TKN Telecommunication Networks Group

Praktikum
Rechnernetze und Verteilte Systeme

(letzter) Block 8

— ARQ-Verfahren —

Termin: 4.-8.2.2018 & 11.-15.2.2018

1 Theoretische Vorbereitungsaufgaben

Die folgenden Aufgaben sollen Ihnen helfen, sich auf den Vorbereitungstest vorzubereiten. Klären Sie bitte mögliche Fragen oder Unklarheiten unbedingt vor den ISIS-Testaten!

Aufgabe 1:
Angenommen Sie greifen auf das Internet über einen "Router" zu. Beantworten Sie die folgenden Fragen:

a) Sie schalten Ihren Rechner an und erhalten automatisch eine IP-Adresse. Die dafür benutzte Technologie nennt sich DHCP. Wie sieht die initiale Anfrage nach einer Adresse aus?

b) Sie haben eine Adresse zugewiesen bekommen. Wie lange ist diese gültig?

c) Was macht Ihr Rechner, damit die Adresse länger gültig bleibt? Wann tut er dies?

d) Ihr "Router" stellt das Internet über NAT bereit. Was bedeutet das?

Aufgabe 2:
Beantworten Sie im Kontext von IP die folgenden Fragen:

a) Was sind Class A, B und C Netze? Welcher Teil der Adresse gehört jeweils zu Host bzw. Netzwerk?

b) Überprüfen Sie, ob die IPv4-Adresse `149.77.115.54` im Netzwerk `149.77.112.0` mit der Netzwerkmaske `255.255.252.0` liegt.

c) Nennen Sie zwei Ansätze, mit dem Problem der knappen Internet-Adressen umzugehen.

d) Wie hilft das heute verwendete Classless Inter-Domain Routing (CIDR), das Problem zu lösen?

Send-and-Wait und Go-Back-N

In der Vorlesung haben Sie das Prinzip von Fehlerkontrollprotokollen bzw. ARQ-Protokollen kennengelernt. Dabei wurde Ihnen das Minimalbeispiel für ARQ gezeigt, bei dem ein Sender für jedes gesendete Paket eine Bestätigung vom Empfänger erwartet und das Paket erneut überträgt falls die Bestätigung innerhalb eines Timeouts nicht ankommt. Dieses Prinzip nennt man **Send-and-Wait**. Abhängig von der Verbindung kann Send-and-Wait aber sehr ineffizient sein. Stellen Sie sich folgendes Szenario vor: Sie senden mittels Send-and-Wait Pakete über eine Satellitenverbindung mit einer Bandbreite von 1 GBit/s und einer Round-trip-time von 100 ms. Die Pakete haben eine Größe von 1000 Bits. Ein Paket benötigt dann zwar nur 1 μs um abgeschickt zu werden, aber Sie erhalten erst nach $100.001ms$ die Bestätigung, bevor Sie das nächste Paket senden können. Sie haben also 99.999% der Zeit mit Warten verbracht. Die effektive Kanalausnutzung beträgt also 0.001% im Idealfall ohne Paketverlust.
Entscheidend ist das sogenannte **Bandbreite-Verzögerungsprodukt (Bandwidth-Delay-Product)**, d.h. wie viele Pakete übertragen werden könnten, bis die erste Bestätigung beim Sender ankommt. Ist dieses sehr hoch, so ist Send-and-Wait sehr ineffizient. In diesem Fall ist es eine bessere Strategie mehrere Pakete innerhalb eines Fensters (Sliding Window) zu versenden bevor man auf eine Bestätigung wartet. Diesen Ansatz verfolgt das **Go-Back-N**-Protokoll, welches Sie im Rahmen dieses Blocks implementieren sollen. Das Protokoll funktioniert wie folgt:

- Sender:
  - Verschicke alle Pakete deren Sequenznummer im Intervall $\left[n; n+w-1\right]$ liegen in Reihenfolge, wobei n die Sequenznummer des ersten noch unbestätigten Pakets ist und w die Größe des Fensters.
  - Nachdem ein Paket versendet wurde, starte jeweils einen Timeout.
  - Läuft der älteste Timer der noch unbestätigten Pakete aus, verschicke das gesamte aktuelle Fenster erneut.
  - Sobald eine Bestätigung mit Sequenznummer i ankommt, wobei $i > n$ setze $n = i$. Alle Pakete mit Sequenznummer $< i$ gelten damit als empfangen (sog. **kumulatives Acknowledgement**).
- Empfänger:
  - Sei e die Sequenznummer des nächsten erwarteten Pakets. Kommt ein fehlerfreies Paket mit Sequenznummer e an, erhöhe e um 1 und verschicke eine Bestätigung mit Sequenznummer e.
  - Für jedes ankommende Paket das nicht der erwarteten Sequenznummer entspricht, verschicke eine Bestätigung mit Sequenznummer e.

