

Web Latenz im Transmission Control Protokoll

Dane Leube

Zusammenfassung

Diese Seminararbeit behandelt das Thema Web Latenz. Die Grundlage ist eine Analyse von Google, wonach die Web Latenz sehr stark von Paketverlusten während einer Übertragung beeinflusst wird. Diese Verluste führen bei TCP zu einer Erhöhung der Web Latenz, da die Pakete erneut gesendet werden. Diese Retransmission führt zu einer sehr großen Latenz wie eine Studie herausgefunden hat. In dieser Arbeit werden drei Lösungsansätze vorgestellt, die versuchen die Latenz zu reduzieren. Hierzu setzen wir uns das idealistische Ziel einer Paket-Recovery innerhalb von einem RTO.

1 Einleitung mit Problemstellung

Verzögerungen im Web kosten den Betreiber laut einer Studie von Amazon pro Verzögerung von 100ms ca. ein Prozent des Profits ¹. Somit ist das Thema Web Latenz ein sehr aktuelles Thema, womit sich jeder Anbieter eines Online-Shops auseinander setzen muss. Der Nutzer erwartet von einem Webshop, dass er schnell verfügbar ist und vor allem, dass er keine langen Wartezeiten hat, wenn er beispielsweise eine Abfrage an den Server schickt. Der einfachste Ansatz der Provider ist die Erhöhung der Backbone Hardware und der POP's [TFGo13]

Dieser Ansatz skaliert nicht endlos, da es weitere Engpässe gibt. Ein weiterer Lösungsansatz soll sich durch die kritische Analyse des TCP Protokolls ergeben. Das TCP Protokoll ist der defakto Standard bei verbindungsorientierten und zuverlässigen Verbindungen im Web. Bei der Spezifizierung dieses Protokolls wurden solche große Durchflussraten, wie sie heutzutage zustande kommen nicht berücksichtigt. Bei einer Untersuchung wurden große Datenmengen mittels TCP Verbindung über das Web transferiert und ausgewertet. Die Auswertung zeigt, dass bei mindestens 10 Prozent der Übertragungen ein Paket Verlust passiert. Weiterhin wurde festgestellt, dass bei Übertragungen mit mindestens einem Verlust bis zu fünf Mal länger bei der Übertragung benötigt wird als welche ohne Verlust. Dieses Resultat zeigt, wie wichtig es ist, dass die Pakete bei einer Verbindung verlustfrei übertragen werden, damit die Web Latenz verringert werden kann. Der Faktor fünf ist sehr hoch, wenn man bedenkt, dass es sich nur um ein einzelnes Paket handelt welches verloren geht und somit neu übertragen werden muss. Die Messung hat weiterhin herausgefunden, dass 77 Prozent dieser Verluste immer am Ende einer Verbindung auftreten, sogenannte Tail-Losses.

Die Ergebnisse der Messung zeigen die Wichtigkeit von verlustfreien TCP Übertragungen, damit die Web Latenz reduziert wird. Wenn doch ein Paket verloren geht, dann ist eine schnelle Recovery dieses Paketes notwendig. An dieser Stelle setzt diese Arbeit an. Das Ziel für die Optimierung ist bei einem Paketverlust die Retransmission so schnell wie möglich durchzuführen und so für eine geringere Latenz zu sorgen.

Wenn ein Paket verloren geht soll dieses so schnell wie möglich erneut gesendet werden. Das idealisierte Ziel ist eine Retransmission bevor der RTO Timer ausläuft. Die drei Ideen, die dieses Ziel erreichen sollen, haben unterschiedliche Ansätze. Die drei Ansätze sind aus

¹Quelle: <http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt>, 2006

dem Paper *Reducing Web Latency: the Virtue of Gentle Aggression* entnommen. Die Ansätze Reactive, Corrective und Proactive werden nach einem kurzen Grundlagenkapitel genauer erläutert und anschließend gegeneinander abgewägt.

2 Grundlagen

In diesem Abschnitt geht es um eine kurze Einführung in die TCP Kommunikation, damit die Problemstellung klarer herausgearbeitet werden kann. Weiterhin wird eine kurze Einführung in die Motivation von fehlerkorrigierenden Codes gegeben, um die spätere Lösung besser verstehen zu können.

2.1 Transmission Control Protokoll

Das Transmission Control Protokoll (im folgenden nur noch TCP genannt) ist ein verbindungsorientiertes Transportprotokoll. Es sitzt in der OSI-Layer Architektur auf dem 4. Layer, dem Transportlayer. Die wichtigste Eigenschaft von TCP ist die Zuverlässigkeit. Zur Erfüllung dieser Eigenschaft werden TCP Pakete, die auf dem Weg zum Empfänger verloren gehen erneut gesendet. Die zweite Eigenschaft, die Verbindungsorientierung von TCP baut eine Verbindung zwischen beiden Kommunikationspartnern auf, die ein Senden von Paketen in beide Richtungen zulässt. Im Folgenden soll der Ablauf einer TCP Verbindung dargestellt werden und die einzelnen Phasen erläutert werden, damit das Problem der Web Latenz im TCP Protokoll genauer erklärt werden kann.

