Weiterleitung zu deklarieren. Dieser Test beschäftigt sich mit der Auswirkung, wenn der Header unnötig ausgelastet wird. Das System ist gezwungen alle Weiterleitungsoptionen zu Prüfen, auch wenn diese keinem Ziel entsprechen.
Testparameter	-lh (Aufrufparameter für Angriff) -ms 10000 (Messagegröße: 10000 Byte) -hs 2500 (Headergröße - Anzahl Einträge in Map) -u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -lh -ms 10000 -hs 2500 -u amqp://testc:testp@192.168.178.153:5672/%2f`

A7	**Channel-Flooding**
Beschreibung	Neben einzelnen Verbindungen können in RabbitMQ auch mehrerer Kanäle aufgebaut werden. Hier stellt sich die Frage, wie das System mit einer Vielzahl von Kanäle zurecht kommt. Auf Basis einer einzelnen Verbindung wird das System so ausgelastet und beobachtet.
Testparameter	-mc (Aufrufparameter für Angriff) -ms 10000 (Messagegröße: 10000 Byte) -u <uri> (URI für Verbindung mit Server) -p 100 (100 Producer) -c 10 (10 Consumer)
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 aber vergleichbar mit 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 100 (100 Producer) -ms 10000 (Messagegröße: 10000 Byte) -mct 10000 (Nachrichtenanzahl pro Producer: 10000) -co false (Commit der Transaktion - hier: Auslassen) -u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -tx -p 100 -ms 10000 -mct 10000 -co false -u amqp://testc:testp@192.168.178.153:5672/%2f`

A9	Handshake-Trickle
Beschreibung	.
Testparameter	-u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -u amqp://testc:testp@192.168.178.153:5672/%2f`

A10	Heartbeat-Flooding
Beschreibung	.
Testparameter	-u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -u amqp://testc:testp@192.168.178.153:5672/%2f`

A11	TCP-Connection-Dropping
Beschreibung	.
Testparameter	-u <uri> (URI für Verbindung mit Server)
Befehlszeile	`Amqpstress -u amqp://testc:testp@192.168.178.153:5672/%2f`

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

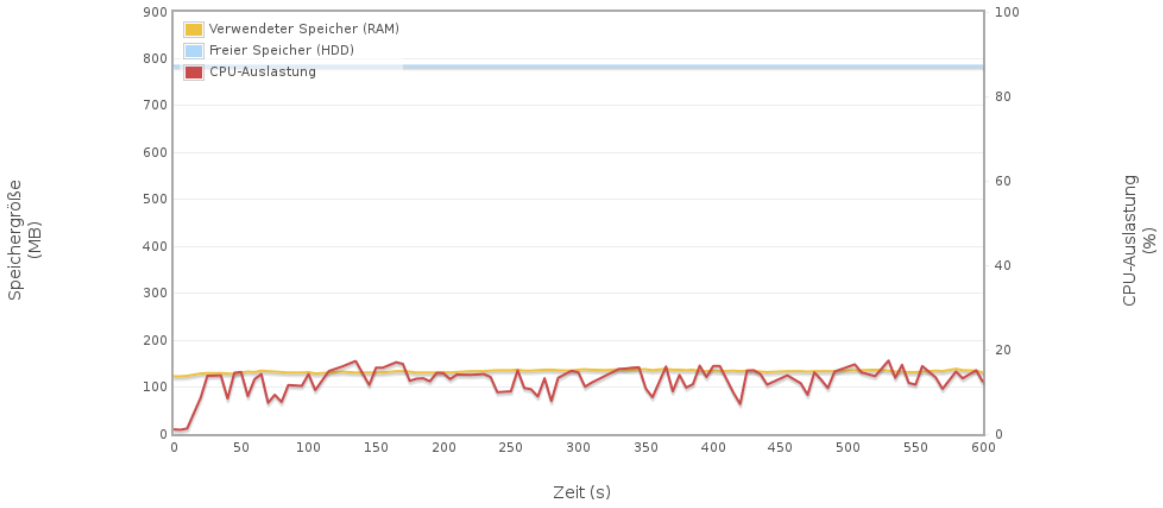

Abbildung 4: Kooperativer Client - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

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

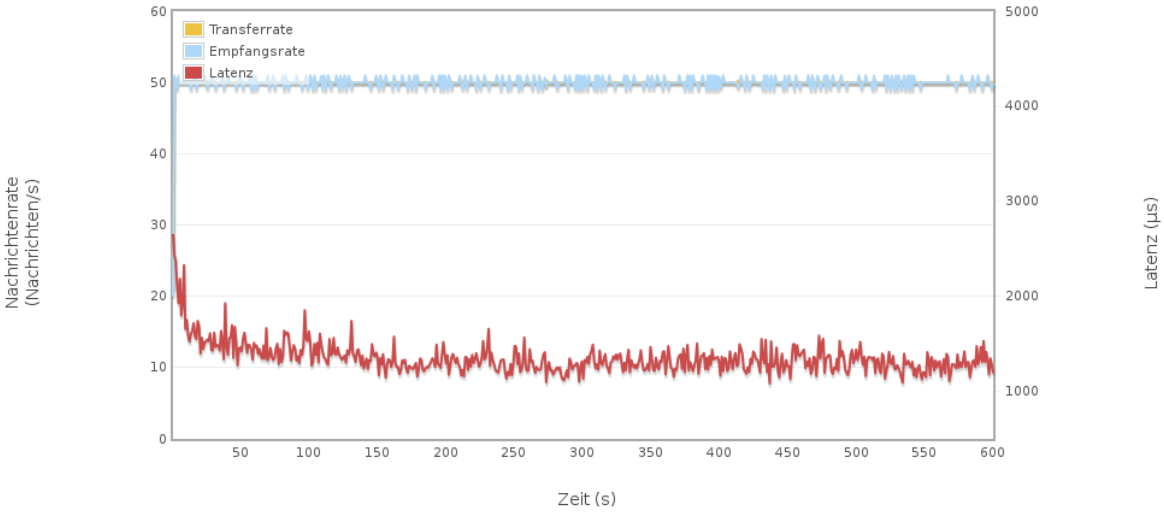

Abbildung 6: Kooperativer Client - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

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.
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.

