Praktikum Simulative Untersuchung von Protokollfunktionen des Internet

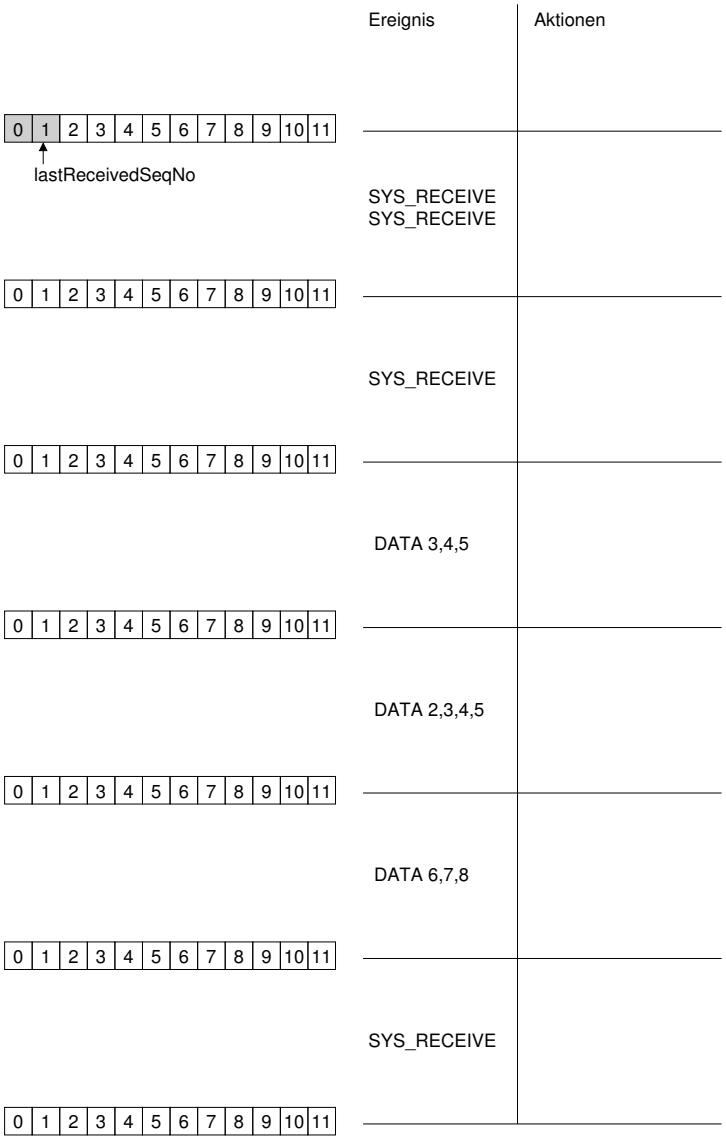
Aufgabenblatt 11 [1]

Vorbereitung

Aufgabe 1: Handsimulation Sender
Bitte lesen Sie sich die Beschreibung im praktischen Teil gut durch. Das folgende Diagramm zeigt einen Ausschnitt des Sequenznummernraums eines Senders. Die Größe des Sendepuffers sei 7 Pakete. Schraffieren Sie bitte in jedem Schritt die im Puffer belegten Sequenznummern und markieren Sie die Positionen von `lastAckSeqNo`, `nextSendSeqNo`, `nextSendBufferSeqNo` und `maxAllowedSeqNo`. Notieren Sie bitte für jedes Ereignis die durchgeführten Aktionen (möglich sind `OKAY`, `TRY_AGAIN` und `send DATA` *`seqNo`*).

[1] Stand: 12. Oktober 2023

Aufgabe 2: Handsimulation Empfänger
Das folgende Diagramm zeigt einen Ausschnitt des Sequenznummernraums eines Empfängers. Die Größe des Empfangspuffers sei 7 Pakete. Schraffieren Sie bitte in jedem Schritt die im Puffer belegten Sequenznummern und markieren Sie die Position von `lastReceivedSeqNo`. Notieren Sie bitte für jedes Ereignis die durchgeführten Aktionen (möglich sind `OKAY`, `TRY_AGAIN` und `send ACK` *`seqNo`* `WIN` *`size`*).



Praktischer Teil

Denken Sie bitte – wie immer bei den praktischen Aufgaben– an aussagekräftige Debug-Ausgaben. Die Sourcen für diese Aufgabe befinden sich im Verzeichnis `protsim11`.

Die Zustandsmaschine in der letzten Aufgabe enthielt für die Datenkommunikation nur ein primitives Send-and-Wait. Nun haben Sie ja schon gelernt, dass Send-and-Wait bei ungünstigem Bandbreite-Verzögerungs-Produkt sehr ineffizient arbeitet.