Bei TCP erfolgt ein Verbindungsaufbau über einen 3-Way-Handshake. Bei jedem Paket wird der TCP Header als Offset hinzugefügt. Der komplette Header von TCP ist in Abbildung 1 zu sehen. Für diese Arbeit relevant ist die sogenannte Sequenznummer, die beiden Flags ACK und SYN. Beim Start einer neuen Verbindung sendet der Sender ein Paket mit einem gesetzten SYN Flag und einer zufällig generierten Sequenznummer. Diese dient als ID um eine eindeutigen Identifizierung des Paketes zu ermöglichen und ist die Basis zum Erkennen eines verloren gegangenen Paketes. Sie wird immer fortlaufend pro neuem Paket erhöht und bei jeder Übertragung mitgeschickt. Zur Bestätigung des erhaltenen Paketes antwortet der Empfänger mit einem gesetzten ACK und SYN Flag. Im ACK-Paket ist die um eins inkrementierte Sequenznummer des ersten Kommunikationsteilnehmers enthalten. Für den 3-Way-Handshake wird eine erneute Bestätigung vom Sender verschickt. Die ACK ist die um eins inkrementierte Sequenznummer des vorherigen Paketes vom Empfänger. Nachdem das ACK beim Empfänger angekommen ist, beginnt die Übertragung der Nutzdaten. Somit ist der Verbindungsaufbau von TCP abgeschlossen. Die Nutzdaten können in Pakete segmentiert werden und übertragen werden. Wichtig hierbei ist der sogenannte Slow-Start. Damit es nicht zu Stauungen und überflüssigen Paketverlusten kommt, wird bei einer Datenübertragung nie sofort das komplette Fenster übertragen. Es wird sich langsam an das Maximum der Leitung angenähert. Dieser langsame Aufbau hat den Vorteil, dass die Leitungen weniger überlastet und die Verbindung sich einpendeln kann. Dieses einpendeln bedeutet, dass die Retransmission Timeouts angepasst werden. Diese Timeouts sind für die vorliegende Arbeit von Bedeutung. Vor allem interessant bei kleinen Nutzdaten. Wenn hier zu große Timeouts gesetzt sind, dann ist die Latenz bei einem Paketverlust dementsprechend hoch.

Die Zuverlässigkeitseigenschaft von TCP ist für unsere Lösungsansätze von einer sehr großen Bedeutung. Die Zuverlässigkeit wird in TCP umgesetzt, indem eine erneute Übertragung des Paketes durchgeführt wird, sobald ein Paket auf dem Weg zum Empfänger verloren geht. Anhand der Sequenznummer kann identifiziert werden, wenn ein Paket fehlt. Der zweite Mechanismus zum Erkennen ist ein Timer auf der Senderseite, der startet, sobald das Paket

versendet wurde. Wenn der Timer abgelaufen ist und kein ACK Paket vom Empfänger erhalten wurde, wird das Paket erneut gesendet. Damit wir die Web Latenz reduzieren ist eine Retransmission des verloren gegangenen Paketes bevor das Timeout abgelaufen ist das Ziel.

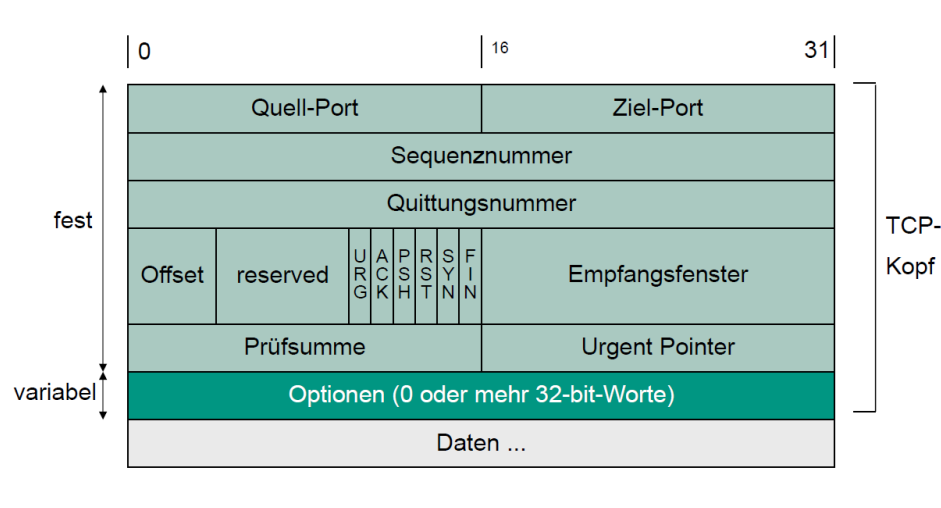


Abbildung 1: Aufbau des TCP Headers Quelle:[TELE]