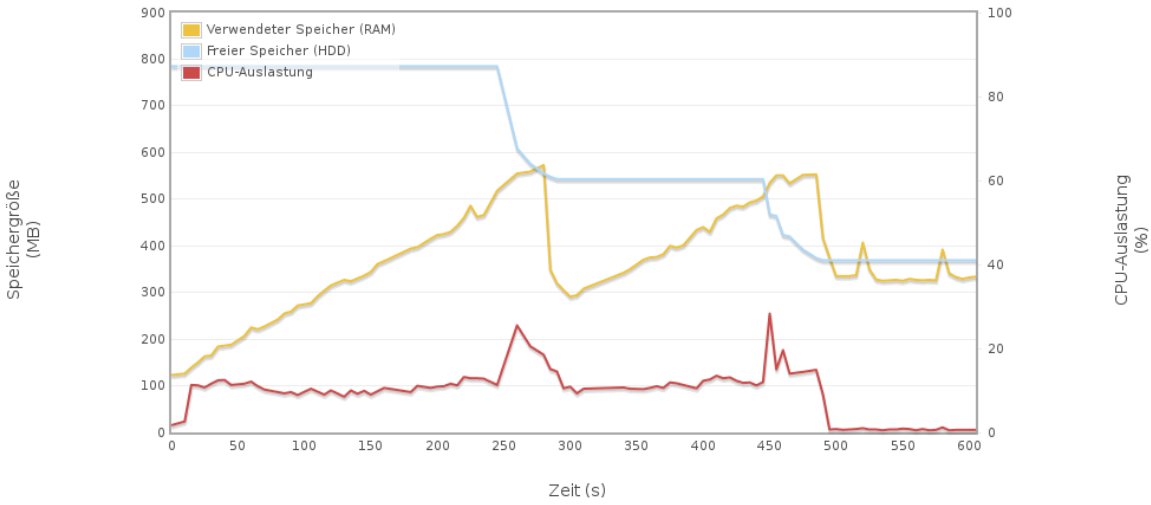

Abbildung 7: Ignorieren von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

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

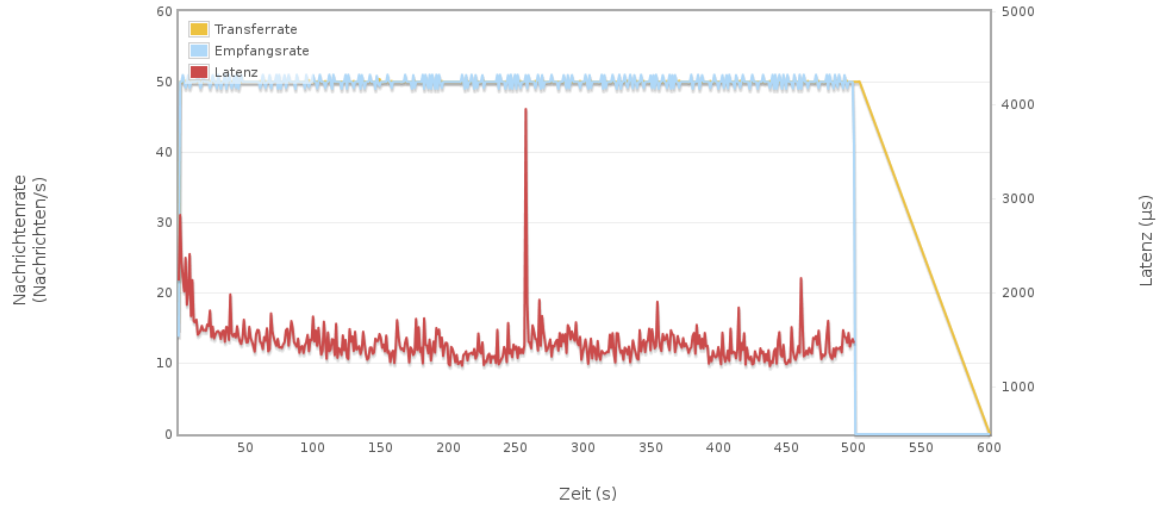

Abbildung 9: Ignorieren von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Sofortiges Abweisen von Nachrichten**
Messwerte	CPU: 24% RAM: 467MB (Verwendet) HDD: 603MB (Frei) NET: RCX 4.2Mbit/s TRX 28.2Mbit/s
Beobachtungen	.
Anmerkungen	.

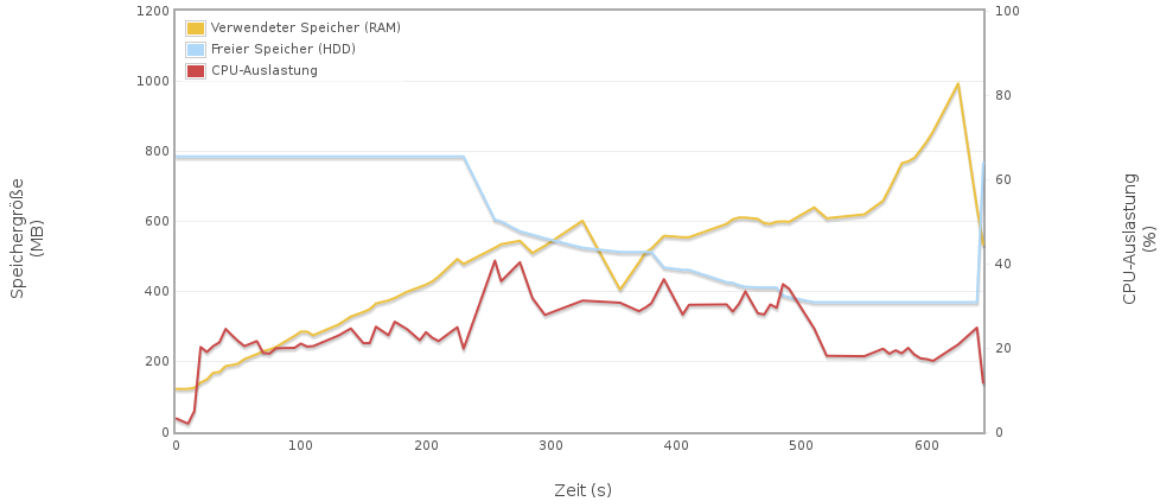

Abbildung 10: Sofortiges Abweisen von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