Eine Verbesserung, die Sie schon kennen gelernt haben, ist Go-Back-N. Ihre Aufgabe in diesem Termin ist nun, Go-Back-N in das Transport-Protokoll einzubauen. Dabei ergeben sich folgende zusätzlichen Schwierigkeiten: Zum einen ist ein Transport-Endpunkt sowohl Sender als auch Empfänger, zum anderen soll er auch eine Flow-Control implementieren um langsame Empfänger nicht zu überlasten. Sie dürfen natürlich Code aus der Go-Back-N-Aufgabe wiederverwenden, allerdings werden Sie einige Änderungen daran vornehmen müssen.

Bei der Go-Back-N-Aufgabe handelte es sich noch um eine eigenständige Applikation, die ihre Daten selbst generiert hat. Bei der Transportschicht ist es nun so, dass sie ihre Dienste der Applikation anbietet, in unserem Fall den einer verlustfreien und reihenfolgetreuen Auslieferung von Datenpaketen. Zur Erfüllung ihrer Aufgabe muss die Transportschicht Pakete von der Applikation oder an die Applikation zwischenpuffern. Dies geschieht bei uns im `sendBuffer` bzw. im `receiveBuffer`. Die Größe dieser Puffer ist begrenzt. Die maximale Größe wird durch die Parameter und Attribute `sendQueue` und `receiveQueue` bestimmt.

System Calls

Die Kommunikation zwischen Applikation und Transportschicht erfolgt mittels sogenannter System Calls. Für diese Aufgabe sind die folgenden vier System Calls relevant:

System Call	Beschreibung
`eSYS_SEND`	Die Applikation möchte eine Nachricht an den Verbindungspartner übermitteln. Die zu übermittelnde Nachricht ist im System Call enkapsuliert. Das genaue Format dieser Nachricht ist nicht festgelegt und für die Transportschicht ohne Belang. Wenn noch Platz im Sendepuffer ist, soll ein `TransportPacket` konstruiert werden, in welches die zu übertragende Nachricht eingebettet wird. Der System-Call wird in diesem Fall (ohne die zuvor eingebettete Nachricht) mit `eOKAY` an die Applikation zurückgeschickt. Die nächste freie Sequenznummer ist im Attribut `nextSendBufferSeqNo` enthalten. Um die Verwaltung dieses Feldes kümmert sich `sendData()`. Wenn der Sendepuffer voll ist, wird der System Call (samt eingebetteter Nachricht) mit dem Fehlercode `eTRY_AGAIN` quittiert.
`eSYS_SEND_READY_IND`	Diese Indication soll von der Transportschicht immer dann gesendet werden, wenn der Sendepuffer zuvor voll war und nun wieder Platz ist.
`eSYS_RECEIVE`	Die Applikation möchte eine Nachricht empfangen. Wenn ein Paket im Empfangspuffer enthalten ist, wird die darin enthaltene Nachricht in den System Call enkapsuliert und mit `eOKAY` zurückgeschickt. Ist der Empfangspuffer leer, wird der Fehlercode `eTRY_AGAIN` zurückgeschickt.
`eSYS_DATA_IND`	Diese Indication soll von der Transportschicht immer dann gesendet werden, wenn der Empfangspuffer zuvor leer war und nun ein neues Datenpaket angekommen ist.

Hilfsfunktionen

Folgende Hilfsfunktionen werden Sie benötigen. Bitte beachten Sie Änderungen und Erweiterungen zum letzten Termin:

Funktion	Beschreibung
`createPacket()`	Erzeugt ein `TransportPacket` mit der angegebenen ID, Sequenznummer und Acknowledgement-Nummer. Wird letztere nicht angegebenen (oder mit -1), enthält das Paket kein Acknowledgement-Feld.
`setTimer()`	Setzt den Timer auf das relativ angegebene Timeout.
`sendAck()`	Sendet eine Acknowledgement-Nachricht mit der entsprechenden erwarteten Sequenznummer. Als weiteres optionales Argument erhält es die Größe des erlaubten Fensters (relativ zur Nummer des acknowledgten Pakets). Da Acknowledgements unter Umständen huckepack auf Datenpaketen reisen, sollte für Acknowledgements immer diese Methode verwendet werden.
`sysCall()`	Sendet einen System-Call mit dem angegebenen Kommando und Ergebnis (`eOKAY`, wenn weggelassen) an die Applikation.
`sysCallReply()`	Beantwortet den angegebenen System-Call mit dem angegebenen Ergebnis-Code (Default: `eOKAY`).
`sendData()`	Sendet ein Datenpaket. Das Paket wird zunächst im Sendepuffer gespeichert und entweder gleich oder später gesendet. Wenn kein Platz mehr im Sendepuffer frei war, gibt die Funktion `false` zurück. Alle Pakete ab Zustand `eESTABLISHED` (also auch `eFIN`) sollten so gesendet werden. Wenn der zweite Parameter `true` ist, wird das Sendefenster ignoriert (z.B. für `eFIN`).
`receiveData()`	Gibt das nächste Paket aus dem Empfangspuffer zurück oder `NULL`, wenn der Puffer leer ist.
`check()`	Wird in der Regel in der Vorgabe schon aufgerufen. Prüft, ob die Nachricht vom richtigen Gegenüber kommt, **eliminiert Duplikate und Out-of-Order-Pakete**. Nur wenn diese Funktion `true` zurückgibt, sollte die Nachricht weiter verarbeitet werden. Ansonsten hat sich schon `check()` darum gekümmert. Beachten Sie bitte, dass sich durch das Vorhandensein dieser Funktion die Go-Back-N-Implementierung vereinfacht.
`invariants()`	Testet, ob alle nötigen Bedingungen (Invarianten) für die verschiedenen Sequenznummern erfüllt sind.