2.2 Fehlerkorrigierende Codes

Damit der spätere Lösungsansatz besser verstanden werden kann, soll hier ein kurzer Überblick über sogenannte fehlerkorrigierende Codes gegeben werden. Zuerst soll geklärt werden, was ein Fehler in diesem Kontext bedeutet. Die Grundidee ist der Austausch von Informationen. Ein Sender A möchte zu einem Empfänger B eine Nachricht senden. Diese Nachricht enthält eine gezielte Information. Diese Information kann vollständig ankommen und es gibt keine Komplikationen. Problematisch wird es erst, wenn die Informationen unvollständig oder verfälscht ankommt. An dieser Stelle setzen die fehlerkorrigierenden Codes an. Hier wird vom Ablauf her eine Zusatzinformation zu der ursprünglichen Information hinzugefügt. Diese Information soll erstens dabei helfen zu erkennen, dass die Nachricht fehlerhaft ist und im zweiten Schritt die Nachricht korrigieren. Wenn es einen Ansatz gibt, der die Nachricht korrigieren kann, dann sprechen wir von fehlerkorrigierenden Codes. Die einfachste Form eines solchen Codes ist eine XOR-Verknüpfung. Mithilfe des XOR-Operators lassen sich Informationen aus zwei Quellen verifizieren. Dies muss man sich folgendermaßen vorstellen. Man nimmt eine Information A und verknüpft diese mit einem Code B. Die Verknüpfung ist eine XOR Verbindung. Das Ergebnis C ist unser fehlerkorrigierender Code. Mithilfe dieses Codes kann durch eine erneute XOR Verknüpfung von C und B auf A zurück geschlossen werden. Dieser Ansatz löst das Problem, wenn Informationen auf dem Weg der Übertragung verloren oder manipuliert werden. In dieser Arbeit ist besonders der Informationsverlust interessant und weniger die Manipulation von Daten auf dem Übertragungsweg. Diese Grundlage hilft uns den späteren *Corrective* Ansatz besser zu verstehen.

3 Problemanalyse

Nachdem die TCP Mechanismen erklärt wurden, ist die Frage nach der konkreten Problemstellung zum Thema Web Latenz zu klären. Grundlage der Analyse ist eine Messung, wonach die Web Latenz deutlich ansteigt, je mehr Pakete auf dem Weg zum Empfänger verloren gehen. Interessant an der Studie ist zum Einen die Menge der Daten, die erhoben wurde und zum

Anderen die Erkenntnis, dass bei vielen Paketverlusten ein großer Anstieg der Web Latenz zu beobachten ist. In Abbildung 2 ist ein Balkendiagramm dargestellt, was diesen Sachverhalt verdeutlicht. Man sieht, dass bereits bei einem Paketverlust die Weblatenz um den Faktor zehn zunimmt. Weiterhin ist interessant zu sehen, dass man sehr nahe an der idealen RTT ist, wenn man keinen Paketverlust hat. Dadurch, dass bei unzuverlässigen Protokollen wie UDP keine erneute Übertragung der Pakete stattfindet, begrenzt sich das Problem mit den Retransmissions auf Protokolle mit der Zuverlässigkeitseigenschaft. Für unseren Ansatz ist das TCP Protokoll interessant, welches im vorigen Abschnitt erläutert wurde. Um die Web Latenz damit die Web Latenz verringert werden kann ist eine Optimierung beim Retransmission Algorithmus von TCP zu suchen.

Eine genauere Analyse zeigt, dass die teuren Verluste sogenannte Tail-Losses sind. Warum ist dies so? Dadurch, dass wir nach dem letzten TCP-Datenpaket, welches wir übertragen kein weiteres Datenpaket folgt, entsteht eine Lücke in der Übertragung. Wenn an dieser Stelle das ACK Paket verloren geht, so wird der Retransmission Timer mit einer hohen Wahrscheinlichkeit ablaufen. Dieses Problem wollen wir im folgenden auch weiterhin untersuchen und eine passende Lösung dazu finden. Wir werden somit genauer auf die End-Phase einer Übertragung eingehen, da hier weitere Problemstellungen, wie eben keine Folgepakete zu beachten sind. Diese Tail-Losses machen einen sehr großen Prozentanteil bei der Web-Latenz aus.