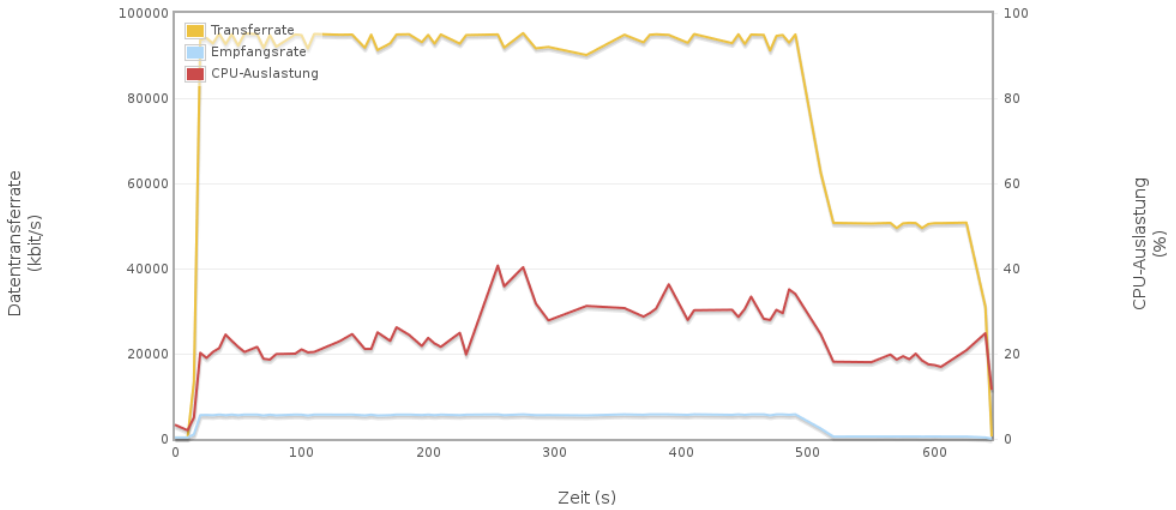

Abbildung 11: Sofortiges Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

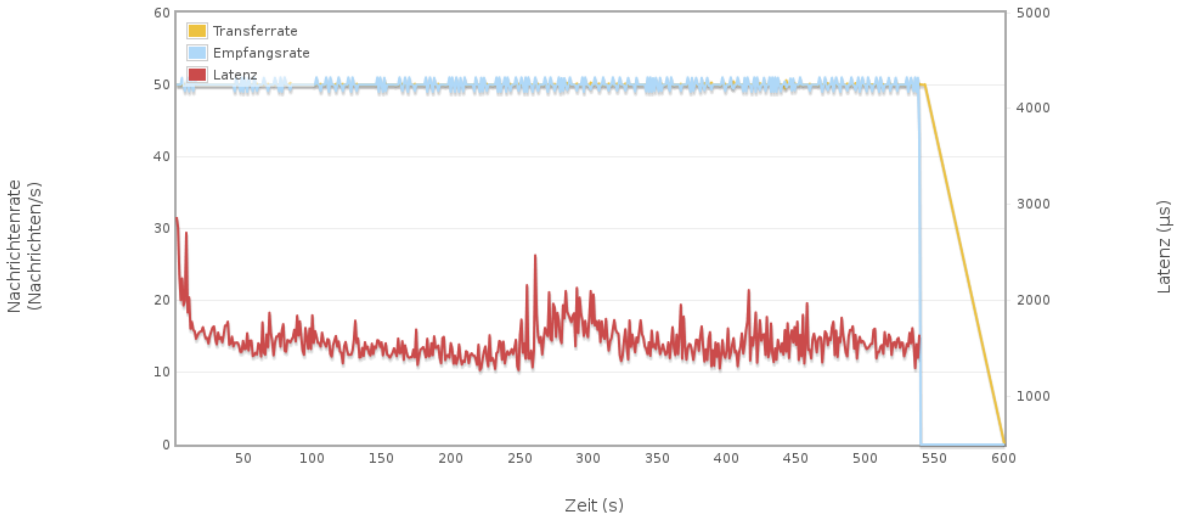

Abbildung 12: Sofortiges Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Gebündeltes Abweisen von Nachrichten**
Messwerte	CPU: 15% RAM: 534MB (Verwendet) HDD: 572MB (Frei) NET: RCX 4.2Mbit/s TRX 81.8Mbit/s
Beobachtungen	.
Anmerkungen	.

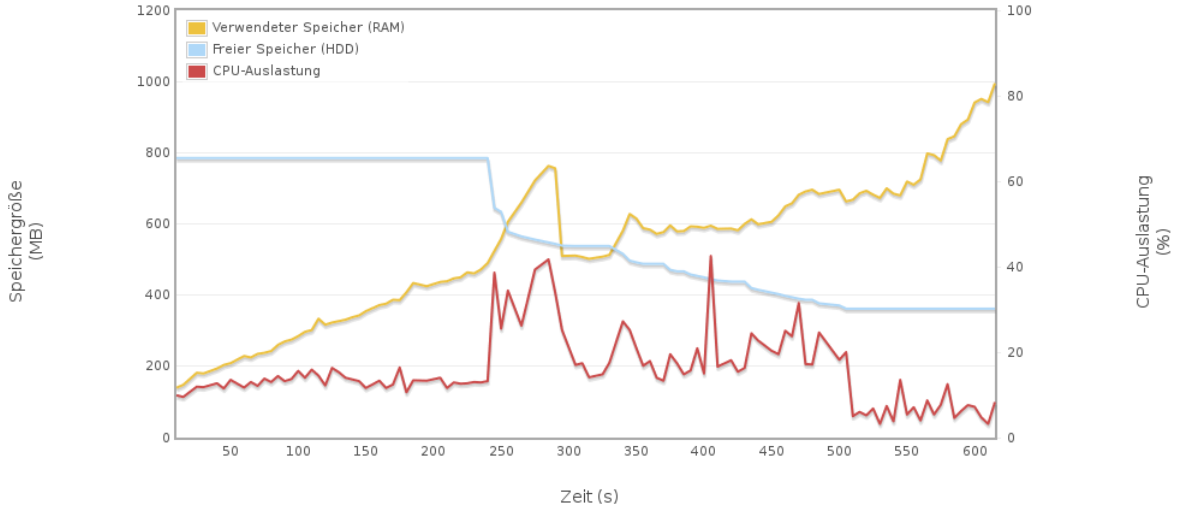

Abbildung 13: Gebündeltes Abweisen von Nachrichten - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

