Formale Simulation und Verifikation verteilter Algorithmen Abgabe der Praktikumsaufgabe 1

Maria Lüdemann und Birger Kamp

April 5, 2016

1 Das Modell als Petrinetz

Zur Lösung der gestellten Aufgabe haben wir zwei Petri-Netze entwickelt. Beide stellen das Modell dar, allerdings wurde das Modell unterschiedlich abstrahiert.

In Variante A wurde der Algorithmus ausführlich implementiert. Dadurch gibt es eine erhöhte Anzahl von Stellen, Transitionen und Kanten.

In Variante B ist der Algorithmus komprimierter und abstrakter dargestellt. Einige der Stellen und Transitionen aus Variante A sind hier zusammengefasst zu finden.

Im Verlaufe dieses Dokumentes wird in der Regel von Modell A gesprochen. Das zweite Modell wird nur zum Vergleich herangezogen und in der weiteren Bearbeitung vernachlässigt.

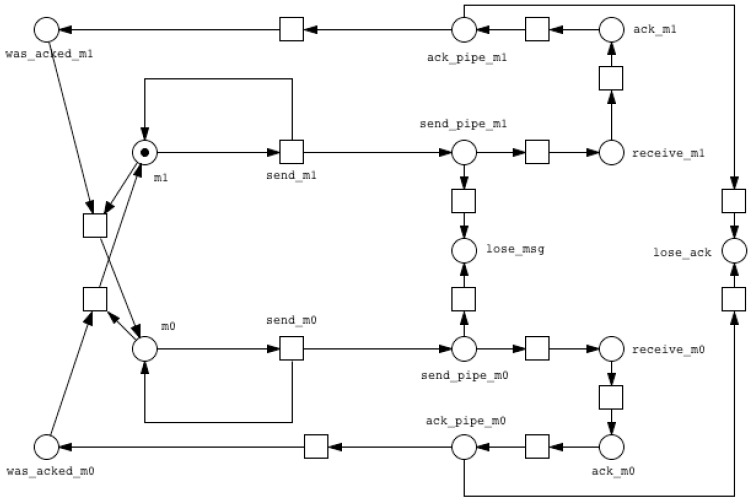

Figure 1: Variante A

2 Erläuterung der Modellierung

2.1 Variante A

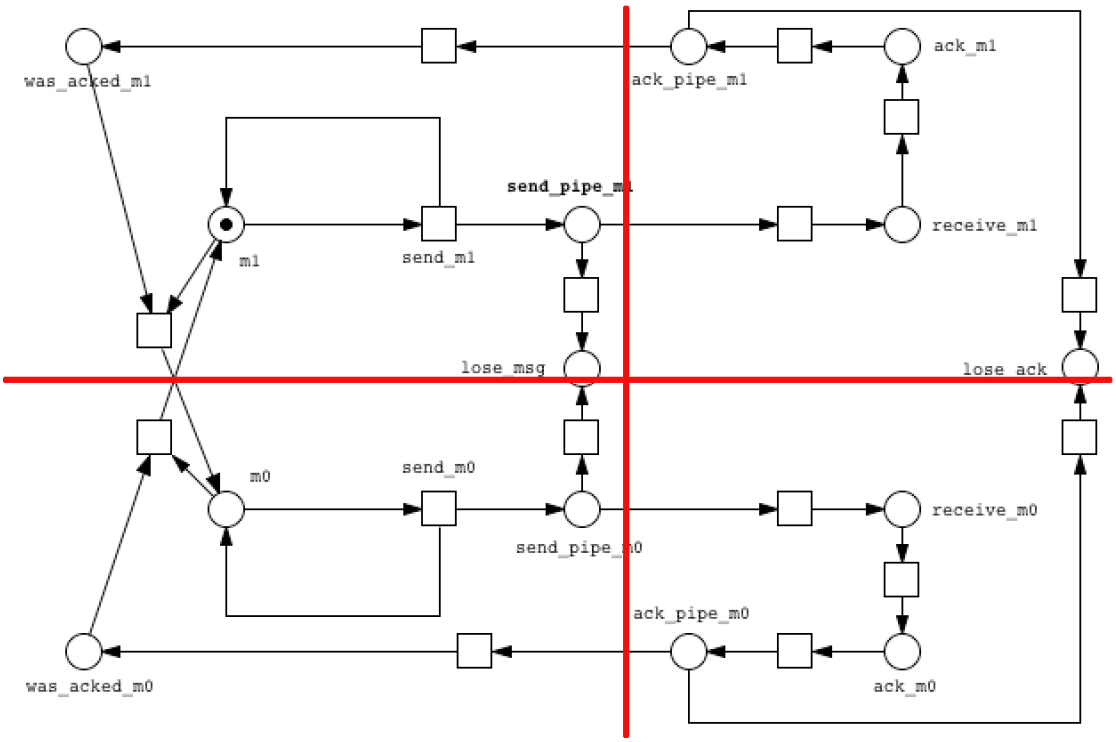

Figure 2: Variante A, in Quadranten eingeteilt

In Abbildung 2 wurde das Petri-Netz in Quadranten eingeteilt, um es hier verständlicher beschreiben zu können.

Die Quadranten oben links und unten links sind der Sender des Algorithmus. Der Empfänger ist in den oben rechts und unten rechts abgebildet. Es fällt auf, dass das Netz über die horizontale Achse symmetrisch ist. Das liegt daran, dass die obere Hälfte sich mit dem Senden und Empfangen der Nachrichten mit positivem Kontrollbit kümmert, während der untere Teil für die Nachrichten mit negativem Kontrollbit zuständig sind. Es reicht daher eine der Hälfte in der Funktionsweise zu erklären, da die andere Hälfte sich genauso verhält, nur dass sie dabei eine andere Nachricht behandelt.

