Modeling the Selective Repeat protocol in EventB

Garvit Juniwal 08005008
Nishant Totla 08005028

guided by
Prof. Om P. Damani

April 20, 2011

Contents

0.1 Introduction 2
0.1.1 Functional requirements 2
0.2 EventB and Rodin 3
0.3 Description of refinements 3
0.3.1 Abstract model SR 0 3
0.3.2 First Refinement model SR 1 4
0.3.3 Second Refinement model SR 2 4
0.3.4 Third Refinement model SR 3 5
0.3.5 Fourth Refinement model SR 4 6
0.4 Proofs using Rodin 6
0.5 Complete Description of all models 7

0.1 Introduction

Selective Repeat ARQ is a specific instance of the Automatic Repeat-Request (ARQ) protocol used for communications. It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units. It is reliable, and offers a distinct advantage over the naive stop-and-wait protocols. Since the protocol uses shifting windows at both the sender and receiver side, the throughput increases. We have developed the Selective Repeat protocol for transferring a data file from a sender to a receiver, using EventB formalism.
Any description of the SR protocol is event based. By past experience of coding it in sequential programming languages (C/C++), it has been realized that an event based framework would be very helpful in modelling such a protocol. Thus, this is a natural choice to consider an EventB model for.

0.1.1 Functional requirements

We shall now enumerate the functional requirements that specify the problem at hand. Note that some of them match those for the already known *Bounded Retransmission Protocol*, but there are some extra requirements, that take us to the Selective Repeat protocol.

1. **FUN 1**: The goal is to totally or partially transfer a certain non-empty original sequential file from one site to another.
2. **FUN 2**: A total transfer means that the transmitted file is a copy of the original one.
3. **FUN 3**: A partial transfer means that the transmitted file is a genuine prefix of the original one.
4. **FUN 4**: Each site may end up in any of the two situations:
   - either if believes that the protocol has terminated successfully
   - or it believes that the protocol has aborted
5. **FUN 5**: When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.
6. **FUN 6**: However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.
7. **FUN 7**: When the Receiver believes that the protocol has terminated successfully, this is because the original file has been entirely copied on the Receivers site.
8. **FUN 8**: When the Receiver believes that the protocol has aborted, this is because the original file has not been copied entirely on the Receivers site.
9. **FUN 9**: The Receiver should maintain a sliding window (the start of which is indicated by a receive base) and buffer the packet it receives if it lies in the range of the window.
10. **FUN 10**: All packets with index not larger than the receive base should have been received and stored correctly at the Receiver. Packets with index number larger than the receive base (but within the window), will either be buffered. And packets beyond the window should not be buffered.
11. **FUN 11**: The Sender also maintains a sliding window. All packets with index not larger than the send base, ACKs (acknowledgements) have been received. Within the window, ACKs have been received for some packets, and packets beyond the window have not been sent.
12. **FUN 12**: All ACKed packets at the Sender should have been correctly received at the Receiver.
13. **FUN 13**: The data or the ACK packets may be lost while transmission.
14. **FUN 14**: For every packet sent by the Sender, it waits for a certain delay `dl` waiting for the ACK to arrive. It should retransmit the packet if the ACK does not arrive. The retransmission of a packet can occur only a bounded number of times, after which, the Sender fails.

15. **FUN 15**: The Sender window should be larger than the Receiver window. This is because a larger Receiver window does not make sense, as it will never be used to the full.

16. **FUN 16**: In order delivery of packets may not be assumed.

17. **FUN 17**: Since sequence number field in a packet is to be kept small, all packet numbers in the data channel and acknowledgement channel must be less than a constant(called MAX_SEQ_NUM henceforth). The value of this constant can however be assumed to be large enough to facilitate correct protocol, given sender and receiver window sizes. This is basically the number after which sender starts to wrap around the sequence numbers of the sent packets. Please see any standard networks textbook for a clearer description of MAX_SEQ_NUM.

IMP:A very important objective of this project is to find a lower bound for the MAX_SEQ_NUM in terms of sender and receiver window sizes(SWS/RWS respectively). For values of MAX_SEQ_NUM $\leq 2\times$ SWS (assuming SWS>RWS), counter examples have been shown which show incorrectness of the protocol as described. But for values above that, we have neither seen any counter examples nor any formal proof of correctness. We expect to find some lower bound for which we can formally prove the protocol correct.

0.2 EventB and Rodin

EventB is a formalism conceived by Jean-Raymond Abrial to model and verify systems. A system is modeled as guided by a specific EventB framework. EventB is event based, so each model consists of a number of events. Events are triggered when the guards guarding them become true. Note that guards are logical predicates. Each event carries out a (possibly non-deterministic) action, to alter the state of the system.

But the key idea that makes the framework useful, is the idea of a refinement. Initially, one starts off with a simple model, called the abstract model. Many details are abstracted out, and this initial model is proved for correctness. After this, a refined model is constructed by adding some more detail, thereby making the model more concrete. The correctness of this new model is proved only against the older model (the one which it refined), and not against the original specification. In this way, we perform refinements, and develop more abstract models, until the final concrete model is obtained. And since the correctness of each iteration is being proved right from the start, we have proved the concrete model correct *by construction.*

Rodin is an Eclipse based platform, that allows a user to model a system within the EventB framework. It allows one to define a model, generates all proof obligations, which can then be proved using either existing theorem provers, or by the user himself. It can be extended using plugins, to cater to extra needs, like proving deadlock freedom etc.

0.3 Description of refinements

Let us assume that the Sender has a file of size n, that it wants to send to the Receiver. Using successive refinements, we shall arrive at a complex protocol that sends the file in packets, and is reliable.
We shall go on to assume that messages can be lost in the data or acknowledgement channels. This is a problem that needs to be taken into account. The following are the refinements that we consider:

0.3.1 Abstract model SR 0

Motivation

We keep the abstract model simple. Here, there are no data channels, no ACKs are sent. The entire file gets transferred to the Receiver in one shot.

