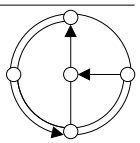
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2007 Prof. R. Wattenhofer / Michael Kuhn / Remo Meier

Vernetzte Systeme
Übung 4

Ausgabe: **19. April 2007**
Abgabe: **27. April 2007**

Bitte schreiben Sie immer Ihre(n) Namen auf die Lösungsblätter.

Vorbemerkung

Sie können sich den für diese Aufgabe benötigten Quellcode von unserer Homepage herunterladen. Da sich die Anwendungen in mehrere Dateien unterteilen, haben wir ein komprimiertes (zip) Archiv bereitgestellt. Bitte Beachten Sie neben den Informationen auf dem Übungsblatt auch die Kommentare im Quellcode[1].

1 Instant Messenger

Mit der Registrierung bei einem zentralen Server haben Sie in Übung 3 bereits einen wichtigen Teil unseres Instant Messengers geschrieben. In dieser Aufgabe geht es darum, die vom Server empfangene Benutzerliste zu verwenden, um anderen Benutzern Nachrichten zukommen zu lassen. Im Gegensatz zur Registrierung, bei der eine TCP-Verbindung verwendet wird, sollen diese Nachrichten als **UDP-Pakete** versendet werden.

Zur Lösung dieser Aufgabe müssen Sie sich mit dem gesamten Quellcode beschäftigen. Schauen Sie sich alle Klassen an, und machen Sie sich deren Bedeutung klar. Schauen Sie sich auch den RegistrationClient an, dessen Struktur und Verhalten sich leicht geändert haben!

Den bereits in der letzten Übung eingeführten Nachrichtentypen fügen wir noch den Typ `MESSAGE` (Wert `0x06`) hinzu. Dieser zeigt einem Empfänger an, dass eine Textnachricht im folgenden Format erhalten wurde[2]:

Nachrichtentyp	Nachrichtenformat (Anzahl der Bytes)
MESSAGE	Typ(1) — Länge des Textes(2) — Text(max. 65535)

Lösen Sie nun die folgenden Teilaufgaben mit Hilfe der hier und auf unserer Webseite aufgeführten Hinweise und der Kommentare im Quellcode:

a) Implementieren Sie die Methode `sendMessage` in der Klasse `MessageSender`.

Die Methode wird aufgerufen, wenn Sie eine Nachricht (*Instant Message*) versenden möchten. Schicken Sie dazu unter Verwendung eines UDP-Pakets den als Parameter angegebenen Text als Nachricht des Typs `MESSAGE` an den ebenfalls spezifizierten Benutzer. Versuchen Sie auch, mögliche Fehler zu erkennen und abzufangen.

[1] Das Gerüst für diese Aufgabe finden Sie wieder auf der Vorlesungs-Homepage. Das Zip-Archiv können Sie entweder einfach entpacken, oder in Eclipse via File -> Import -> Existing Projects into Workspace direkt als neues Projekt importieren.

[2] Die maximale Länge des Textes von 65535 Bytes ist eher theoretischer Natur, da die tatsächlich mögliche Länge eines UDP-Paketes in Abhängigkeit der jeweiligen Protokoll-Implementierung variieren kann. Um diese Probleme zu umgehen, werden Pakete von Anwendungen meist nur mit einer maximalen Grösse von 512 Bytes versendet.

b) Implementieren Sie die Methode `receiveMessage` in der Klasse `MessageReceiver`.

Die Methode wird kontinuierlich in einem eigenen Thread durchlaufen und empfängt somit zeitlich unabhängig von der übrigen Anwendung UDP-Pakete anderer Benutzer. Schreiben Sie den notwendigen Code zum Empfang von Nachrichten des Typs `MESSAGE` über die vorhandene UDP-Verbindung, und zeigen Sie diese an. Versuchen Sie auch hier, mögliche Fehler zu erkennen und abzufangen.

2 Flusssteuerung

Wenn ein Sender schneller sendet, als der Empfänger die empfangenen Daten verarbeiten kann, werden diese meistens auf Empfängerseite gepuffert. Pufferspeicher sind jedoch begrenzt. Um ein Überlaufen des Puffers zu vermeiden, beinhalten Transportprotokolle eine (typischerweise auf der Sicherungsschicht angesiedelte) **Flusssteuerung**. Die Aufgabe der Flusssteuerung ist es, den Empfänger vor einem zu grossen Zufluss von Paketen eines Senders und damit auch vor dem Überlaufen des Puffers zu schützen. Eine einfache Flusssteuerung kann mittels **Halt**- und **Weiter**-Nachrichten realisiert werden, die der Empfänger dem Sender in entgegengesetzter Richtung des Datenflusses übermittelt.

a) Welcher Zusammenhang besteht zwischen dem Produkt von Verzögerung V und Bandbreite R einer Verbindung und der benötigten Puffergrösse?

b) Angenommen, der Empfänger schickt dem Sender sofort eine Halt-Nachricht, sobald er merkt, dass er mit der Verarbeitung der Daten nicht mehr nachkommt. Wie gross muss der Puffer des Empfängers mindestens dimensioniert sein, damit keine Pakete verloren gehen?