Ein weiterer Aspekt, der beachtet werden muss sind in kleinere Datenmengen. Wenn geringe Datenmengen übertragen werden, dann sind die oben aufgezeigten Tail-Loss Probleme sehr gravierend. Am Gravierendsten sind die Probleme, wenn es nur ein Datenpaket gibt. Wenn also direkt nach dem Verbindungsaufbau ein Datenpaket folgt und dann die Verbindung beendet wird. Da am Anfang einer Übertragung noch sehr hohe RTO-Zeiten gelten, wird hierdurch die Web Latenz sehr groß, wenn ein Paketverlust stattfindet. Wenn die Übertragung länger dauert, dann haben sich die Timer sehr gut eingespielt und die Verbindung ist sehr stabil, wodurch weniger Paketverluste auftreten. Somit entstehen die teuren Paketverluste welche eine hohe Web Latenz verursachen bei kleinen Datenmengen und wenn die Daten am Ende einer Übertragung verloren gehen. Bei der Lösungsfindung muss somit verstärkt auf die Optimierung des RTO und der Tail-Losses eingegangen werden.

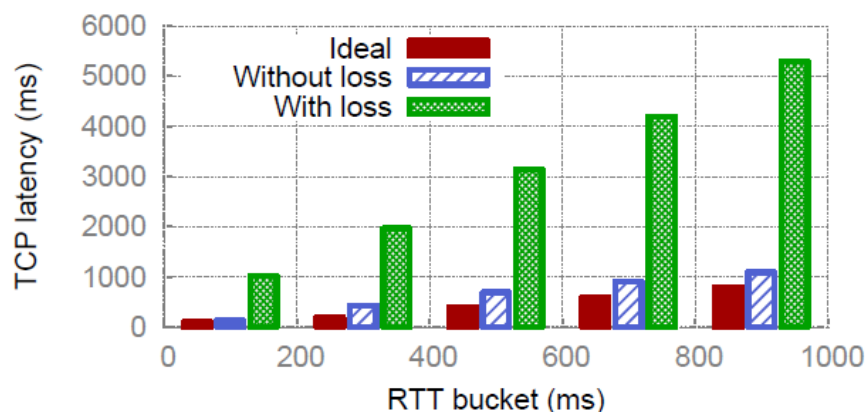


Abbildung 2: Googles Auswertung der Paketverluste und deren Konsequenz für die Web Latenz:[TFGo13]

4 Zielsetzung - Zieldefinition Recovery innerhalb einer RTT

Das Ziel, worauf wir in dieser Arbeit hinarbeiten wollen soll sich am Ideal von einer Paketrecovery innerhalb einer RTT orientieren. Dies bedeutet, bei einem Paketverlust zu reagieren, bevor das Retransmission Timeout erreicht ist. Damit das Problem nicht zu einseitig betrachtet wird, müssen wir in der Praxis weitere Aspekte als die Zeit der Übertragung betrachten. Ein wichtiger Faktor ist die Staukontrolle. Wenn ein Paket auf einem Pfad in einen Stau gerät, so ist die Wahrscheinlichkeit, dass bei einer erneuten Übertragung das Paket wieder verloren geht sehr hoch. Die Gründe hierfür liegen in einem viel genutzten Pfad oder gar einem Flaschenhals im Netz. Aus diesem Grund ist das Ziel eine Retransmission innerhalb eines RTT mit Vorsicht zu betrachten und als Ideal zu sehen. In dieser Arbeit werden die Lösungen vorgestellt, die diesem Ideal nahe kommen sollen. Nach der Vorstellung sollen die drei Lösungsansätze diskutiert werden und zum Abschluss eine Empfehlung abgegeben werden.

5 Lösungsvarianten

Nachdem unser Ziel definiert wurde, kümmern wir uns um die Wege, wie wir das Ziel Paket-Recovery innerhalb einer RTT erreichen wollen. Hierzu gibt es im Paper [TFGo13] die drei Ansätze *Reactive*, *Corrective* und *Proactive*. Alle drei Ansätze nutzen als Basis die Redundanz und wurden auch implementiert. Damit eine schnelle Recovery erfolgen kann, werden redundante Informationen zu der eigentlichen Information hinzugefügt. Diese Redundanz soll vermeiden, dass auf der Senderseite ein Paketverlust erkannt wird, bevor das RTO erreicht ist. Der *Reactive* Ansatz beugt dem Problem des nicht erhaltenen ACKs vor indem es mehrere kleinere Zusatzpakete zum Empfänger schickt. Somit ist die Wahrscheinlichkeit höher, dass der Empfänger eines dieser Pakete erhält und damit bestätigen kann. Der *Corrective* Ansatz versucht den Sender und Empfänger einzubinden, so dass mit nachzukorrigierenden Codes gearbeitet wird. Die *Proactive* Lösung geht den aggressivsten Weg und versendet zu 100 Prozent redundant seine Daten. Wie die drei Ansätze im Detail funktionieren, wird in den folgenden Abschnitten behandelt, damit deren Vor- und Nachteile im Folgenden diskutiert werden können.