Model Description

There are only two events. An initialisation event, and an observer event `brp`, that puts the entire file `f` into `g` in one go. It is to be noted here that the Receiver can directly access the data of the Sender, and create its copy of the file to be transferred.

0.3.2 First Refinement model SR 1

Motivation

In this refinement, we introduce the receiver window. A salient feature of the SR protocol is that the Receiver can buffer packets that it gets. Thus, packets reaching out of order will not be discarded. This directly helps in increasing the throughput of the data transfer, as a whole.

Model Description

With the introduction of the receiver window, we have a *receiver window size*, and a *receiver base*. The receiver window buffers packets that are located not farther from the receiver base than a constant (the receiver window size). All packets upto the index receiver base are asserted to have been buffered at the Receiver. The receiver window shifts when a continuous chunk of data after the receiver base is buffered and available.
Note that in this refinement too, the receiver picks up data directly from the sender. There are no ACKs sent, and no data channel. The sender, in this case, is simple, and does not explicitly send data.

0.3.3 Second Refinement model SR 2

Motivation

This is a rather large refinement. It introduces many new ideas. The first notion is that of a data channel and ACK channel. The Receiver, instead of picking data up directly from the Sender, now picks up data from a data channel. The Sender, on the other hand, keeps putting data into the data channel. There is an analogous ACK channel, which is used to send acknowledgements, which will be introduced shortly. With this arrangement, the complete separation of the Sender and Receiver has been achieved. The only shared part - the size of the file n, is assumed to be shared at the start by some handshaking mechanism.

As just previously mentioned, we have also introduced the concept of an acknowledgement (ACK) in this model. Just like the Sender puts data into a data channel, the Receiver puts packets into an ACK channel after it has received a certain packet. The Sender then picks up ACKs from the ACK channel.

Another important idea put in is that of the sender window. This comes along with a *sender base*, *sender window size*, and *next sequence number* at the sender. Each packet has a status assigned to it (ACKed, Not Sent or Not ACKed). Packets lying before the sender base are all ACKed, which means that they have been sent, their ACKs received, and the Receiver surely has them stored away safely. As for packets in the range from sender base+1 to next sequence number, some might be ACKed (those whose ACKs have been received, and the Receiver has also got them), not ACKed (the packet has been sent, but no response has been received) or not sent (the packet has not been sent yet).

Note that further ideas of packet retransmission and packet loss in channels will be introduced in the next refinement. So, here, the channels are lossless. Also, the range of sequence numbers is assumed to be large, so no wrap around issues are to be dealt with, as of now.

Model Description

The data and ACK channels are sets, as opposed to queues, so that in-order packet transmissions are not necessary. The SR protocol that is used in real world applications assumes in order packet transmission. But a subtle assumtion on the timeouts which we make in the next refinement removes this need, and simplifies our models.

We shall now describe the control flow of the Sender and Receiver automata.

1. **Sender**
   - If the sender window is not full, then send the packet just after next sequence number.
   - If ACK is received for an unACKed packet, change its status from not ACKed to ACKed.
   - If a continuous sequence of packets after sender base are ACKed, shift sender base forward to the last ACKed packet of that chunk.
2. **Receiver**
   - If the received packet has a sequence number in receiver base to receiver base+receiver window size, then buffer the packet if it wasn't already buffered. Also, send an ACK for that sequence number.
   - Otherwise, just send an ACK for the packet received, if it was just buffered, or was buffered sometime in the past.

The events in this refinement implement the above control mechanism. It is taken care that neither the sender nor the receiver cheat in any manner(the receiver always picks up packets only from data channel and doesn't have access to sender's data structure and sender always picks up packets from the acknowledgement channels and doesn't have acccess to receiver's data structures). But certain assumptions are made as described below:

- The total size of the file n, is known to both sender and receiver before the transfer starts.
- Both sender and receiver know each others' window sizes before the transfer starts.

0.3.4 Third Refinement model SR 3

Motivation

In this refinement, we add two more considerations. We introduce daemons, that cause packet loss from the data and ACK channels. So now, the channels are no longer lossless. Secondly, since the protocol is claimed to be reliable, we introduce timeouts for packets sent by the Sender. When the timeout of a packet occurs, the packet is retransmitted, on the assumption that either the packet or its ACK got lost.
When compared to the general SR protocol, where timeouts can be small, we assume that timeouts are large enough so that they become larger than twice the packet propagation time. This means that, when a timeout has occured, either the packet, or its ACK definitely got lost. This assumption simplifies the control automata for the Sender and Receiver, by reducing the number of cases, and thereby also making the model simple. The Sender, in this case, has a limit on the number of consecutive retries, so that if a packet is being resent successively too many (`MAX`) times, then the Sender fails.

Model Description

Since losses are allowed, we now have new (lossy) data and ACK channels, and the lossy ACK channel is a subset of the corresponding ACK channel. Here too, in order packet transmission in not assumed. The control automaton for the Receiver does not change. As for the sender, we only have one more case, where the sender retransmits a packet if the ACK is not received within the timeout interval after its transmission.
Also, the Receiver can conclusively determine that the Sender has failed in case it does not receive any packet for a time interval greater than $dl \times (MAX + 1)$. In this case, the Receiver also fails.
The notion of time has been simulated by using global data structure which hold information about the mortality information of packets i.e. whether a packet is alive in a channel or was lost during transfer etc. Any re-transmit would only occur is a packet was lost. This takes care of our assumption that a time-out difinitely means that a loss has occured in the channels.
We note that the unreliable channels now completely shield the reliable channels of the previous model

in the sense that both the sender and the receiver now only look at the unreliable channels for any action.