Bitte machen Sie sich mit Go-Back-N vertraut und stellen Sie ggf. Ihre Fragen im ersten Tutoriumstermin oder vorab im ISIS-Forum. Go-Back-N-spezifische Fragen werden nicht Teil des Tests sein aber ein gutes Verständnis wird Ihnen die Implementierung erheblich erleichtern.

2 Präsenzaufgaben

Die folgenden Aufgaben werden im Termin unter Anleitung des Tutors durchgeführt.

Aufgabe 3:
Nutzen Sie das verlinkte Onlinetool[1] um eine Simulationen von GoBackN durchzuführen. Nehmen Sie eine Fenstergröße von 5 Paketen an und spielen Sie verschiedene Szenarien durch (z.B. Verlust eines oder mehrerer Pakete und/oder Acknowledgements)

Interessant dürften extreme Fälle sein, z.B.:

- Das erste Paket kommt nicht an, aber die restlichen 4
- Alle 5 Pakete kommen an. Die ersten 4 ACKs gehen verloren, nur das 5. kommt an.

3 Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i. d. R. 3 Personen zu lösen. **Die Ergebnisse des ersten Termins führen Sie im zweiten Termin dem Tutor vor.** Reichen Sie bitte des weiteren den Quelltext bzw. Lösungen bis Sonntag vor dem dem zweiten Termin 23:55 Uhr per ISIS ein.

3.1 Im ersten Termin zu lösen

Aufgabe 4:
Ihre Aufgabe in diesem Termin ist die Implementierung des Fehlerkontrollprotokolls GoBackN. Hierfür müssen Sie einen Sender und einen Empfänger implementieren. Die zu ergänzenden Dateien sind:
`GoBackNSender.c` und `GoBackNReceiver.c`.

Da Fehler (z.B. Paketverluste und -verfälschungen) und Verzögerungen unter realen Bedingungen kaum reproduzierbar erzeugt werden können, wird die Verbindung zwischen Sender und Empfänger mittels eines Programms (`udp_redirect`) emuliert. GoBackN-Sender und -Empfänger kommunizieren hierbei nicht direkt miteinander, sondern senden ihre Datagramme an `udp_redirect`, welches die Daten ggf. verfälscht und weiterleitet.

Da eine komplette Implementierung recht umfangreich ist, haben wir schon einiges an Code vorgegeben. So können Sie sich auf die eigentliche GoBackN-Implementierung konzentrieren. Bitte verwenden Sie zum Kompilieren unter Linux **m**ake.

Der Sender soll folgende Eigenschaften besitzen (vieles davon erledigt bereits die Vorgabe):

- Die auf der Kommandozeile angegebene Datei wird vom vorgegebenen Code in Pakete a 1024 Bytes zerlegt, mit Sequenznummern versehen und in den Sendepuffer `dataBuffer` gepackt. Das erste Paket bekommt die Sequenznummer 0.
- Alle im Sendepuffer enthaltenen Pakete sollen mittels GoBackN zum Empfänger übertragen werden.
- Als Zeichen, dass die Datei komplett übertragen wurde, soll ein Datenpaket mit 0 Bytes Nutzlast verschickt werden (auch dieses wird bereits automatisch dem Puffer hinzugefügt).
- Es dürfen maximal `window` unbestätigte Pakete versendet werden.