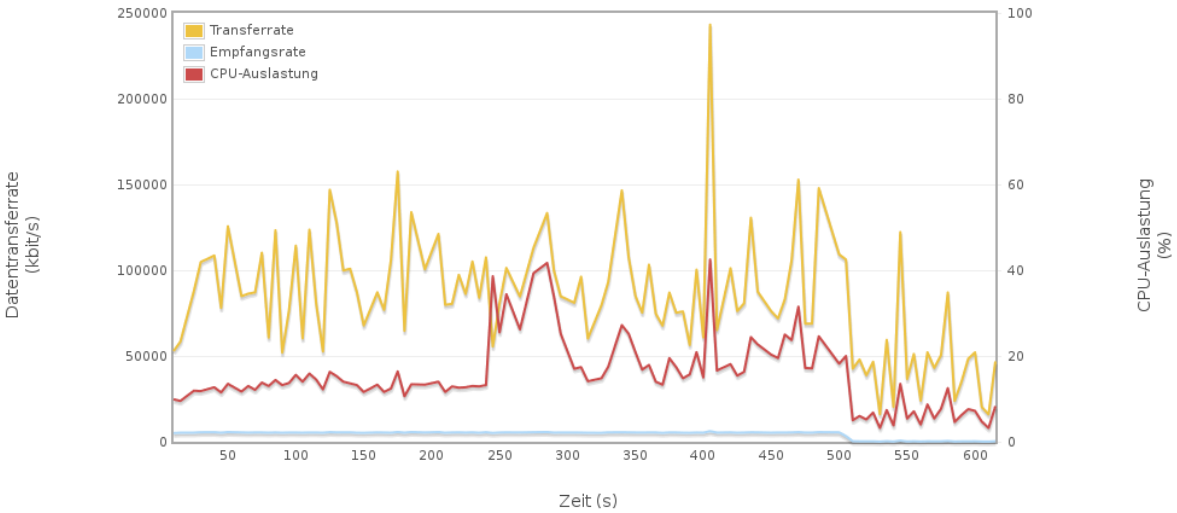

Abbildung 14: Gebündeltes Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

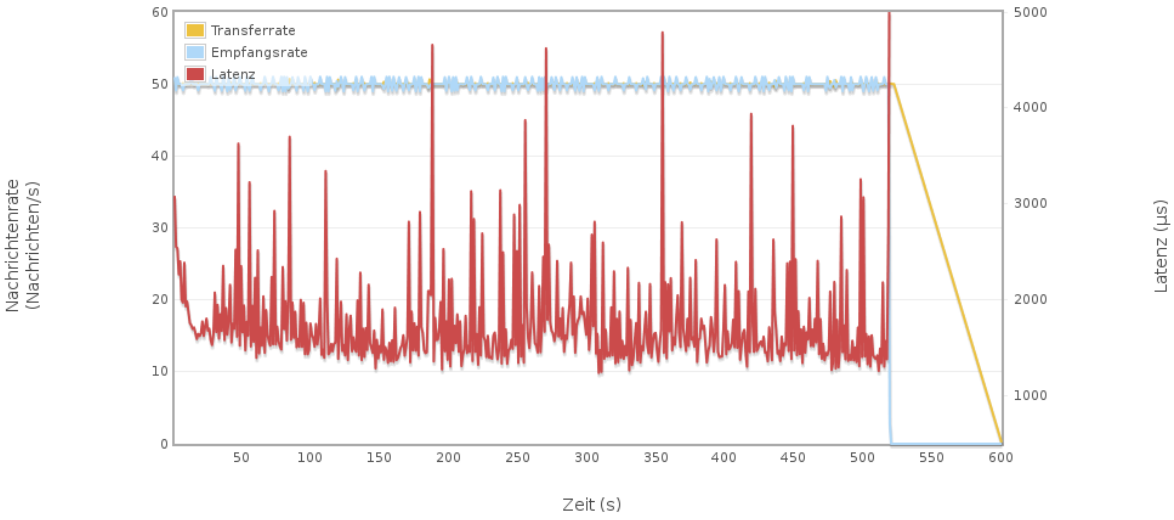

Abbildung 15: Gebündeltes Abweisen von Nachrichten - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Queue-Churning**
Messwerte	CPU: 44% RAM: 1542MB (Verwendet) HDD: 784MB (Frei) NET: RCX 265kbit/s TRX 358kbit/s
Beobachtungen	.
Anmerkungen	.

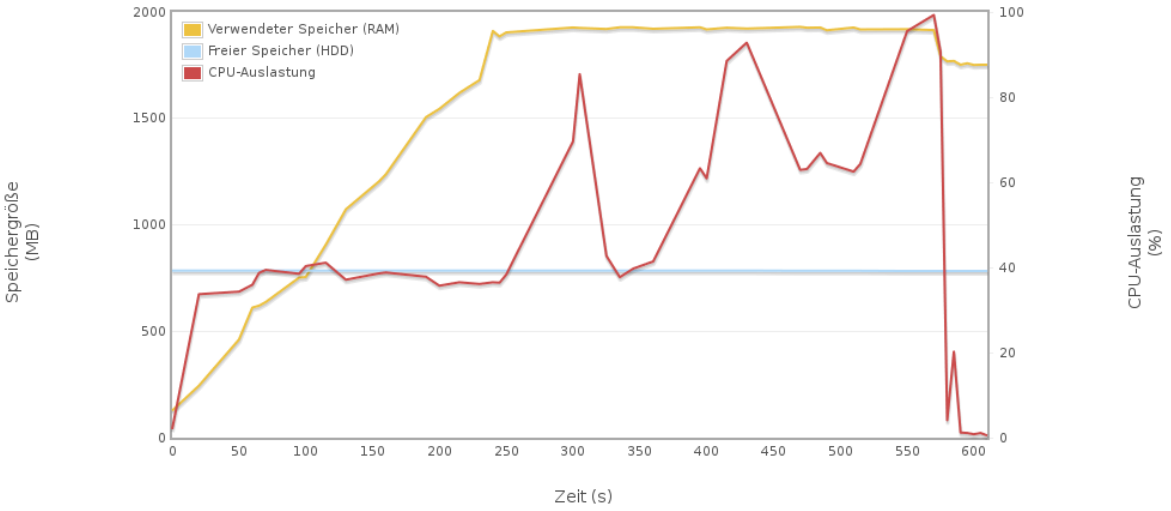

Abbildung 16: Queue-Churning - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