0.3.5 Fourth Refinement model SR 4

Motivation

Until now, we have assumed that the set of sequence numbers that a packet can get is unbounded. But in a real implementation, that is not the case. A file can be arbitrarily big, requiring many packets to send, and since the size of a packet header is limited, there is naturally a limit on the maximum sequence number(MAX_SEQ_NUM) that can be assigned to a packet.
Having noted this, it is clear that the same sequence numbers will have to be reused. The most natural method of reuse, that comes to mind, is to assume that the sequence numbers are cyclic, and a wrap around occurs, once the maximum sequence number is reached.
The transition to imposing a limit on the sequence numbers, and the following wrap around is not trivial, and many considerations need to be taken care of. As it turns out, it the maximum sequence number is too small, then wrap around might occur too soon, and two different packets might get sent in the same sender window, with the same sequence number. The Receiver will then not be able to disambiguate between them. Due to this, the value of the maximum sequence number is crucial for correct functioning of the protocol, and depends on the values of the sender window size, and the receiver window size.

Model Description

We introduce new data and acknowledgement channels which carry the wrapped around sequence number of the packets. As in. for a packet at position p in the file, the sequence number transmitted along with it would be $p\%MAX\_SEQ\_NUM$. We remove the reliable data and acknowledgement channels of model sr_2 here because thet are completely shielded by the unreliable channels.On any receive event(either at sender or receiver), we have to carefully determine which real packet does the received packet correspond to since there are many possibilities due to wrap arounds. This also gives an intuitive idea of why smaller than reuquired values of MAX_SEQ_NUM can cause problems.
The SND_rcv_current_ack was refined into two different events here, because in one case you have to determine that an ack is a genuine ack of a previously unacked packet in the window and in the other case you have to determine that the ack is redundant.
All sender and receiver events only look at modulo data/acknowledge channels.

0.4 Proofs using Rodin

- We were able to prove models sr_0 through sr_3 completely in Rodin. Many proofs were done interactively. It was surprising in many cases that rodin was not able to do certain very trivial and direct proofs even after many simplifications. We had to add extra invariants which were not required by the specification just to facilitate the proofs to go through in every model except the first.
- Working with the assumption MAX_SEQ_NUM $> 2 \times$ SWS, we were not able to prove all the proof obligations of model sr_4, since they are too many to be done interactively and too complicated to be handled in little time. Rodin does not behave very well when we use modulo operator profusely. So we cannot make any claims about the bounds of the sequence number as of now. We feel that given sufficient time, we may be able to make a provably correct model for the SR protocol as it exists. But intuitively the model looks correct, expect for the assumption MAX_SEQ_NUM $> 2\times$ SWS which may need to change.
- Proving deadlock freedom in such a protocol is very important. But again, the model has become very huge and the DLF PO is too complicated to be done either by hand or using the flow plugin of rodin.
- Finally, we had a really bad experience with Rodin wherein , all the PO that were discharged interactively were dislodged only because we tried to rename certain invariant. It was very dissapointing.

0.5 Complete Description of all models

The rodin description provided below has been well commented to facilitate understanding. But a minimum knowledge of the SR protocol is assumed.

An Event-B Specification of sr_ctx_1
Creation Date: 20 Apr 2011 @ 00:05:55 AM

CONTEXT sr_ctx_1
contains constants about the file, its data and size

SETS

`D` members of this set can be any data corresponing to each packet in the file

CONSTANTS

`n` number of packets in the file

`f` map of the each packet to data it carries

AXIOMS

axm0_1 : $0 < n$

axm0_2 : $f \in 1 \,..\, n \to D$
total function

END

An Event-B Specification of sr_ctx_2
Creation Date: 20 Apr 2011 @ 00:08:17 AM

CONTEXT sr_ctx_2
protocol can be in working, success or failure states

SETS

`STATUS`

CONSTANTS

`working`

`success`

`failure`

AXIOMS

axm1_1 : $STATUS = \{working, success, failure\}$

axm1_2 : $working \neq success$

axm1_3 : $working \neq failure$

axm1_4 : $success \neq failure$

END

An Event-B Specification of sr_ctx_4(RWS)
Creation Date: 20 Apr 2011 @ 00:08:31 AM

CONTEXT sr_ctx_4(RWS)
constant : receiver window size

CONSTANTS

`RWS` receiver window size

AXIOMS

axm4_1 : $RWS \in \mathbb{N}$

axm4_2 : $RWS > 0$

END

An Event-B Specification of sr_ctx_5
Creation Date: 20 Apr 2011 @ 00:08:34 AM

CONTEXT sr_ctx_5
a packet at sender can be either not-sent, sent but not-acked, or acked

SETS
PACKET_STATUS

CONSTANTS
not_sent
not_acked
acked

AXIOMS
axm5_1 : $PACKET\_STATUS = \{not\_sent, not\_acked, acked\}$
axm5_2 : $not\_sent \neq not\_acked$
axm5_3 : $not\_sent \neq acked$
axm5_4 : $not\_acked \neq acked$

END

An Event-B Specification of sr_ctx_6(SWS)
Creation Date: 20 Apr 2011 @ 00:08:36 AM

CONTEXT sr_ctx_6(SWS)
constant: sender window size

EXTENDS sr_ctx_4(RWS)

CONSTANTS
SWS

AXIOMS
axm6_1 : $SWS \in \mathbb{N}$
axm7_1 : $SWS \geq RWS$
FUN 15,this assumption is good since a larger receiver window does not make sense if the sender cannot send those many packets.

END

An Event-B Specification of sr_ctx_7(life-death)
Creation Date: 20 Apr 2011 @ 00:08:39 AM

CONTEXT sr_ctx_7(life-death)
a packet in its life can either be never sent (zombie), alive in the channels or been dropped(dead)

SETS
MORTAL

CONSTANTS
alive
dead
zombie

AXIOMS
axm1 : $MORTAL = \{alive, dead, zombie\}$
axm2 : $alive \neq dead$
axm3 : $alive \neq zombie$
axm4 : $dead \neq zombie$

END

An Event-B Specification of sr_ctx_8(max retransmissions)
Creation Date: 20 Apr 2011 @ 00:08:42 AM

CONTEXT sr_ctx_8(max retransmissions)
constant: bound on the no. of retransmits

CONSTANTS
MAX_RETRANSMIT

AXIOMS
axm1 : $MAX\_RETRANSMIT \in \mathbb{N}$
axm2 : $MAX\_RETRANSMIT > 0$

END

An Event-B Specification of sr_ctx_9(max_seq_num)
Creation Date: 20 Apr 2011 @ 00:08:45 AM