5.1 Reactive

Beim Reactive Ansatz wird auf der Sender Seite bei einer TCP Übertragung Änderungen vorgenommen. Die Änderung besteht aus dem Senden von mehreren Probe Paketen für ein Datenpaket wie in der Abbildung 3 dargestellt. Ein solches Probe Paket ist ein kleineres Datenpaket welches beim Empfänger das Senden von mehreren ACK Paketen provozieren soll. Wir senden ein großes Datenpaket und dazu viele kleinere Probe Pakete. Der Reactive Ansatz setzt am Problem mit den Tail-Losses an. Dieses Problem wird mit dem Ansatz gelöst in dem es zum Ende der Übertragung noch weitere Feedbackmöglichkeiten für den Empfänger gibt. Dies bedeutet, dass alleine durch kleine Redundanz Pakete von der Sender Seite dem Empfänger mehrfach die Möglichkeit gegeben wird auf diese zu reagieren und ein ACK zu schicken. Somit wird ein ACK Flooding als Lösungsansatz provoziert. Dies bedeutet man beugt dem Problem vor, wenn ein ACK Paket auf dem Weg zurück verloren gehen sollte. Wenn ein Datenpaket auf dem Weg zum Empfänger verloren gehen sollte, dann reduziert dieser Ansatz nicht die Web Latenz. Der Lösungsansatz funktioniert nur, wenn das Paket wirklich ankommt.

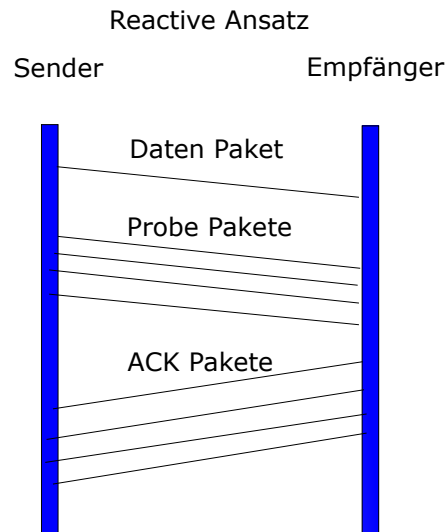


Abbildung 3: Abstrakter Ablauf des Reactive Ansatzes

5.2 Corrective

Im zweiten Ansatz muss eine Änderung von Sender und Empfänger erfolgen. Der sogenannte Corrective Ansatz verwendet die Redundanz um aus den Zusatzinformationen auf die ursprüngliche Information zu schließen. Der Sender fügt der ursprünglichen Nachricht, die er versenden möchte eine Zusatzinformation hinzu. Den sogenannten fehlerkorrigierenden Code. Dieser Code hilft dem Empfänger dabei bei Informationsverlust auf dem Weg der Übertragung auf die ursprüngliche Information zurückzuschließen. Dieser Ansatz erfordert Anpassungen beim Sender und Empfänger. Der Sender muss die Zusatzinformation bereitstellen und verschicken, so wie in Abbildung 4 dargestellt. Der Empfänger verwendet diese bei Paketverlust um die ursprüngliche Information wieder herzustellen. Wenn wir diesen Ansatz auf unser Ursprungsproblem anwenden und die Verluste bei kleinen Datenpaketen betrachten, so fällt uns auf, dass wir eine gute Recovery bei kleinen Datenpaketen erreichen. Diese gute Recovery erfolgt daraus, dass keine erneute Übertragung des Paketes von seitens des Senders stattfinden muss. Voraussetzung, dass dieser Ansatz funktioniert ist eine ausreichende Größe des fehlerkorrigierenden Codes. Es macht keinen Sinn den Code so groß zu wählen, wie die ursprüngliche Information, da es sonst zu einer kompletten Redundanz kommt. Vielmehr macht es mehr Sinn, wenn wir eine Eingabegröße A auf eine kleinere Ausgabemenge B abbilden. Um konkreter zu werden eine Information mit einer Länge von 2^8 Bit auf eine Information von 2^4 Bit abzubilden. Hier gehen natürlich Informationen verloren. Aus diesem Grund müssen sinnvolle Reduktionsfunktionen gewählt werden. Die einfachste Lösung, wie bereits diskutiert ist eine XOR Lösung. Hierzu teilen wir die Information, die wir versenden wollen in zwei Teile auf und verknüpfen diese mit XOR. Das Ergebnis dieser Verknüpfung ist dann unser fehlerkorrigierender Code. Dieser wird als zusätzliche Redundanz dem Paket beigelegt. Auf der Empfänger Seite lassen sich dann Rückschlüsse darauf führen, ob das Paket korrekt ist.