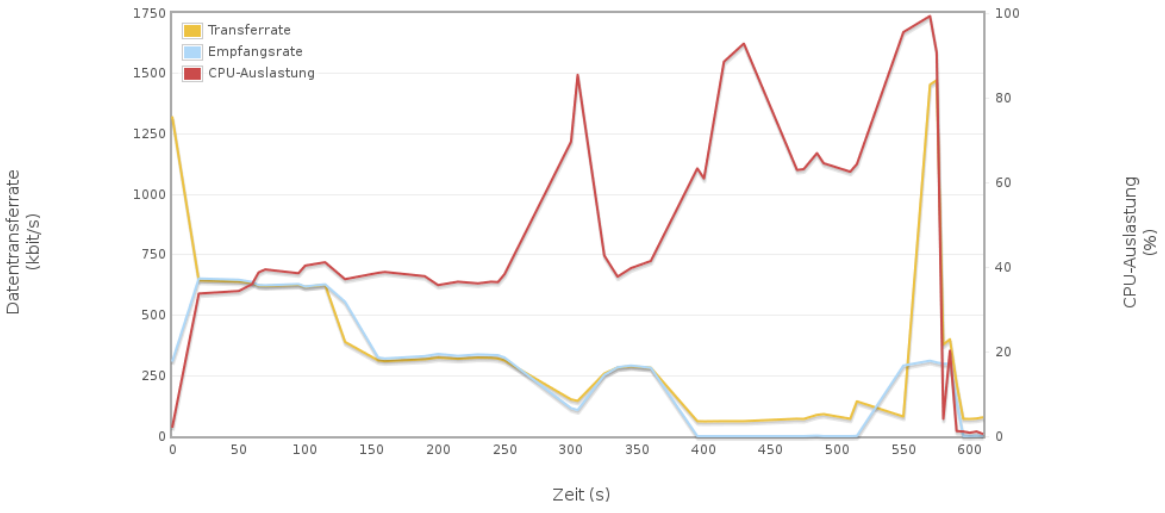

Abbildung 17: Queue-Churning - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

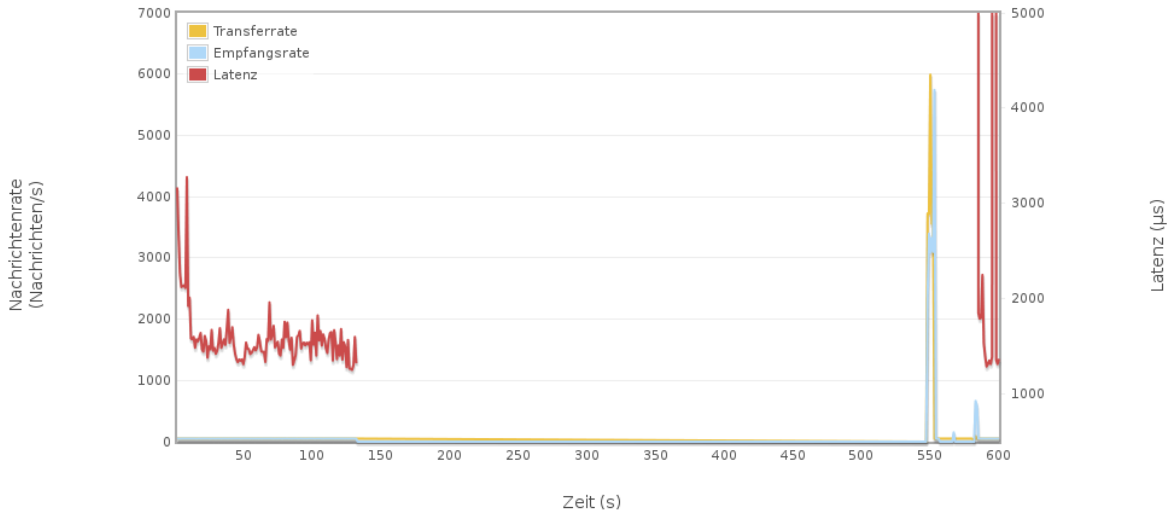

Abbildung 18: Queue-Churning - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Ausbleiben von Commits im Transaktionsmodus**
Messwerte	CPU: % RAM: MB (Verwendet) HDD: MB (Frei) NET: RCX bit/s TRX bit/s
Beobachtungen	Durch Auslassen des Commits steigt der Speicherverbrauch stark an. Nach wenigen Sekunden ist die Speichergrenze erreicht und keine weiteren Nachrichten werden übertragen. Daraufhin verharren alle Producer bis der Speicher wieder freigegeben wird (Falls Consumer vorhanden)
Anmerkungen	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

Angriff	**Nachrichten mit großem Header**
Messwerte	CPU: % RAM: MB (Verwendet) HDD: MB (Frei) NET: RCX bit/s TRX bit/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 das System stark ausgelastet und der Durchsatz der Nachrichten schrumpft stark auf 10 - 20 Nachrichten pro Sekunde. Es macht keinen Unterschied ob der Key oder Value vergrößert wird, die geringe Übertragungsrate bleibt bestehen. Der Value wird daher nicht ausgewertet. Die CPU Last liegt bei 10 -15%
Anmerkungen	Die Headergröße ist bei etwa 2500 Weiterleitungsoptionen begrenzt. Wird diese überschritten lässt das System die Verbindung fallen, aufgrund einer zu großen Framegröße. 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: % RAM: MB (Verwendet) HDD: MB (Frei) NET: RCX bit/s TRX bit/s
Beobachtungen	Das System ist stark ausgelastet. Ähnelt aber der Auslastung unter der Erstellung mehrerer Verbindungen. Allerdings beansprucht der Aufbau der Channel extrem viel Zeit. Nach Aufbau aller Kanäle bricht die Übertragungsrate stark ein. Dabei liegt die CPU-Last bei etwa 45 - 55%
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

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)
7.000m/s	100	10	100
800m/s	100	10	1.000
100m/s	100	10	10.000

Tabelle 3: Mehrere Channel

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	1.4ms Latenz; 50Nachrichten/s
Anmerkungen	.