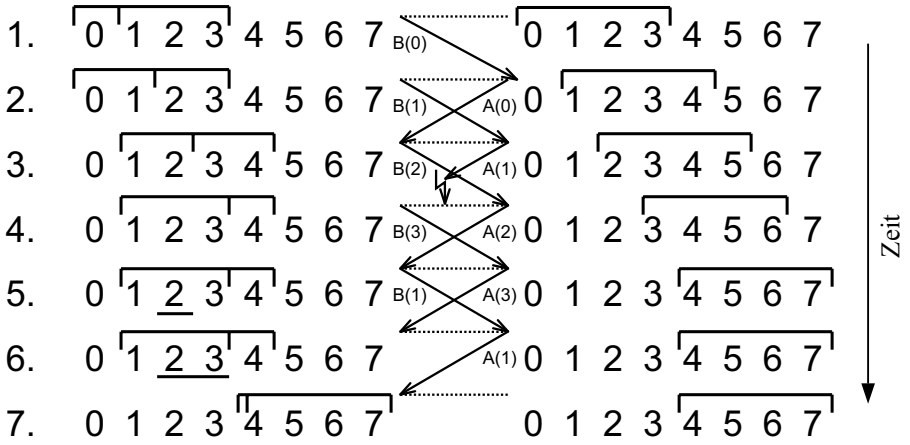
c) Im Internet können die verschiedenen Teilstrecken (Hops) einer Verbindung von A nach B mit verschiedenen Technologien wie Modem- oder Satellitenverbindungen realisiert sein. Welche Probleme ergeben sich, wenn die Teilstrecken unterschiedliche Bandbreiten haben?

3 Schiebefenster und Sequenznummern

In der Vorlesung haben Sie *go-Back-N* und *selective repeat* als zwei Formen für zuverlässige Datentransferprotokolle mit Pipelining kennen gelernt. Pipelining bedeutet dabei, dass der Sender nacheinander mehrere Pakete verschicken kann und diese erst nach und nach vom Empfänger bestätigt (acknowledged) werden können. Damit dies funktioniert, verwenden diese Protokolle auf Sender- und Empfängerseite Puffer, sogenannte Schiebefenster (sliding windows). Für zuverlässigen Datentransfer (damit die Reihenfolge der Pakete erhalten bleibt und kein Paket verloren geht) werden Sequenznummern (sequence numbers) verwendet.

a) In dieser Aufgabe spielen Sie selber einmal das Selective-Repeat-Protokoll durch. Folgendes Beispiel zeigt den Zeitablauf für das Sliding-Window-Protokoll mit Sende- und Empfangspuffergrösse 4, bei dem die Bestätigung des Blocks B(1) verloren geht. Pro Zeiteinheit wird maximal ein Paket versandt. Im Sendefenster sind zusätzlich die Blöcke markiert, die schon bestätigt wurden. Als Fehlerbehandlungsverfahren wird *selective repeat* mit einem Timeout von 3 Zeiteinheiten verwendet, d.h. falls ein gesendeter Block nach drei Zeiteinheiten nicht bestätigt ist, wird er nochmals gesendet.

In jedem Schritt (zum Zeitpunkt der gepunkteten Linien) wird zuerst ein allfälliges Paket empfangen, dann entschieden, ob und wenn ja welches Paket versandt wird, dann dieses ggf. versandt und der neue Zustand dargestellt. Im Beispiel läuft die Zeit von oben nach unten. Dargestellt ist jeweils der Zustand des Sende- bzw. Empfangsfensters kurz nach dem Zeitpunkt der gepunkteten Linien. Der Abstand zwischen zwei gepunkteten Linien entspricht der Zeiteinheit T. Nachrichten sind jeweils genau eine Zeiteinheit (also T) unterwegs.



Nun zu Ihrer Aufgabe: Es sollen 3 Blöcke mit den Sequenznummern 0-2 übertragen werden. Dazu wird das Sliding-Window-Protokoll mit Sendefenstergrösse 3 und Empfangsfenstergrösse 2 und ein Sequenznummernbereich von [0, 5] verwendet. Als Fehlerbehandlungsverfahren verwenden wir *selective repeat* mit einem Timeout von 3 Zeiteinheiten. Block B(0) geht bei der ersten Übertragung verloren. Auch die Bestätigung A(0) der erneuten Übertragung von B(0) geht verloren. Alle anderen Übertragungen verlaufen erfolgreich.

Zeigen Sie mit Hilfe eines Diagramms analog zum Beispiel oben, wie die Fensterstellung nach jedem Erhalt eines Blocks bzw. einer Bestätigung aussieht. Zeichnen Sie nach jeder Übertragung bzw. Bestätigung den Zustand des Sende- und Empfangsfensters in die Schablone a) auf dem Zusatzblatt zum Übungsblatt ein. Markieren Sie auf Empfängerseite auch bereits empfangene, aber noch nicht weitergereichte Pakete.

b) Angenommen, der Sequenznummernbereich für Aufgabenteil a) sei [0, 3], ansonsten bleibe alles gleich. Zeichnen Sie den Ablauf in die Schablone b) ein. An welcher Stelle (in welcher Zeile) scheitert der Algorithmus? Warum scheitert er?

c) Wie viele Sequenznummern (k) sind mindestens nötig, so dass Probleme wie auf Folie 3/41 nicht vorkommen können? Begründen Sie ihr Resultat. Überlegen Sie sich dazu, wie gross der mögliche Bereich von Sequenznummern im Sende- und Empfangsfenster sein kann. Gehen Sie davon aus, dass Sende- und Empfangsfenster gleich gross sind, beide haben Grösse w. Nehmen Sie weiter an, dass der Empfänger Pakete, die in der richtigen Reihenfolge empfangen werden, direkt der Anwendungsschicht weiter reicht.