[1] `http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/`

- Um von vornherein den Socket nur anzusprechen, wenn er bereit ist, und gleichzeitig das Timeout zu überwachen, soll `select()` verwendet werden.
- Wenn der Timeout für das älteste gesendete Paket abläuft, sollen alle bisher noch nicht bestätigten Pakete nochmal gesendet werden. Die Timer müssen natürlich vorher mit `resetTimers()` zurückgesetzt werden, da wir die Pakete ja gerade ohnehin wiederholen.
- Wenn ein positives Acknowledgement (also mit einer größeren Sequenznummer als `lastAckSeqNo`) ankommt, sollen die Puffer für ältere Pakete mittels `freeBuffer()` freigegeben werden und Timer und Sequenznummern aktualisiert werden.
- Ältere Acknowledgements soll das Programm verwerfen.
- Zur Verwaltung der Sequenznummern können die Variablen `lastAckSeqNo` und `nextSendSeqNo` verwendet werden. Beachten Sie bitte, dass `lastAckSeqNo` immer die Sequenznummer des vom Empfänger als nächstes erwarteten Pakets enthalten soll. `veryLastSeqNo` enthält die Sequenznummer des letzten für die Datei benötigten Pakets.
- Für die Verwaltung von Timeouts ist die Variable `timerExpiration` gedacht. Sie sollte immer die Ablaufzeit für das älteste noch nicht bestätigte Paket enthalten. Existiert kein solches, sollte `timerExpiration.tv_sec` auf `LONG_MAX` gesetzt werden. Die Länge des Timeouts (wie lange nach dem Versenden des Pakets spätestens eine Bestätigung erwartet wird) ist in der Variablen `timeout` gespeichert.
- Die Timeouts für die einzelnen Pakete sind zusammen mit den Paketen im Sendepuffer abgelegt (siehe unten).

Der Empfänger soll folgende Eigenschaften besitzen:

- Wird das als nächstes erwartete Paket ohne Fehler empfangen, soll mit `sendAck()` ein Acknowledgement gesendet werden. `sendAck()` erwartet als Argumente den Socket-Deskriptor, das Paket, welches bestätigt wird und die Sequenznummer des als nächstes erwarteten Pakets.
- Fehlerhafte Pakete oder Pakete außerhalb der Reihe werden mit einem erneuten Acknowledgement beantwortet. Überlegen Sie sich, welche Sequenznummer dieses Acknowledgement haben muss.
- Zur Verwaltung der Sequenznummern kann die Variable `lastReceivedSeqNo` verwendet werden.

Weitere Hinweise finden Sie in den beiden zu ergänzenden Quelltexten. Stellen, die Sie ergänzen müssen sind jeweils mit einem Kommentar mit `YOUR TASK` gekennzeichnet. Einige Informationen über die zur Verfügung stehenden Hilfsfunktionen folgen im nächsten Abschnitt dieses Blattes.
Zum Testen liegt der Vorlage jeweils für Sender und Empfänger ein vorkompiliertes Binary bei. Eine Anleitung zur Benutzung befindet sich in der README-Datei.

Reichen Sie bitte ihre Lösungen bis Sonntag vor dem zweiten Termin 23:55 Uhr per ISIS ein. Die Lösungen werden **automatisch überprüft** und müssen ohne Ausnahme in einer Datei mit dem Namen **Block8.TXXGYY.tar.gz** (XX = Praktikumstermin, YY = Gruppennummer) eingereicht werden. Darin enthalten erwarten wir in einem Ordner mit dem Namen Block8.TXXGYY ein Makefile, welches die Binärdateien `GoBackNSender` und `GoBackNReceiver` entsprechend kompiliert.[2]

[2]Ein solches Makefile ist schon Teil der Vorlage.

Hilfsfunktionen

Pakete

Die Pakete werden in einer Datenstruktur vom Typ `GoBackNMessageStruct` gespeichert. Diese enthält neben den eigentlichen Daten auch einige zusätzliche Informationen zum Paket. Folgende Felder dürften für Sie besonders wichtig sein:

Funktion	Beschreibung
`seqNo`	Die Sequenznummer des enthaltenen Datenpakets (-1 bei Acknowledgements).
`seqNoExpected`	Enthält bei Acknowledgements die vom Empfänger als nächstes erwartete Sequenznummer. Bei Datenpaketen -1.
`crcSum`	Enthält eine Prüfsumme.

Paketpuffer

Damit Sie sich auf den eigentlichen Fehlerkontroll-Mechanismus konzentrieren können haben wir den nötigen Datenpuffer bereits für Sie implementiert. Er enthält Datenblöcke, die zum einen das Paket selbst (Feld `packet` vom Typ `GoBackNMessageStruct`) und das aktuelle Timeout für dieses Paket (Feld `timeout` von Typ `timeval`).