CONTEXT sr_ctx_9(max_seq_num)
thr wrap around constant for packet sequence numbers

EXTENDS sr_ctx_6(SWS)

CONSTANTS
MAX_SEQ_NUM

AXIOMS
axm1 : $MAX\_SEQ\_NUM \in \mathbb{N}$
axm2 : $MAX\_SEQ\_NUM > RWS + SWS$

END

An Event-B Specification of sr_ctx_9(max_seq_num)
Creation Date: 20 Apr 2011 @ 00:08:45 AM

CONTEXT sr_ctx_9(max_seq_num)
thr wrap around constant for packet sequence numbers

EXTENDS sr_ctx_6(SWS)

CONSTANTS
MAX_SEQ_NUM

AXIOMS
axm1 : $MAX\_SEQ\_NUM \in \mathbb{N}$
axm2 : $MAX\_SEQ\_NUM > RWS + SWS$

END

An Event-B Specification of sr_0
Creation Date: 20 Apr 2011 @ 00:09:01 AM

MACHINE sr_0
receiver gets the sender's file completely non-deterministically, but correct. The protocol may also terminate immaturely.

SEES sr_ctx_1

VARIABLES
i
g

INVARIANTS
inv1 : $i \in 0 \,..\, n$
inv2 : $g \in 1 \,..\, i \rightarrow D$

EVENTS

Initialisation

begin

act1 : $i := 0$

act2 : $g := \varnothing$

end

Event $brp \;\widehat{=}$

begin

act1 : $i, g :| i' \in 0 \,..\, n \land g' = (1 \,..\, i') \lhd f$

end

END

An Event-B Specification of sr_1
Creation Date: 20 Apr 2011 @ 00:09:04 AM

MACHINE sr_1

this model introduces the reeiver window in the system

REFINES sr_0

SEES sr_ctx_1, sr_ctx_2, sr_ctx_4(RWS)

VARIABLES

h file at the receiver's end

r_b base of the receiver's window

s_st

r_st

INVARIANTS

inv1_1 : $r\_b \in 0 \,..\, n$

inv1_2 : $(1 \,..\, r\_b) \lhd h = (1 \,..\, r\_b) \lhd f$

FUN3 the file recieved upto reciver base is correct

inv1_3 : $h \subseteq f$

FUN 10, receiver may buffer some packets

inv1_7 : $dom(h) \subseteq 1 \,..\, n$

inv1_4 : $r\_st = success \Leftrightarrow r\_b = n$

FUN 7 when whole file is received, receiver is successful

inv1_5 : $s\_st = success \Rightarrow r\_st = success$

FUN 5 and 6

inv1_6 : $\forall p \cdot p \in \mathbb{N} \land p > r\_b + RWS \Rightarrow p \notin dom(h)$

FUN 9 and 10, no packet ahead of the current window is received

inv1_8 : $r\_b < n \Rightarrow r\_b + 1 \notin dom(h)$

the maximum prefix of the buffered file must be under receiver base

EVENTS

Initialisation

begin

act1 : $r\_b := 0$

act2 : $h := \varnothing$

act3 : $r\_st := working$

act4 : $s\_st := working$

end

Event $brp \;\widehat{=}$

refines brp

when

grd1 : $s\_st \neq working$

grd2 : $r\_st \neq working$

with

$i' : i' = r\_b$
$g' : g' = (1 \mathrel{..} r\_b) \lhd h$
then
skip
end

Event *RCV_buffer_current_data* $\widehat{=}$
buffer all arriving packets which lie in the window, but window does not advance
any
p
where
grd1 : $r\_st = working$
grd2 : $p \in r\_b + 2 \mathrel{..} r\_b + RWS$
p within RCV window but not r_b+1
grd3 : $p \leq n$
grd4 : $p \notin dom(h)$
p not already received
then
act1 : $h := h \cup \{p \mapsto f(p)\}$
end

Event *RCV_rcv_at_base* $\widehat{=}$
advance receiver window when the paclet just ahead of the base arrives
any
p p is the new base of rcv window
where
grd1 : $r\_st = working$
grd2 : $r\_b + 1 < n$
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $\forall k \cdot k \neq r\_b + 1 \land k \leq p \Rightarrow k \in dom(h)$
grd7 : $p \neq r\_b$
all packets till p are received, except r_b+1
grd5 : $p + 1 \notin dom(h)$
packet next to p must not be received
grd6 : $p \in 1 \mathrel{..} n - 1$
transfer is not successful yet, success is a separate event
then
act1 : $r\_b := p$
act2 : $h := h \cup \{r\_b + 1 \mapsto f(r\_b + 1)\}$
end

Event *RCV_success_on_rcv* $\widehat{=}$
last packet is received
when
grd1 : $r\_st = working$
grd2 : $r\_b + 1 = n$
last packet is welcome
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $\forall k \cdot k \neq r\_b + 1 \land k \leq n \Rightarrow k \in dom(h)$
then
act1 : $r\_st := success$
act2 : $r\_b := n$
act3 : $h := h \cup \{n \mapsto f(n)\}$
end

Event *RCV_failure* $\widehat{=}$
when
grd1 : $r\_st = working$
grd2 : $s\_st = failure$
then

act1 : $r\_st := failure$
end
Event $SND\_success \widehat{=}$
when
grd1 : $s\_st = working$
grd2 : $r\_st = success$
then
act1 : $s\_st := success$
end
Event $SND\_failure \widehat{=}$
when
grd1 : $s\_st = working$
then
act1 : $s\_st := failure$
end
END