Aufgabe 3: Go-Back-N-Implementierung
Sie müssen zunächst `TransportEndpoint.cc` im Verzeichnis `transport` aus dem letzten Projekt übernehmen. Implementieren Sie anschließend Go-Back-N. Ergänzen Sie bitte folgende Funktionen in der Datei `transport/TransportEndpointData.cc`:

Funktion	Beschreibung
`handleDataSysCall()`	dient der Bearbeitung der System Calls `eSYS_SEND` und `eSYS_RECEIVE`. Es können die Hilfsfunktionen `sendData()` und `receiveData()` benutzt werden.
`handleData()`	dient der Behandlung eingehender `TransportPacket`s, also der Bearbeitung von Daten und Acknowledgements.
`handleDataTimer()`	dient der Bearbeitung ablaufender Sende-Timer.
`doSending()`	soll das eigentliche Senden durchführen, wenn Pakete zum Senden bereit sind, und noch Platz im Sende-Fenster ist. Andernfalls sollte es nichts tun. `doSending()` sollte immer aufgerufen werden, wenn möglicherweise weitere Pakete gesendet werden können. Dies ist z.B. der Fall, wenn Acknowledgements eintreffen, ein Timer abläuft oder `eSYS_SEND` aufgerufen wurde.

In den `SYN`-Paketen sowie in allen Acknowledgements sollte das `window`-Feld so gesetzt werden, dass es die Anzahl freier Plätze im Empfangspuffer angibt. Wenn ein Paket mit gültigem `window` empfangen wird, sollte `maxAllowedSeqNo` entsprechend gesetzt werden.

Der zur Verfügung gestellte Code sendet automatisch ein Paket mit der ID `ePERM`, der nächsten erwarteten Sequenznummer und dem passenden Window, wenn durch diesen Mechanismus das Fenster komplett geschlossen wurde und danach wieder etwas frei wird. Der Verlust des Permits wird auch bereits durch einen Timer abgesichert.

Benutzen Sie bitte wieder die Vektoren `sendSeqNumVector` und `ackSeqNumVector` zum Aufzeichnen der Sequenznummern. `outOfOrderSeqNumVector` wird schon von `check()` korrekt benutzt.

Bitte beachten Sie folgende wichtige Hinweise:

- Beachten Sie bitte, dass `ePERM` Pakete eine Art duplizierter Acknowledgements sind, die jedoch **nicht** von `check()` herausgefiltert werden. Sie müssen also in `handleData()` geeignet auf diese Duplikate reagieren.
- Vergessen Sie nicht, in `TransportEndpoint.cc` das Senden und Verarbeiten von `eSYN`-Paketen um den Fenstermechanismus zu erweitern.
- Denken Sie bitte daran, in den Zuständen vor `eESTABLISHED` `nextSendBufferSeqNo` passend zu initialisieren.
- In den Zuständen ab `eESTABLISHED` ist immer `nextSendBufferSeqNo` zu benutzen, **nie** `nextSendSeqNo`. `nextSendBufferSeqNo` wird von `sendData()` verwaltet, greifen Sie also **nur lesend** darauf zu.

Aufgabe 4: Testen

Rufen Sie Run 1 auf. Die schon implementierten `PeerToPeerAppl`-Applikationen werden nun versuchen dynamisch Verbindungen auf- und abzubauen und einigen sich darauf, wer sendet und wer empfängt. Splitten Sie den transport Vektor mit `./splitvec.py ../results/transpB01.vec out` in alle Subvektoren auf (Skript befindet sich im eval Ordner). Schauen sie sich anschließend mit `./plot.sh out0 out2` das Ergebnis an, es wird der SendSeqNum Vektor und der AckSeqNum Vektor des linken Knotens dargestellt. Überprüfen sie Anhand der Grafik ob ihre Implementierung funktioniert.

Führen Sie nun auch die Runs 1-32 aus (mit `./runAll.sh`). Wechseln Sie in das Verzeichnis `eval` und werten Sie die Runs mit `./ps_eval_arq.py -r 1,32 ../results/transpB` aus.