In *m1* ist das Start-Token gesetzt, es wird daher zu Beginn eine Nachricht mit positivem Kontrollbit verschickt. Sobald die Nachricht durch die Stelle *send_m1* verschickt ist, erhält der Initiator *m1* erneut ein Token, dadurch wird ermöglicht, dass die Nachricht erneut verschickt wird, falls der Sender das Empfangen des Empfängers nicht bestätigt bekommt. Außerdem wird die Nachricht "physikalisch" auf die Leitung gelegt, die hier durch die Stelle *send_pipe_m1* repräsentiert wird. Dort kann es passieren, dass die Nachricht verloren geht. Sollte die Nachricht nicht verloren gehen, wird sie vom

Empfänger in *receive_m1* erhalten.

Der Empfänger wird nun den Erhalt der Nachricht an den Sender bestätigen. Dazu möchte der Empfänger eine Nachricht senden in der Stelle *ack_m1*. Die Bestätigungs-Nachricht wird auf die "physikalische" Leitung gelegt, die hier durch die Stelle *ack_pipe_m1* dargestellt wird. Dort kann es passieren, dass die Bestätigungsnachricht verloren geht, und der Sender diese nicht erhält. Sollte der Sender die Bestätigung erhalten haben, merkt er sich dies in der Stelle *was_acked_m1*.

Sobald der Sender wieder sendebereit ist - wenn also ein Token in *m1* ist - kann der Sender wechseln zum Senden von Nachrichten mit negativem Kontrollbit bzw. der Stelle *m0*. Dort beginnt der Sende-Empfangs-Kreislauf wieder wie bereits für *m1* erklärt.

2.2 Variante B

Diese Modellierung beinhaltet zwei autarke Schleifen die das Senden, eventuelle Verlieren und empfangen der Nachrichten modellieren. Diese Schleife gibt es je für die Nachrichten mit dem Kontrollbit 1 und dem für 0. Die Funktionalität ist in beiden Schleifen identisch weshalb hier nur eine Seite beschrieben werden soll.