An Event-B Specification of sr_2
Creation Date: 20 Apr 2011 @ 00:09:06 AM

MACHINE sr_2
sender, data channel and ack channel are introduced
REFINES sr_1
SEES sr_ctx_1, sr_ctx_2, sr_ctx_5, sr_ctx_6(SWS)
VARIABLES
h
r_b
s_st
r_st
w sender keeps a track of sent, not acked, acked packets
s_b base of sender's window
s_n s_n+1 is the next packet to be sent
data_channel
ack_channel
INVARIANTS
inv2_1 : $s\_b \in 0 \,..\, n$
inv2_2 : $s\_n \in 0 \,..\, n$
inv2_3 : $w \in 1 \,..\, n \rightarrow PACKET\_STATUS$
every packet has a status (not_sent, not_acked, acked)
inv2_4 : $data\_channel \in 1 \,..\, n \nrightarrow D$
data channel contains the packet number and the data of sent packets, since it is a set FUN 16 is met
inv2_5 : $ack\_channel \subseteq 1 \,..\, n$
ack channel contains packets number of packets which are acked by the receiver, since it is a set FUN 16 is met
inv2_6 : $s\_n - s\_b \leq SWS$
FUN 11, at most SWS packets can have a status of not_acked at any time, i.e. sender window is ¡ = SWS
inv2_7 : $s\_n \geq s\_b$
inv2_8 : $data\_channel \subseteq f$
data passed in the data channel is correct
inv2_9 : $\forall p \cdot p \in dom(data\_channel) \lor p \in ack\_channel \Rightarrow w(p) = not\_acked$
all packets in data or ack channels are yet to be acked

inv2_10 : $\forall p \cdot p \in 1 \,..\, n \land w(p) = not\_acked \Rightarrow s\_b < p \land p \leq s\_n$
FUN 11, not-acked packet are only withint the window
inv2_11 : $dom(data\_channel) \cap ack\_channel = \varnothing$
inv2_12 : $\forall p \cdot p \in 1 \,..\, n \land w(p) = not\_sent \Rightarrow p \notin dom(data\_channel) \land p \notin ack\_channel$
inv2_13 : $\forall p \cdot p \in 1 \,..\, n \land p > s\_n \Rightarrow w(p) = not\_sent$
FUN 11
inv2_14 : $s\_b \leq r\_b$
inv2_15 : $ack\_channel \subseteq dom(h)$
inv2_16 : $\forall p \cdot p \in 1 \,..\, n \land w(p) = acked \Rightarrow p \in dom(h)$
FUN 12, acked packets are correctly received

EVENTS

Initialisation

extended

begin

act1 : r_b := 0
act2 : h := ∅
act3 : r_st := working
act4 : s_st := working
act5 : $s\_b := 0$
act6 : $s\_n := 0$
act7 : $w := 1 \,..\, n \times \{not\_sent\}$
act8 : $data\_channel := \varnothing$
act9 : $ack\_channel := \varnothing$

end

Event *brp* $\widehat{=}$

extends *brp*

when

grd1 : s_st ≠ working
grd2 : r_st ≠ working

then

skip

end

Event *RCV_buffer_current_data* $\widehat{=}$

refines *RCV_buffer_current_data*

any

p

where

grd1 : $r\_st = working$
grd2 : $p \in dom(data\_channel)$
receiver takes out a packet from the data channel
grd3 : $p \leq r\_b + RWS$
grd4 : $p \neq r\_b + 1$
grd5 : $p \notin dom(h)$

then

act1 : $h := h \cup \{p \mapsto data\_channel(p)\}$
act2 : $ack\_channel := ack\_channel \cup \{p\}$
send an ack for a received packet
act3 : $data\_channel := \{p\} \lhd data\_channel$

end

Event *RCV_rcv_at_base* $\widehat{=}$

refines *RCV_rcv_at_base*

any

p

where

grd1 : $r\_st = working$

grd2 : $r\_b + 1 < n$
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $r\_b + 1 \in dom(data\_channel)$
grd5 : $p \in 1 .. n - 1$
grd6 : $\forall k \cdot k \neq r\_b + 1 \wedge k \leq p \Rightarrow k \in dom(h)$
grd7 : $p + 1 \notin dom(h)$
grd8 : $p \neq r\_b$

then

act1 : $r\_b := p$
act2 : $h := h \cup \{r\_b + 1 \mapsto data\_channel(r\_b + 1)\}$
act3 : $ack\_channel := ack\_channel \cup \{r\_b + 1\}$
act4 : $data\_channel := \{r\_b + 1\} \lhd data\_channel$

end

Event $RCV\_success\_on\_rcv \widehat{=}$
refines $RCV\_success\_on\_rcv$

when

grd1 : $r\_st = working$
grd2 : $r\_b + 1 = n$
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $r\_b + 1 \in dom(data\_channel)$
grd5 : $\forall k \cdot k \neq r\_b + 1 \wedge k \leq n \Rightarrow k \in dom(h)$

then

act1 : $r\_st := success$
act2 : $r\_b := r\_b + 1$
act3 : $h := h \cup \{n \mapsto data\_channel(n)\}$
act4 : $ack\_channel := ack\_channel \cup \{r\_b + 1\}$
act5 : $data\_channel := \{r\_b + 1\} \lhd data\_channel$

end

Event $RCV\_rcv\_just\_ack \widehat{=}$
send an ack for a retransmitted packet

any

p

where

grd1 : $r\_st = working$
grd3 : $p \leq r\_b + RWS$
grd4 : $p \in dom(h)$
p has already been received
grd5 : $w(p) = not\_acked$

then

act1 : $ack\_channel := ack\_channel \cup \{p\}$
act2 : $data\_channel := \{p\} \lhd data\_channel$

end

Event $RCV\_rcv\_ignore \widehat{=}$
ignore packets otuside the receiver window

any

p

where

grd1 : $r\_st = working$
grd2 : $p \in dom(data\_channel)$
grd3 : $p > r\_b + RWS$
outside the receiver window