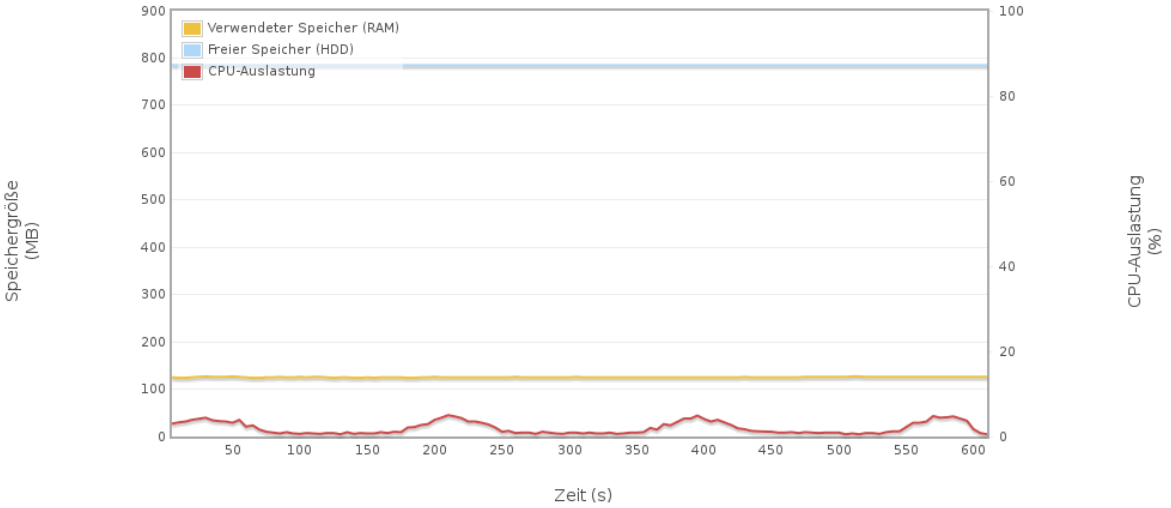

Abbildung 28: Handshake-Trickle - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

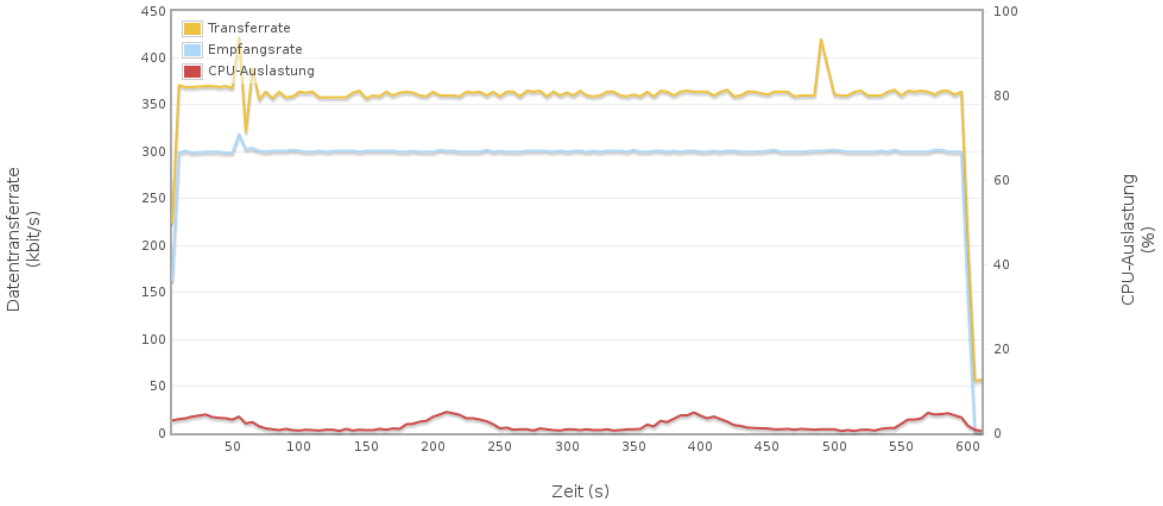

Abbildung 29: Handshake-Trickle - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

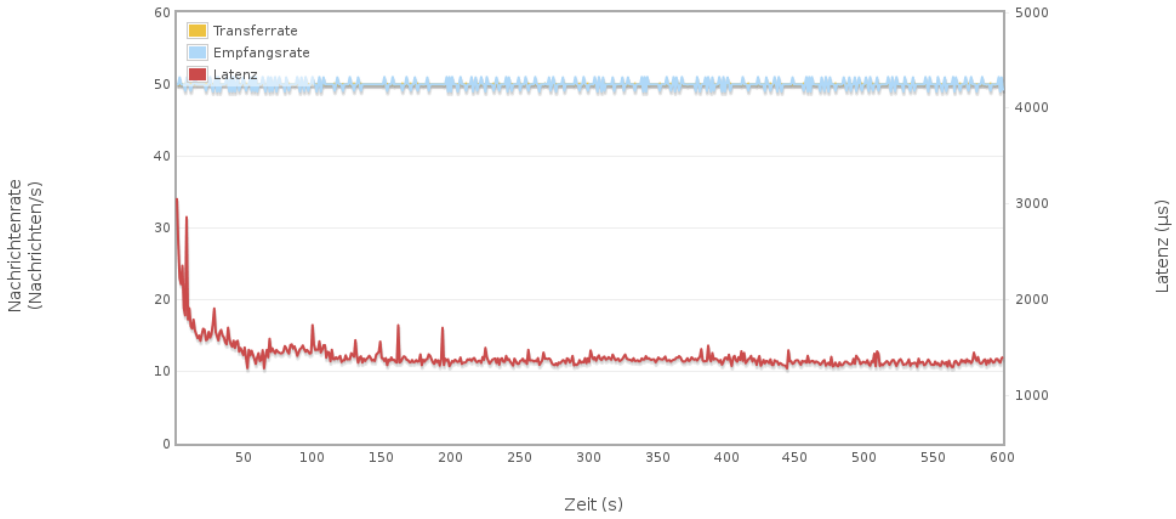

Abbildung 30: Handshake-Trickle - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**Heartbeat-Flooding**
Messwerte	CPU: 4% RAM: 134MB (Verwendet) HDD: 785MB (Frei) NET: RCX 318kbit/s TRX 386kbit/s
Beobachtungen	1.4ms, 50Nachrichten/s
Anmerkungen	.