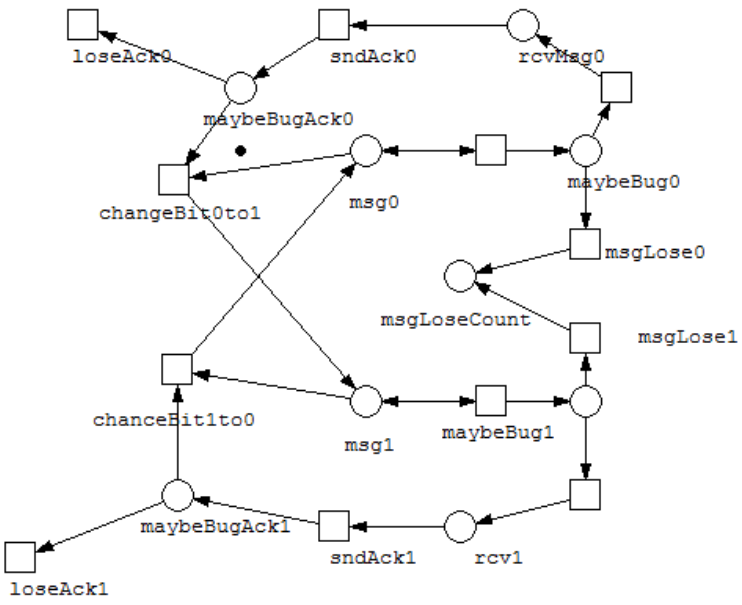

Figure 3: Variante B

3 Begründung der Korrektheit

Anforderung: *Es existiert ein Sender und ein Empfänger*
Der Sender ist in diesem Netz modelliert durch *m0, m1, send_m1* und *send_m0*. Sie

stellen gemeinsam das senden der Nachrichten dar. Der Empfänger wird durch *recieve_m1, ack_m1, recieve_m0* und *ack_m0* modeliert. Sie nehmen die Nachrichten aus dem Sender an und versenden die Bestätigungen.

Anforderung: *In jede Richtung gibt es je einen Nachrichtenkanal*
In diesem Modell werden der Sender und der Empfänger in jeweils zwei Teile aufgeteilt dargestellt. So dass sich zwei Schleifen ergeben die jeweils zwei Kanäle haben. Dies ist eine Modell Entscheidung zur besseren Darstellung des Kontrollbits. Jedoch sind *send_pipe_m0* und *send_pipe_m0* als ein Kanal zum Senden von Sender zu Empfänger gedacht und *ack_pipe_m0* und *ack_pipe_m0* als ein Kanal für die Bestätigung des Empfängers zum Sender.

Anforderung: *Nachrichten können verloren gehen*
Die beiden pipe-Stellen *send_pipe_m1* und *send_pipe_m0* können ihr Token undeterministsch die Stelle *lose_msg* oder *receive_m**weitergeben. Dadurch ist sichergestellt, dass einige Nachrichten verloren gehen und einige Nachrichten ihr Ziel erreichen. Wieviele Nachrichten wo landen, hängt von der Simulationsumgebung ab.

Anforderung: *Nachrichten können nicht verfälscht oder überholt werden*
Die Nachrichten auf den Kanälen werden im Petrinetz als Token dargestellt. Diese Token können nicht verfälscht werden da sie keinen 'Inhalt' tragen. Das verfälschen von Nachrichten ist nicht modelliert worden. Das Überholen von Nachrichten hingegen ist schwieriger. In unseren Testdurchläufen in Snoopy schalten die Transitionen nicht immer automatisch wenn ausreichend Token in den notwendigen Stellen liegen. Sodass sich teilweise Token in Stellen sammeln. Laut Petrinetz Definition sollten Transitionen jedoch sofort schalten sobald die erforderlichen Token vorhanden sind. Somit dürfte laut Modell dieses Verhalten nicht auftreten und sich die Token nicht überholen dürfen. Da Snoopy aber dieses Verhalten an den Tag legt könnte es sein dass sich Token , also Nachrichten, überholen.

Anforderung: *Nachrichten enthält Kontrollbit*
Das Kontrollbit einer Nachricht wird hier repräsentiert durch die Stellen *m1* und *m0*. Wenn eine dieser Stellen schalten kann, wird die Nachricht mit dem jeweiligen Kontrollbit verschickt.

Anforderung: *Nachricht wird wiederholt bis Empfang bestätigt wurde*
Soll eine Nachricht mit Kontrollbit verschickt werden, wird die jeweilige Transition *send_m** geschaltet. Diese Transition legt nicht nur die "Nachricht auf die Leitung", sondern teilt dem dazugehörigen *m** ein Token zu, sodass *m** wieder das Senden einer Nachricht initiieren kann. Dies wird solange wiederholt bis die Stelle *was_acked_m** schalten kann. Dadurch wird das "Sende-Token" von der aktuellen *m**-Stelle abgezogen und auf die jeweils andere *m**-Stelle übertragen, sodass ab dem Zeitpunkt Nachrichten mit dem anderen Kontrollbit verschickt werden.