then

act1 : $data\_channel := \{p\} \lhd data\_channel$
no ack is sent

end

Event $RCV\_failure \widehat{=}$

extends *RCV_failure*
  when
    grd1 : r_st = working
    grd2 : s_st = failure
  then
    act1 : r_st := failure
  end
Event *SND_snd_data* $\widehat{=}$
  send new data, but within the allowed window size
  when
    grd1 : $s\_st = working$
    grd2 : $s\_n < n$
    grd3 : $s\_n < s\_b + SWS$
      window size restriction
  then
    act1 : $data\_channel := data\_channel \cup \{s\_n + 1 \mapsto f(s\_n + 1)\}$
    act2 : $w(s\_n + 1) := not\_acked$
    act3 : $s\_n := s\_n + 1$
  end
Event *SND_rcv_current_ack* $\widehat{=}$
  receive an ack but not at s_b+1, such that window does not advance
  any
    p
  where
    grd1 : $s\_st = working$
    grd2 : $p \in ack\_channel$
    grd3 : $p \neq s\_b + 1$
  then
    act1 : $w(p) := acked$
    act2 : $ack\_channel := ack\_channel \setminus \{p\}$
  end
Event *SND_rcv_ack_at_base* $\widehat{=}$
  receive an ack at s_b+1, such that sender window advances by 1
  when
    grd3 : $s\_st = working$
    grd1 : $s\_b + 1 < n$
    grd2 : $s\_b + 1 \in ack\_channel$
  then
    act1 : $ack\_channel := ack\_channel \setminus \{s\_b + 1\}$
    act2 : $w(s\_b + 1) := acked$
    act3 : $s\_b := s\_b + 1$
  end
Event *SND_shift_base* $\widehat{=}$
  since acks are also buffered, previously buffered ack can cause sender window to advance
  when
    grd1 : $s\_st = working$
    grd2 : $s\_b + 1 < n$
    grd3 : $w(s\_b + 1) = acked$
  then
    act1 : $s\_b := s\_b + 1$
  end
Event *SND_success_on_ack* $\widehat{=}$
  last ack is received when window base is at n-1
refines *SND_success*
  when
    grd1 : $s\_st = working$

grd2 : $s\_b + 1 = n$
grd3 : $s\_b + 1 \in ack\_channel$

then

act1 : $s\_st := success$
act2 : $w(n) := acked$
act3 : $ack\_channel := ack\_channel \setminus \{s\_b + 1\}$
act4 : $s\_b := s\_b + 1$

end

Event *SND_success_on_shift_base* $\widehat{=}$

last ack was already buffered, success due to simple advancemetn of sender window

refines *SND_success*

when

grd1 : $s\_st = working$
grd2 : $s\_b + 1 = n$
grd3 : $w(s\_b + 1) = acked$

then

act1 : $s\_st := success$
act2 : $s\_b := s\_b + 1$

end

Event *SND_failure* $\widehat{=}$

extends *SND_failure*

when

grd1 : s_st = working

then

act1 : s_st := failure

end

END

An Event-B Specification of sr_3
Creation Date: 20 Apr 2011 @ 00:09:08 AM

MACHINE sr_3

channels are made unreliable, packet loss may occur in either channel

REFINES sr_2

SEES sr_ctx_1, sr_ctx_2, sr_ctx_5, sr_ctx_7(life-death), sr_ctx_8(max retransmissions), sr_ctx_6(SWS)

VARIABLES

h
r_b
s_st
r_st
w
s_b
s_n
data_channel
ack_channel
data_channel_unreliable
ack_channel_unreliable packet loss may occur in these channels
packet_life_indicator used to indicate whether a packet is not yet sent out, alive in the channels or has been killed used to simulate time outs. a resend would only occur if a packet is dead. This ensures that when a packet times out, it is sure that it(or its ack) has nbeen lost
number_of_retries for ensuring bounded no. of re transmissions for each packet

INVARIANTS

inv3_1 : $packet\_life\_indicator \in 1 \mathbin{..} n \rightarrow MORTAL$
packet is either zombie(not yet sent), alive or dead(lost)

inv3_2 : $data\_channel\_unreliable \in 1 \mathbin{..} n \mapsto D$

inv3_3 : $ack\_channel\_unreliable \subseteq 1 \mathbin{..} n$

inv3_4 : $\forall p \cdot p \in 1 \mathbin{..} n \wedge (p \in dom(data\_channel\_unreliable) \vee p \in ack\_channel\_unreliable) \Rightarrow packet\_life\_indicator(p) = alive$

inv3_5 : $number\_of\_retries \in 1 \mathbin{..} n \rightarrow \mathbb{N}$

inv3_7 : $ack\_channel\_unreliable \subseteq ack\_channel$
acks can be lost in unreliable channel, so it is a subset of the abstract reliable channel

inv3_6 : $\forall p \cdot p \in dom(data\_channel\_unreliable) \wedge p \notin ack\_channel \Rightarrow p \in dom(data\_channel)$
unreliable channel is a subset of the abstract reliable one, but unreliable data channel may contain resent packets which are lost in ack channel

inv3_8 : $\forall p \cdot p \in 1 \mathbin{..} n \wedge packet\_life\_indicator(p) = dead \Rightarrow p \in dom(data\_channel) \vee p \in ack\_channel$
all dead packets are present in abstract reliable channels

inv3_9 : $dom(data\_channel\_unreliable) \cap ack\_channel\_unreliable = \varnothing$

inv3_10 : $s\_st = failure \Rightarrow (\exists p \cdot p \in 1 \mathbin{..} n \wedge number\_of\_retries(p) > MAX\_RETRANSMIT)$
FUN 14, sender fails only if, at least one packet has been resent for more than allowed times

inv3_11 : $\forall p \cdot p \in dom(data\_channel\_unreliable) \Rightarrow data\_channel\_unreliable(p) = f(p)$

inv3_12 : $\forall p \cdot p \in 1 \mathbin{..} n \wedge p \leq s\_b \Rightarrow w(p) = acked$
this could have been added in the previous refinement, but as not needed. Here it was added to facilitate proofs.