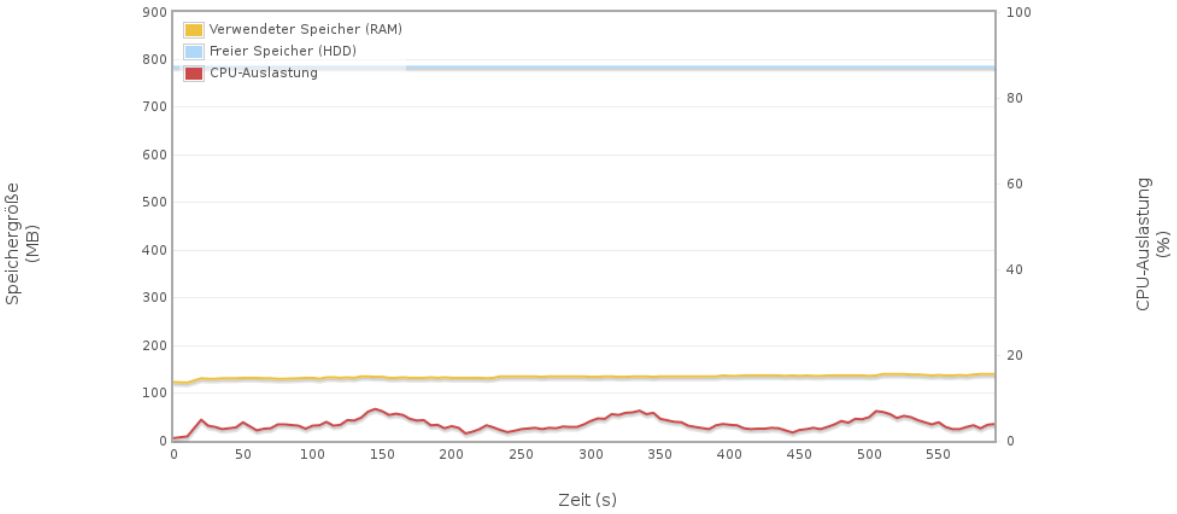

Abbildung 31: Heartbeat-Flooding - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

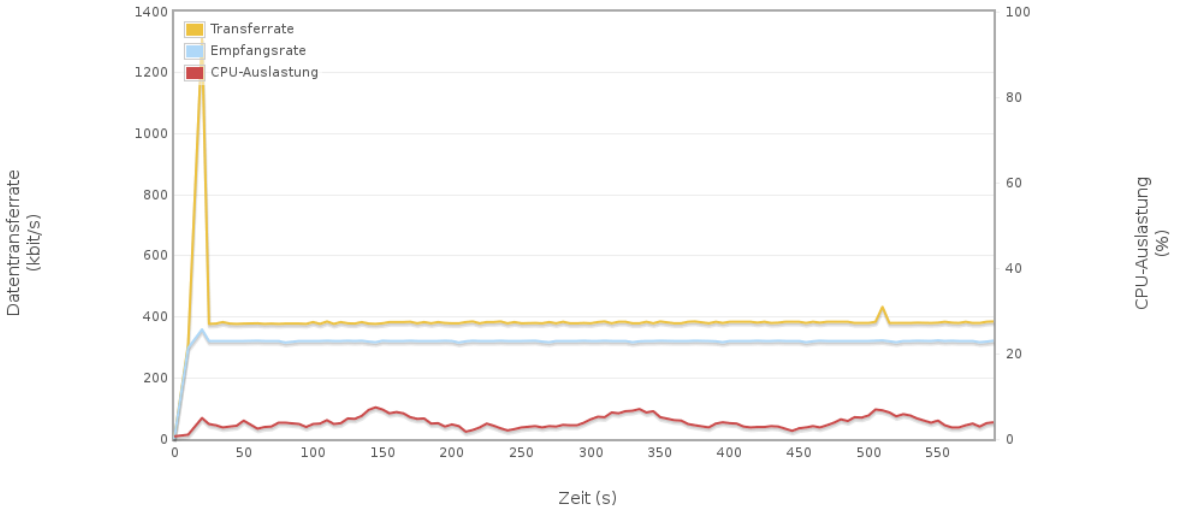

Abbildung 32: Heartbeat-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

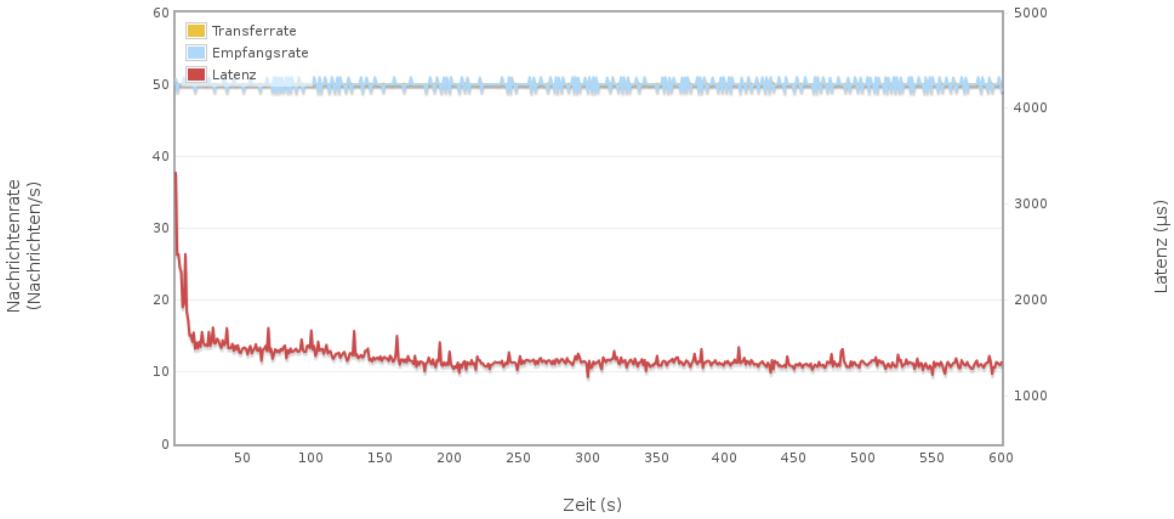

Abbildung 33: Heartbeat-Flooding - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Angriff	**TCP-Connection-Dropping**
Messwerte	CPU: 2% RAM: 416MB (Verwendet) HDD: 785MB (Frei) NET: RCX 303kbit/s TRX 522kbit/s
Beobachtungen	1.4ms, 50Nachrichten/s
Anmerkungen	.