Anforderung: *Empfänger sendet bei Nachrichterhalt eine Bestätigung an den Sender*
Dies wird modelliert durch die Stelle *receive_m**. Diese kann schalten, sobald eine Nachricht erhalten wurde. Daraufhin wird in *ack_pipe_m** die Bestätigung "auf die Leitung zum Sender gelegt".

Anforderung: *Empfänger wiederholt Bestätigung bis eine Nachricht mit anderem Kontrollbit ankommt*
Der Empfänger bestätigt jede Nachricht, die er bekommt. Sollte der Sender mehrfach Nachrichten mit demselben Kontrollbit schicken, wird für jede der Nachrichten eine Bestätigung an den Sender geschickt.

Anforderung: *Bestätigungen können verloren werden*
Sobald eine Bestätigung vom Empfänger "auf die Leitung gelegt" wird - Stelle *ack_pipe_m** - kann die Transition zur Stelle *lose_ack* geschaltet werden, wodurch die Bestätigungsnachricht "im Netzwerk verloren" wurde.

4 Verlust der Nachrichten

Der Verlust der Nachrichten innerhalb der Modellierung wurde getestet in dem alle erzeugten Token gezählt wurden. Also alle Nachrichten die die Transition *send_m1* oder *send_m0* durchlaufen. Diese Zahl wurde als Randbedingung n für drei Testdurchläufe genutzt. Die Stellen *lose_ack* und *lose_msg* zählen die jeweils verlorenen Pakete und Bestätigungen. Um einen Vergleich haben zu können wurde dieses Verfahren auch in Modell B durchgeführt und die Zahlen gegeneinander gehalten. Dabei ergab sich folgende Tabelle:

Verlorene Pakete	**Verlorene Bestätigungen**	**Gesamt**
30	8	38
25	9	34
23	12	35

Table 1: n = 50 drei Testdurchläufe Modell A

Dabei ergibt sich eine durchschnittliche Verlustrate von ca. 71%. Als Vergleich hierzu die Ergebnisse aus dem Modell B.

Verlorene Pakete	**Verlorene Bestätigungen**	**Gesamt**
24	17	41
19	21	40
21	22	43

Table 2: n = 50 drei Testdurchläufe Modell B

Bei diesem Modell liegt die Verlustrate bei ca. 83% und somit überraschender Weise 10% höher als bei Modell A. Angenommen wurde, dass aufgrund der Eigenschaften der Stellen eine sehr starke Ähnlichkeit in den Verlusten liegt. Die Ergebnisse liegen jedoch relativ Konstant weit auseinander.

5 Verifizieren

Für das Modell sollen diverse Eigenschaften aus zwei Beispielnetzen und unserem Modell A Netz mit dem Charlie Model-Checker verifiziert werden. Dafür werden diverse in den jeweiligen Abschnitten aufgeführten CTL-Ausdrücke verwendet.

5.1 Ab Bestätigung 0-Zustand

Immer wenn der Sender im 1-Zustand ist, kommt irgendwann auch ein 1-Acknowledgement und ab dann (irgendwann) ist der Sender im 0-Zustand.

Dies lies sich durch folgenden CTL-Ausdruck verifizieren:

$(m1 == 1) -> (EF(was\_acked\_m1 > 0) -> EF(m0 == 1))$;

5.2 Solange 0-Zustand beliebig viele 0-Nachrichten

Wenn der Sender im 0-Zustand ist, können 0-Nachrichten solange gesendet werden bis der Sender im 1-Zustand ist.

Dies lies sich durch folgenden CTL-Ausdruck verifizieren:

$(m0 == 1) -> E((EGm0 == 1 \;\&\&\; send\_pipe\_m0 == 1) U EG(m1 == 1 \;\&\&\; m0 == 0))$;

5.3 1-Nachrichten dürfen nicht vom nächsten 1-Zyklus empfangen werden

1-Nachrichten, die gesendet wurden bevor oder während des 0-Bit Zyklus dürfen in dem folgenden 1-Zyklus nicht empfangen werden.

Dies lies sich durch folgenden CTL-Ausdruck verifizieren:

$(m0 == 1 \;\&\&\; m1 == 0) -> (EG(m0 == 0 \;\&\&\; m1 == 1 \;\&\&\; send\_pipe\_m1 == 0 \;\&\&\; receive\_m1 == 0))$;