TKN Telecommunication Networks Group

Praktikum
Rechnernetze und Verteilte Systeme

(letzter) Block 8

— TCP und ARQ-Varfahren —

Termin: 6.-12.2.2016 & 13.-19.2.2016

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:
Sie haben in der Vorlesung die Funktionsweise von TCP kennengelernt. Beantworten Sie in diesem Kontext die folgenden Fragen:

a) Welche Pakete werden beim TCP-Handshake ausgetauscht?

b) Welche Pakete werden ausgetauscht, wenn der eigene Rechner die TCP-Verbindung abbauen möchte? Was passiert, wenn der entfernte Rechner noch ausstehende Daten hat?

c) Was sind die 3 Probleme, mit denen man bei Best-Effort-Netzen rechnen muss?

d) Welche Mechanismen werden genutzt, um die Probleme aus Teil c zu lösen?

Aufgabe 2:
Sie haben in der Vorlesung verschiedene ARQ-Mechanismen kennengelernt. Beantworten Sie in diesem Kontext die folgenden Fragen:

a) Mit Hilfe welcher zwei Mechanismen erkennt ein Sender bzw. Empfänger bei Verwendung eines ARQ Protokolls Fehler?

b) Wie funktioniert das Go-Back-N Verfahren?

c) Wie groß muss bei Go-Back-N der Puffer auf Empfängerseite sein? Wie groß auf Senderseite?

d) Angenommen es wird TCP genutzt, der Sequenznummernraum entspricht deshalb den übertragenen Bytes und der Empfänger hat gerade Byte 4711 empfangen, welche Sequenznummer schreibt der Empfänger in sein ACK-Paket?

e) Was ist ein kumulatives ACK?

Aufgabe 3:
Sie haben in der Vorlesung Sliding-Window-Mechanismen kennengelernt. Beantworten Sie in diesem Kontext die folgenden Fragen:

a) Was ist der Unterschied zwischen Flow- und Congestion-Control?

b) Sei RTT die Round-trip-time zwischen einem Sender und Empfänger mittels eines Sliding-Window-Protokolls. Sei ferner t die Zeit, um das komplette window zu verschicken. Was passiert, wenn $RTT > t$ gilt (ein Blick in die Vorlesungsunterlagen hilft!)?

c) Was machen Router, wenn ihre Puffer voll sind und sie trotzdem weiterhin eingehende Pakete erhalten?

d) Wie interpretiert TCP Paketverluste? Geben Sie ein Beispiel, wann diese Interpretation falsch ist.

2 Präsenzaufgaben

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

Aufgabe 4:
Das folgende Schema soll den Sequenznummernraum beim GoBackN-Sender darstellen. Nutzen Sie dieses um Handsimulationen von GoBackN durchzuführen, indem Sie die beim Sender gepufferten Pakete schraffieren und die Position von `lastAckSeqNo` und `nextSendSeqNo` (siehe unten) markieren. Nehmen Sie eine Fenstergrö”se von 5 Paketen an und spielen Sie verschiedene Szenarien durch (z.B. Verlust eines oder mehrerer Pakete und/oder Acknowledgements)[1].

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

[1] Kreativitätsnachhilfe: Das erste Paket kommt NICHT an, aber die restlichen 4. Oder: 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 5:
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 namens *protsim* emuliert.[2] GoBackN-Sender und -Empfänger verbinden sich hierbei nicht direkt miteinander, sondern per TCP-Socket mit *protsim* und senden Daten und Acknowledgements über diesen Socket an *protsim*. *Protsim* überträgt die Daten über eine emulierte eine Netzverbindung (Fehler, Verzögerungen, ...) aus drei Links und liefert die Pakete über eine weitere TCP-Verbindung an das jeweilige andere Programm aus.

Wir empfehlen die Bearbeitung auf den Ubuntu-Rechnern des IRB. Falls Sie protsim auf dem eigene Rechner verwenden möchten bedenken Sie das es sich bei *protsim* um eine 32-bit binary handelt. Die Nutzung der Ubuntu-Rechner des IRB ist per SSH mit X-Forwarding auch remote möglich:
`xhost + ; ssh -X <account>@furor-ubu.eecsit.tu-berlin.de`.

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. Die notwendige Verbindung zum Netzemulator wird bereits in der Vorgabe aufgebaut.
- 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.
- Das Senden und Empfangen soll nicht-blockierend geschehen. Um zu verhindern, dass beim Senden oder Empfangen blockiert wird, muss zwingend `send` und `recv` verwendet werden und das Flag `MSG_DONTWAIT` übergeben werden (`write` und `read` gehen nicht, da diesen keine Flags mitgegeben werden können). Wenn `send` bzw. `recv` blockieren müssten, um ihre ihre Aufgabe zu erfüllen (z.B. keine Daten zum Empfangen bereit oder Sendepuffer voll), geben sie `-1` zurück und setzen `errno` auf `EAGAIN`.
- Um von vornherein den Socket nur anzusprechen, wenn er bereit ist, und gleichzeitig das Timeout zu überwachen, soll `select()` verwendet werden.

[2] Für die Interessierten: *Protsim* dient eigentlich der Simulation von Kommunikationsnetzen. Es verwendet zur Implementierung das OMNeT++-Framework (`www.omnetpp.org`).