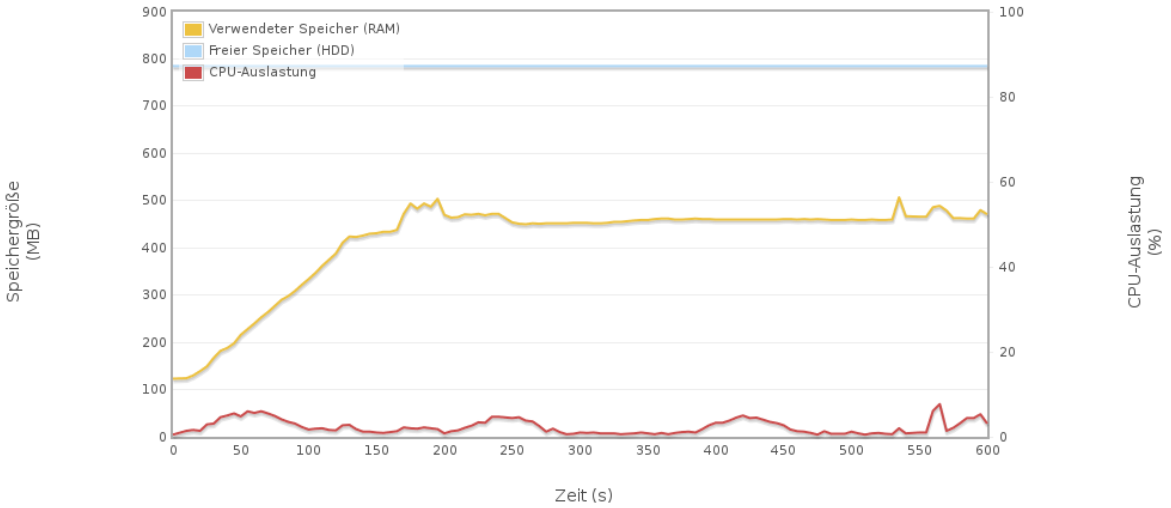

Abbildung 34: TCP-Connection-Dropping - Verlauf des Speicherbedarfs für RAM/HDD und Verlauf der CPU-Last auf dem RabbitMQ-Server

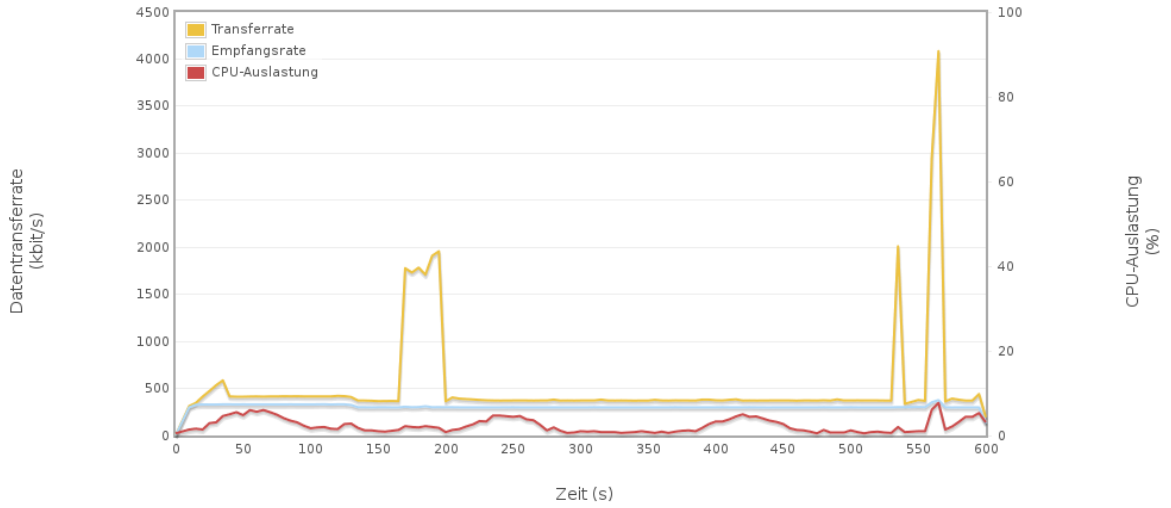

Abbildung 35: TCP-Connection-Dropping - Verlauf der Transfer-, Empfangsrate und Verlauf der CPU-Last auf dem RabbitMQ-Server

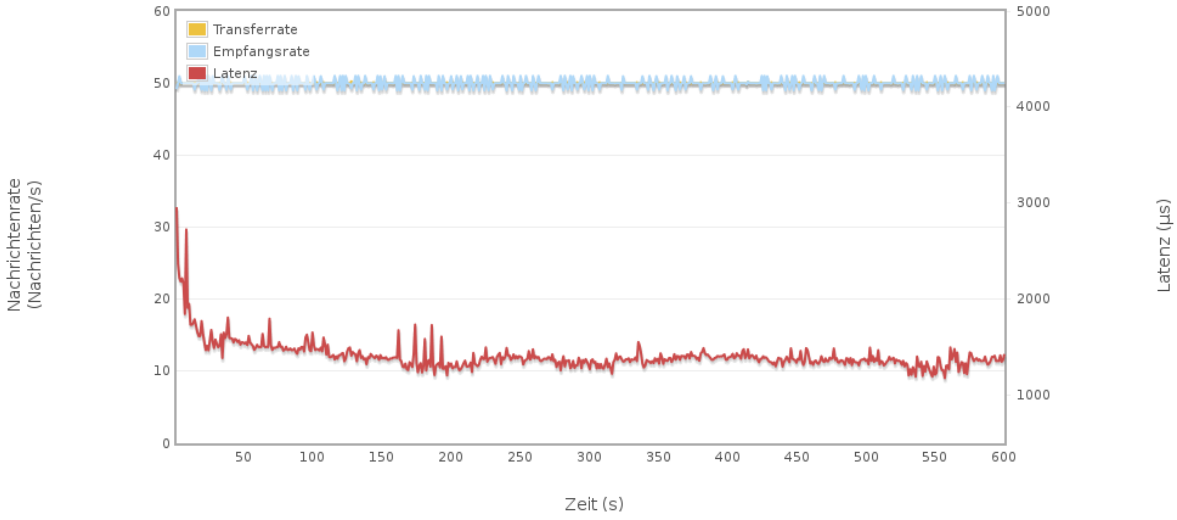

Abbildung 36: TCP-Connection-Dropping - Verlauf der Transfer-, Empfangsrate und Verlauf der Latenz im Anwendungsszenario

Zusammenfassung und Fazit