Ein Implementierungsversuch, wie so ein Code aussehen kann, wird im Paper [TFGo13] erläutert. Es beginnt mit dem Aushandeln, dass so ein fehlerkorrigierender Code bei der Übertragung verwendet wird. Der Sender und Empfänger einigen sich während des initialen Handshakes auf ein Verfahren. Es werden anschließend mehrere Sequenzen zusammen gruppiert

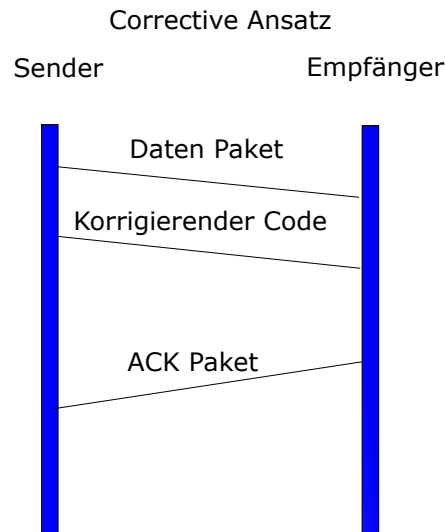


Abbildung 4: Abstrakter Ablauf des Corrective Ansatzes

und darauf eine Checksumme gebildet. Diese Checksumme kann eine XOR Verknüpfung sein, wie oben erläutert. Die erzeugte Checksumme wird als zusätzliches Paket hinzugefügt und versendet. Auf der Empfängerseite wird mithilfe der Checksumme verifiziert und kann bei Verlust auf die Informationen zurück schließen. Wie bereits erwähnt kann bei diesen korrigierenden Codes nicht auf die komplette Information zurück geschlossen werden, sonst könnte man auch einfach das Paket 100 prozentig redundant senden. Aus diesem Grund gehen immer Informationen verloren. Dies bedeutet bei unserem Ansatz, dass wenn man die Informationen nicht nachstellen kann, dann hat die Empfängerseite zumindest die Möglichkeit schnell ein ACK zu versenden oder eben nicht, um auf der Senderseite ein RTO zu provozieren. Dieser Ansatz behandelt somit das Problem, wenn ein Tail-Loss vorkommt und dies auf der Empfängerseite bemerkt wird. Der Handlungsfokus liegt somit auf der Empfängerseite. Diese ist für eine schnelle Sendung des ACK Paketes verantwortlich um eine schnelle Retransmission zu forcieren. Weiterhin kann der Empfänger bei wenig Informationsverlust selbst die Information wiedergewinnen.

5.3 Proactive

Der dritte Lösungsansatz ist der aggressivste der Drei. Aggressiv bedeutet in unserem Kontext eine maximale Redundanz. Einfach gesagt versendet der Proaktive Ansatz die Daten zu 100 Prozent redundant. Ein Datenpaket wird also einfach zweimal versendet. Im Idealfall gibt es somit immer eines der beiden Pakete an. Abbildung 5 veranschaulicht den Vorgang. Zu einem Datenpaket wird ein zweites zusätzliches Paket hinzugefügt und ebenfalls versendet. Von der Empfängerseite werden somit zwei ACK Pakete versendet, wenn beide Pakete ankommen sollten. Der Problemfall ist, wenn ein Paket verschollen geht, dann ist die Wahrscheinlichkeit sehr hoch, dass das zweite Paket ebenfalls nicht ankommt sehr hoch. Dies ist eine einfache Wahrscheinlichkeitsrechnung. Wenn ein Paket in einer Stauung feststeckt und nicht zu der vorgegeben Zeit beim Empfänger ankommt, dann hat das redundante Paket keinen Sinn. Jedoch muss man das Problem weiter fächern. Ein Paket kann auch fehlerhaft ankommen. Wenn die Information also unvollständig in Paket eins ankommt und dann in Paket

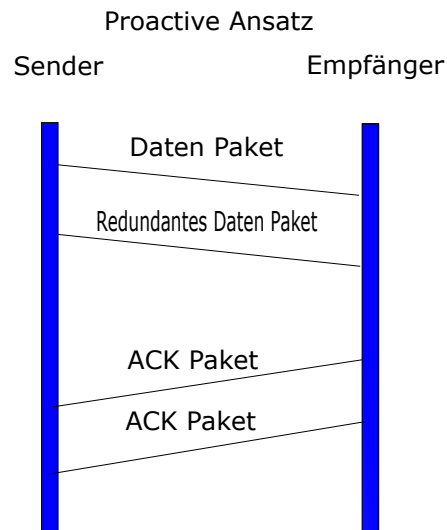


Abbildung 5: Abstrakter Ablauf des Proactive Ansatzes

zwei diese Information erhalten ist, so kann der Empfänger ohne eine erneute Retransmission des Senders auf die Information zugreifen.