EVENTS

Initialisation

extended

begin

act1 : r_b := 0
act2 : h := ∅
act3 : r_st := working
act4 : s_st := working
act5 : s_b := 0
act6 : s_n := 0
act7 : w := 1 .. n × {not_sent}
act8 : data_channel := ∅
act9 : ack_channel := ∅
act13 : $data\_channel\_unreliable := \varnothing$
act12 : $ack\_channel\_unreliable := \varnothing$
act10 : $packet\_life\_indicator := 1 \mathbin{..} n \times \{zombie\}$
act11 : $number\_of\_retries := 1 \mathbin{..} n \times \{0\}$

end

Event *brp* $\widehat{=}$

extends *brp*

when

grd1 : s_st ≠ working
grd2 : r_st ≠ working

then

skip

end

Event *RCV_buffer_current_data* $\widehat{=}$

refines *RCV_buffer_current_data*

any

p

where

grd1 : $r\_st = working$

grd2 : $p \in dom(data\_channel\_unreliable)$

pick up from the unreliable channel instead of the reliable one

grd3 : $p \leq r\_b + RWS$

grd4 : $p \neq r\_b + 1$

grd5 : $p \notin dom(h)$

then

act1 : $h := h \cup \{p \mapsto data\_channel\_unreliable(p)\}$

act2 : $ack\_channel := ack\_channel \cup \{p\}$

act3 : $data\_channel := \{p\} \lhd data\_channel$

act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{p\}$

act4 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$

end

Event $RCV\_rcv\_at\_base$ $\widehat{=}$

refines $RCV\_rcv\_at\_base$

any

p

where

grd1 : $r\_st = working$

grd2 : $r\_b + 1 < n$

grd3 : $r\_b + 1 \notin dom(h)$

grd4 : $r\_b + 1 \in dom(data\_channel\_unreliable)$

grd5 : $p \in 1 \,..\, n - 1$

grd6 : $\forall k \cdot k \neq r\_b + 1 \land k \leq p \Rightarrow k \in dom(h)$

grd7 : $p + 1 \notin dom(h)$

grd8 : $p \neq r\_b$

then

act1 : $r\_b := p$

act2 : $h := h \cup \{r\_b + 1 \mapsto data\_channel\_unreliable(r\_b + 1)\}$

act3 : $ack\_channel := ack\_channel \cup \{r\_b + 1\}$

act4 : $data\_channel := \{r\_b + 1\} \lhd data\_channel$

act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{r\_b + 1\}$

act6 : $data\_channel\_unreliable := \{r\_b + 1\} \lhd data\_channel\_unreliable$

end

Event $RCV\_success\_on\_rcv$ $\widehat{=}$

refines $RCV\_success\_on\_rcv$

when

grd1 : $r\_st = working$

grd2 : $r\_b + 1 = n$

grd3 : $r\_b + 1 \notin dom(h)$

grd4 : $r\_b + 1 \in dom(data\_channel\_unreliable)$

grd5 : $\forall k \cdot k \neq r\_b + 1 \land k \leq n \Rightarrow k \in dom(h)$

then

act1 : $r\_st := success$

act2 : $r\_b := r\_b + 1$

act3 : $h := h \cup \{n \mapsto data\_channel\_unreliable(n)\}$

act4 : $ack\_channel := ack\_channel \cup \{r\_b + 1\}$

act5 : $data\_channel := \{r\_b + 1\} \lhd data\_channel$

act6 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{r\_b + 1\}$

act7 : $data\_channel\_unreliable := \{r\_b + 1\} \lhd data\_channel\_unreliable$

end

Event $RCV\_rcv\_just\_ack$ $\widehat{=}$

refines $RCV\_rcv\_just\_ack$

any

p

where
- grd1 : $r\_st = working$
- grd2 : $p \in dom(data\_channel\_unreliable)$
- grd3 : $p \leq r\_b + RWS$
- grd4 : $p \in dom(h)$

then
- act1 : $ack\_channel := ack\_channel \cup \{p\}$
- act2 : $data\_channel := \{p\} \lhd data\_channel$
- act3 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{p\}$
- act4 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$

end

Event *RCV_rcv_ignore* $\widehat{=}$
refines *RCV_rcv_ignore*

any
p

where
- grd1 : $r\_st = working$
- grd2 : $p \in dom(data\_channel\_unreliable)$
- grd3 : $p > r\_b + RWS$

then
- act1 : $data\_channel := \{p\} \lhd data\_channel$
- act2 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$

end

Event *RCV_failure* $\widehat{=}$
refines *RCV_failure*

any
p

where
- grd1 : $r\_st = working$
- grd4 : $p \in 1 \,..\, n$
- grd3 : $number\_of\_retries(p) > MAX\_RETRANSMIT$

  if any packet is retransmitted ¿ MAX_RETRANSMIT times, receiver also fails. It is like receiver does not receive anything for dl*(MAX_RETRANSMIT +1) time

then
- act1 : $r\_st := failure$

end

Event *SND_snd_data* $\widehat{=}$
refines *SND_snd_data*

when
- grd1 : $s\_st = working$
- grd2 : $s\_n < n$
- grd3 : $s\_n < s\_b + SWS$

then
- act1 : $data\_channel := data\_channel \cup \{s\_n + 1 \mapsto f(s\_n + 1)\}$
- act2 : $w(s\_n + 1) := not\_acked$
- act3 : $s\_n := s\_n + 1$
- act4 : $packet\_life\_indicator(s\_n + 1) := alive$
- act5 : $data\_channel\_unreliable := data\_channel\_unreliable \cup \{s\_n + 1 \mapsto f(s\_n + 1)\}$

end

Event *SND_rcv_current_ack* $\widehat{=}$
refines *SND_rcv_current_ack*

any
p

where
- grd1 : $s\_st = working$
- grd2 : $p \in ack\_channel\_unreliable$

grd3 : $p \neq s\_b + 1$
then
act1 : $w(p) := acked$
act2 : $ack\_channel := ack\_channel \setminus \{p\}$
act3 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{p\}$
end

Event *SND_rcv_ack_at_base* $\widehat{=}$
refines *SND_rcv_ack_at_base*
when
grd3 : $s\_st = working$
grd1 : $s\_b + 1 < n$
grd2 : $s\_b + 1 \in ack\_channel\_unreliable$
then
act1 : $ack\_channel := ack\_channel \setminus \{s\_b + 1\}$
act2 : $w(s\_b + 1) := acked$
act3 : $s\_b := s\_b + 1$
act4 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{s\_b + 1\}$
end