Bei dem Datenpuffer handelt es sich um eine FIFO-Queue mit zusätzlich wahlfreiem Lesezugriff. Das heißt Pakete können ausschließlich am Ende und am Anfang des Puffers entfernt werden, es kann jedoch über die Sequenznummer direkt auf jedes im Puffer vorhandene Paket zugegriffen werden. Die Pakete müssen in der Reihenfolge ihrer Sequenznummer und ohne Lücken im Puffer liegen.

Die Implementierung liegt in Form eines abstrakten Datentyps vor, nämlich in Form eines opaken Datentyps `DataBuffer` und mehrerer Funktionen, die als erstes Argument jeweils einen `DataBuffer` erwarten und auf diesem Operationen ausführen. Da einige Aufgaben wie das Initialisieren des Puffers und das Einlesen der zu versendenden Datei bereits in der Vorgabe erledigt wurden, enthält die folgende kurze Aufstellung nur die Funktionen, die Sie in Ihrem Code möglicherweise benötigen. Die genaue Signatur der erwähnten und auch der weiteren Funktionen können Sie der Header-Datei `DataBuffer.h` entnehmen.

Funktion	Beschreibung
`getFirstSeqNoOfBuffer()`	Gibt die Sequenznummer des ersten Pakets im Puffer zurück.
`getLastSeqNoOfBuffer()`	Gibt die Sequenznummer des letzten Pakets im Puffer zurück.
`getBufferSize()`	Gibt die derzeitige Anzahl der Pakete im Puffer zurück.
`bufferContainsPacket()`	Prüft, ob sich ein Paket mit der angegebenen Sequenznummer im Puffer befindet.
`getDataPacketFromBuffer()`	Liefert einen Zeiger auf die `DataPacket`-Struktur des Pakets mit der angegebenen Sequenznummer. Das Paket wird **nicht** aus dem Puffer entfernt. Eine Änderung des Pakets über diesen Pointer ändert direkt das Paket im Puffer.
`freeBuffer()`	Löscht die Pakete im angegebenen Sequenznummernbereich aus dem Puffer und gibt diese frei. Der Bereich muss immer mit der ersten im Puffer befindlichen Sequenznummer beginnen.
`printBuffer()`	Gibt eine Liste der im Puffer befindlichen Pakete auf der Standardausgabe aus. Nützlich zum debuggen.
`resetTimers()`	Löscht die Timeouts aller im Puffer befindlichen Pakete (setzt sie auf `LONG_MAX`).

3.2 Im zweiten Termin zu lösen

Aufgabe 5:
Gehen Sie die Vorlesungs- und Praktikumsunterlagen durch und machen Sie sich klar, welche Zusammenhänge Sie noch nicht verstanden haben. Notieren Sie sich Fragen. Wir werden eine Fragestunde anbieten, in der Sie diese Fragen stellen können und sollten.

4 Vertiefungsaufgaben

Aufgabe 7:
Sie haben in der Vorlesung Sliding-Window-Mechanismen kennengelernt. Beantworten Sie in diesem Kontext die folgenden Fragen:

a) Wozu ist bei Sliding-Window-Protokollen ein Verbindungsaufbau nötig?

b) Was macht Slow-start und warum wird es genutzt?

c) Welchen Einfluss haben Paketverluste auf die Größe des windows?

Aufgabe 8:
Effizienz des GoBackN Protokolls.

a) Bestimmen sie die Effizienz des GoBackN Protokolls, wenn bei der Übertragung von Daten keine Fehler auftreten. Die Größe des verwendeten Fensters ist w. Die Größe jedes Datenpakets ist n_d Bits, die Größe der Acknowledgements n_a Bits. Bei jeder Datenübertragung entsteht eine Verzögerung τ. Die Datenrate des verwendeten Mediums ist R. Fertigen sie eine Skizze an und geben sie die allgemeine Formel an.

b) Wie gross muss das Fenster mindestens sein, damit das Protokoll eine Effizienz von 1 erreicht?

Aufgabe 9:
Ein Paket der Länge n soll über einen fehleranfälligen Kanal übertragen werden. Die Bitfehlerwahrscheinlichkeit betrage p_b.

a) Wie groß ist die Wahrscheinlichkeit, dass das Paket ohne Fehler übertragen wird?

b) Welche Annahme liegt Ihrer Berechnung zugrunde und weshalb ist sie realistisch, bzw. unrealistisch?