Weiterhin hilft dieser Ansatz auch bei der Kommunikation zwischen den beiden Kommunikationspartnern. Es kann somit genauer festgestellt werden, welches Paket fehlerhaft und somit erneut zu übertragen ist. Dieser Ansatz löst unserer Tail-Loss Problem nur bedingt. Das RTO läuft trotzdem aus, wenn ein fehlerhaftes Paket versendet wird. Um die Latenz zu reduzieren hilft der Ansatz bei einer schnelleren Erkennung von Fehlern. Dies führt zu einem schnelleren Senden eines ACK Paketes und kann somit zu einer schnelleren Recovery führen. Die wichtigste Frage bei diesem Ansatz ist, ob der zusätzliche Overhead gerechtfertigt ist, da dass Paket ja doppelt versendet wird. Bei kleinen Paketen kann dies sinnvoll sein, da die Leitung nicht stark belastet wird. Bei großen Paketen jedoch ist dieser Ansatz nicht praktikabel. Da wir in dieser Arbeit uns auf die kleineren Datenpakete konzentrieren lohnt es sich, diesen Ansatz weiter zu verfolgen. Der große Nachteil dieser Lösung ist ein höheres Datenvolumen bei der Übertragung und Probleme bei der Staukontroll-Mechanismen von TCP.

6 Diskussion der Lösungsansätze

Nachdem die Lösungsansätze vorgestellt und evaluiert wurden, kann im nächsten Schritt eine genaue Abwägung der einzelnen Ansätze erfolgen. Der Grundgedanke für eine schnellere Recovery ist eine zusätzliche Redundanz beim Versenden der Informationen. Diese Redundanz hat als Ziel das schnelle Erkennen eines Paketverlustes und darauf aufbauend eine schnelle Retransmission des Paketes vom Sender. Hierbei gibt es verschiedene Aspekte, die man beachten muss. Zum einen sind immer Anpassungen an das ursprüngliche TCP Protokoll notwendig. Die Anpassungen können Sender- und Empfänger-seitig sein. Der Reactive Ansatz beschränkt sich auf Anpassungen auf der Senderseite. Hier werden die zusätzlichen Probe Pakete generiert, die den Empfänger zum Senden von zusätzlichen ACK Paketen zwingen sollen. Die Anpassungen sind sehr gering, so dass die Middleboxen nicht angepasst werden müssen. Dies ist ein großer Vorteil dieses Ansatzes. Weiterhin müssen wenige zusätzliche Informationen

versendet werden, was die Leitung entlasten kann. Jedoch wird das Problem, wenn das Datenpaket nicht ankommt nicht angegangen. Es wird lediglich umrundet. Gesetzt der Fall, dass nur die Probe Pakete zum Empfänger ankommen und das Datenpaket nicht. So gibt es keine Chance für den Sender vor dem RTO zu reagieren. Aus diesem Grund ist dieser Ansatz nur effektiv, wenn kein Paket zum Empfänger durchkommt, da dann ein schnelles reagieren des Senders stattfinden kann.

Der Corrective Ansatz geht hier einen Schritt weiter. Mithilfe dieser Lösung kann eine erneute Retransmission bei einem kleinen Paketverlust vermieden werden. Dieser Ansatz hat mehr Redundanz als der Proactive Ansatz, jedoch bietet er auch eine zusätzliche Möglichkeit der Recovery auf der Empfängerseite. Ein sehr großer Nachteil ist die Anpassung der Sender und Empfängerseite. Beide Seiten müssen zuerst ein spezielles Verfahren aushandeln, welche Codes sie behandeln wollen und weiterhin muss auf beiden Seiten die Möglichkeit gegeben sein auf die Codes reagieren zu können. Ein großes Hindernis bei der Implementierung sind die Middleboxes. Da auf beiden Seiten (Empfänger und Sender) Anpassungen nötig sind, müssen die Middleboxes ebenfalls angepasst werden. Dies erfordert zusätzlichen Aufwand und gibt eventuell Probleme bei der Standardisierung. Die Vorteile, dass durch eine kleinere Redundanz auf die ursprüngliche Information geschlossen werden kann sind vor allem für unsere kleinen Datenpakete interessant. Weiterhin sind die Vorteile eines schnelleren Erkennens eines Fehlers und im Idealfall eine selbständige Recovery sehr gewichtig.