- 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 ohne Reaktion gelöscht.
- 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 weiter unten auf diesem Blatt.

Nachdem Sie Sender und Empfänger komplettiert haben, testen Sie Ihre Implementierungen wie folgt:

1. Wechseln Sie ins *protsim*-Verzeichnis. Sorgen Sie dafür, dass *protsim* die notwendigen Bibliotheken findet:

   ```
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:../lib/ ; export LD_LIBRARY_PATH
   ```

2. Starten Sie *protsim* indem Sie an der Shell `protsim.tk` eingeben (*protsim.cmdenv* stellt eine Kommandozeilenversion bereit, die Sie z.B. über eine Remote-Shell benutzen können oder falls die Animation das Programm zu sehr abbremst). Sie sollten sich hierbei in dem Verzeichnis befinden, in welches Sie die Vorgaben entpackt haben, da sonst die Konfigurationsdateien nicht gefunden werden. Es kann vorkommen, dass ein oder mehrere Fenster von protsim initial nur einige Pixel groß dargestellt werden. Achten Sie auf solche Fenster und vergrößern Sie sie nach Bedarf.

3. Wählen Sie auf die entsprechende Frage Run 1 aus und drücken Sie den "Run"-Button, um den Emulator zu starten. Schieben Sie den Regler für die Animationsgeschwindigkeit (rechts oben in der Netzwerkübersicht) auf Maximum.

4. Öffnen Sie für Sender und Empfänger jeweils ein eigenes Terminal.

5. Starten Sie den Empfänger im `src-task`-Verzeichnis wie folgt:
   `./GoBackNReceiver.musterlsg localhost test.out`
   Statt `test.out` gegen Sie bitte den Namen der Datei an, in die die empfangenen Daten geschrieben werden sollen.

6. Starten Sie den Sender wie folgt:
   `./GoBackNSender.musterlsg -d 10/5 localhost test.in`
   Hierbei steht `test.in` für die zu versendende Datei, z.B. könnten Sie einfach die ausführbare Datei *GoBackNSender* versenden. **Hinweis:** Die voreingestellten Ports sollten i. d. R. funktionieren. Sollte das nicht der Fall sein, können Sie mit der Option `-l` den lokalen Port ändern. Da über die lokale Port-Nummer die Zuordnung zu den Knoten im Emulator stattfindet, sollten Sie darauf achten, dass die letzten drei Ziffern der Port-Nummer gleichbleibt. Zum Ändern des Ports der *protsim*-Instanz müssen Sie in der Datei `omnetpp.ini` die Option `socketrtscheduler-port` ändern und dem Sender und Empfänger diesen Port mit der Option `-r` übergeben.

7. Wenn alle Daten übertragen sind, drücken Sie den *Stop*-Button und beenden Sie *protsim* (beantworten Sie die Frage, ob *finish* aufgerufen werden soll, mit *yes*).

8. Nach der Übertragung vergleichen Sie die versendete und empfangene Datei mittels `diff`. Sind diese nicht identisch, suchen Sie den Fehler in Ihrem Programm. Auch das Ergebnis des nächsten Schrittes kann bei der Fehlersuche hilfreich sein.

9. *protsim* zeichnet während es läuft die Sequenznummern der gesendeten und empfangenen Pakete in der Datei `techgi4.vec` auf und unterscheidet dabei nach intakten und beschädigten Paketen (*corrupted*).

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.
`hasErrors`	Enthält ein Flag zur Prüfung, ob das Paket bei der Übertragung verfälscht wurde und daher verworfen werden muss. In der Realität würde dies mit Hilfe einer Prüfsumme oder eines CRC realisiert werden. Da wir uns in dieser Aufgabe auf die Fehler**korrektur** konzentrieren wollen, wird von der Fehler**erkennung** mittels dieses Flags abstrahiert.

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
`getFirstSeqNo...` `...OfBuffer()`	Gibt die Sequenznummer des ersten Pakets im Puffer zurück.
`getLastSeqNo...` `...OfBuffer()`	Gibt die Sequenznummer des letzten Pakets im Puffer zurück.
`getBufferSize()`	Gibt die derzeitige Anzahl der Pakete im Puffer zurück.
`bufferContains...` `...Packet()`	Prüft, ob sich ein Paket mit der angegebenen Sequenznummer im Puffer befindet.
`getDataPacket...` `...FromBuffer()`	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 6:
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:
Automatic-Repeat-Request-Protokolle

- Welche Arten von Fehlern kann das Send-and-Wait-Protokoll beheben? Welche Mechanismen setzt es dafür ein, und welche neuen Probleme kann dies unter Umständen verursachen?
- Vergleichen sie das Send-and-Wait-Protokoll mit GoBackN und Selective Repeat. Welche Fehlerarten können erkannt werden, und welche Schwachstellen der anderen Fehlerkontrollmechanismen werden jeweils angegangen?

Aufgabe 8:
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 9:
Stellen Sie die Vor- und Nachteile von Go-Back-N und Selective-Repeat unter folgenden Gesichtspunkten gegenüber:

a) Welche Puffergröße wird auf Empfängerseite benötigt?

b) Welche Puffergröße wird auf Senderseite benötigt?

c) Welches ARQ Verfahren ist effizienter?

Aufgabe 10:
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 11:
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?