Der aggressivste Ansatz hat seine Stärken ganz klar auf der Empfängerseite. Sobald beide Informationspakete zumindest teilweise angekommen sind, ist eine erneute Retransmission hinfällig. Auch wenn das erste Paket verloren gehen sollte, so hat man immer noch das zweite Paket als Redundanz. Probleme ergeben sich erst dann, wenn beide Pakete verloren gehen. Dann gibt es auf der Empfängerseite keine Chance zu reagieren und das RTO fällt. Es ist der riskanteste Ansatz, allerdings bei einer stabilen Verbindung auch der Erfolgsversprechendste. Voraussetzung für diesen Ansatz ist kein Engpass in der Übertragung der Daten. Da die Leitungen mit diesem Ansatz stärker beansprucht werden als bspw. mit dem reaktiven Ansatz ist eine gute Leitung die Grundvoraussetzung. Die Vorteile dieses Ansatzes liegen ganz klar in der Redundanz. Da die doppelte Menge an Informationen übertragen werden, ist die Wahrscheinlichkeit, dass die Informationen auf der Empfängerseite vollständig ankommen deutlich erhöht.

Diese Lösung hat ihre Stärken, wenn Daten auf dem Weg zum Empfänger manipuliert werden und man die Informationen trotzdem gewinnen möchte. Zur Reduzierung der Web Latenz trägt dieser Ansatz bei, indem er versucht eine komplette Retransmission zu verhindern. Im Gegensatz dazu hat der Reactive Ansatz das Ziel einen Paketverlust schnell zu erkennen und eine erneute Übertragung schnell durchzuführen. Der Corrective Weg geht den Mittelweg und versucht beide Möglichkeiten zu vereinen. Hierzu wird weniger Redundanz als bei dem Proactive Ansatz verwendet bei einer gleichzeitig höheren Korrekturmöglichkeit als beim Reaktiven Ansatz.

7 Fazit

Alle drei Ansätze verbindet die Redundanz. Der Reactive Ansatz versucht mit Änderungen auf der Senderseite eine Verbesserung zu erreichen und hat die geringste Redundanz zu bieten. Dieser Ansatz funktioniert in seiner einfachen Implementierung. Das Problem bei diesem Ansatz ist die Staukontrolle, die ein Flooding der kleinen Probe Paketen verhindert und so eine Herausforderung bei der Implementierung darstellt. Mit etwas mehr Redundanz und Fehlerkorrigierenden Codes versucht es der Corrective Ansatz. Dieser verursacht auf der Sender und Empfängerseite Änderungen. Hier kann der Empfänger selbst reagieren und man kann

geringe Verluste bei der Übertragung selbst ausgleichen ohne eine erneute Retransmission zu benötigen. Schwächen hat dieses Verfahren, wenn das Paket aus den erhaltenen Informationen nicht wiederherzustellen ist. In diesem Fall läuft kann von der Empfängerseite zwar reagiert werden aber es erfolgt keine Verbesserung im Vergleich zum bereits implementierten TCP Verfahren. Mit der meisten Redundanz geht der Proactive Ansatz vor. Dieser verwendet hundertprozentige Redundanz um dafür zu sorgen, dass die Information auf der Empfängerseite in jedem Fall ankommt. Mit dieser Lösung gibt es Probleme bei der Staukontrolle von TCP. Diese muss überlistet werden um eine erfolgreiche Implementierung zu erhalten.

Die Ansätze lassen sich für die Zukunft auch kombinieren und aushandeln. Ein Corrective Ansatz ist für jede Übertragung sinnvoll, da ein verbesserter Mechanismus zur Erkennung von manipulierten Daten besteht. Für die Reduzierung der Web Latenz ist er nur nützlich, wenn die Informationen wirklich mithilfe des Codes zurückgewonnen werden können. Damit vor dem Ablauf des Retransmission Timouts reagiert werden kann, eignet sich der Reactive Ansatz. Hier erhält der Empfänger ein wirklich sehr schnelles Feedback, wenn ein Paket verloren geht. Somit ist eine Kombination aus Reactive und Corrective für eine Verbesserung der Web Latenz meiner Ansicht nach am Erfolgsversprechendsten.

Literatur

- [Laem] Philipp Laemmel. Der Retransmission Timeout von TCP.
- [TFGo13] Andreas Terzis Barath Raghavan Neal Cardwell Yuchung Cheng Ankur Jain Shuai Hao Ethan Katz-Bassett Tobias Flach, Nandita Dukkupati und Ramesh Govindan. Reducing Web Latency: the Virtue of Gentle Aggression. 2013.