Event *SND_shift_base* $\widehat{=}$
extends *SND_shift_base*
when
grd1 : $\mathtt{s\_st} = \mathtt{working}$
grd2 : $\mathtt{s\_b} + 1 < \mathtt{n}$
grd3 : $\mathtt{w}(\mathtt{s\_b} + 1) = \mathtt{acked}$
then
act1 : $\mathtt{s\_b} := \mathtt{s\_b} + 1$
end

Event *SND_success_on_ack* $\widehat{=}$
refines *SND_success_on_ack*
when
grd1 : $s\_st = working$
grd2 : $s\_b + 1 = n$
grd3 : $s\_b + 1 \in ack\_channel\_unreliable$
then
act1 : $s\_st := success$
act2 : $w(n) := acked$
act3 : $ack\_channel := ack\_channel \setminus \{s\_b + 1\}$
act4 : $s\_b := s\_b + 1$
act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{s\_b + 1\}$
end

Event *SND_success_on_shift_base* $\widehat{=}$
extends *SND_success_on_shift_base*
when
grd1 : $\mathtt{s\_st} = \mathtt{working}$
grd2 : $\mathtt{s\_b} + 1 = \mathtt{n}$
grd3 : $\mathtt{w}(\mathtt{s\_b} + 1) = \mathtt{acked}$
then
act1 : $\mathtt{s\_st} := \mathtt{success}$
act2 : $\mathtt{s\_b} := \mathtt{s\_b} + 1$
end

Event *SND_failure* $\widehat{=}$
refines *SND_failure*
any
p
where

grd1 : $s\_st = working$
grd3 : $p \in 1 \,..\, n$
grd2 : $number\_of\_retries(p) = MAX\_RETRANSMIT$
grd4 : $packet\_life\_indicator(p) = dead$

then

act1 : $s\_st := failure$
act2 : $number\_of\_retries(p) := number\_of\_retries(p) + 1$

end

Event $DMN\_data\_channel \,\widehat{=}$

FUN 13, packet is lost in data_channel

any

p

where

grd1 : $p \in dom(data\_channel\_unreliable)$

then

act1 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$
act2 : $packet\_life\_indicator(p) := dead$

end

Event $DMN\_ack\_channel \,\widehat{=}$

FUN 13, ack is lost in ack channel

any

p

where

grd1 : $p \in ack\_channel\_unreliable$

then

act1 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{p\}$
act2 : $packet\_life\_indicator(p) := dead$

end

Event $SND\_resend\_data \,\widehat{=}$

FUN 14, sender re-transmits afer a timeout. Here after a packet is lost in the data/ack channel.

any

p

where

grd3 : $s\_st = working$
grd4 : $p \in 1 \,..\, n$
grd1 : $packet\_life\_indicator(p) = dead$
grd2 : $number\_of\_retries(p) < MAX\_RETRANSMIT$

then

act1 : $data\_channel\_unreliable := data\_channel\_unreliable \cup \{p \mapsto f(p)\}$
act2 : $packet\_life\_indicator(p) := alive$
act3 : $number\_of\_retries(p) := number\_of\_retries(p) + 1$

end

END

An Event-B Specification of sr_4
Creation Date: 20 Apr 2011 @ 00:09:10 AM

MACHINE sr_4

channels are converted into modulo channels. instead of real packet/ack numbers, we transmit packet/ack number modulo MAX_SEQ_NUM

REFINES sr_3

SEES sr_ctx_1, sr_ctx_2, sr_ctx_5, sr_ctx_7(life-death), sr_ctx_8(max retransmissions), sr_ctx_9(max_seq_num)

VARIABLES

`h`
`r_b`
`s_st`
`r_st`
`w`
`s_b`
`s_n`
`data_channel_unreliable`
`ack_channel_unreliable`
`packet_life_indicator`
`number_of_retries`
`data_channel_modulo` contain packets with packet numbers as (absolute packet number modulo MAX_SEQ_NUM)
`ack_channel_modulo` same for ack Note: In this refinement, the variables data_channel and ack_channel are removed, since thery are completely shielded by respective unreliable channels

INVARIANTS

inv4_1 : $data\_channel\_modulo \in 0 \mathinner{..} MAX\_SEQ\_NUM - 1 \nrightarrow D$
inv4_2 : $ack\_channel\_modulo \subseteq 0 \mathinner{..} MAX\_SEQ\_NUM - 1$
inv4_3 : $\forall p \cdot p \in dom(data\_channel\_unreliable) \Rightarrow (p \% MAX\_SEQ\_NUM) \in dom(data\_channel\_modulo)$

inv 3,4,5 say that there is one-one mapping between data_channel_unreliable and data_channel_modulo Packets are same in both. This also ensures that no two packets in previous channel map to the same modulus, i.e.MAX_SEQ_NUM should be large enough to ensure this.

inv4_4 : $\forall p \cdot p \in dom(data\_channel\_modulo) \Rightarrow (\exists q \cdot q \in 1 \mathinner{..} n \land (q \% MAX\_SEQ\_NUM) = p \land q \in dom(data\_channel\_unreliable))$
inv4_5 : $\forall p1, p2 \cdot p1 \in dom(data\_channel\_unreliable) \land p2 \in dom(data\_channel\_unreliable) \land (p1 \% MAX\_SEQ\_NUM = p2 \% MAX\_SEQ\_NUM) \Rightarrow p1 = p2$
inv4_6 : $\forall p \cdot p \in ack\_channel\_unreliable \Rightarrow (p \% MAX\_SEQ\_NUM) \in ack\_channel\_modulo$
inv 6,7,8 say the same about ack channel
inv4_7 : $\forall p \cdot p \in ack\_channel\_modulo \Rightarrow (\exists q \cdot q \in 1 \mathinner{..} n \land (q \% MAX\_SEQ\_NUM) = p \land p \in ack\_channel\_unreliable)$
inv4_8 : $\forall p1, p2 \cdot p1 \in ack\_channel\_unreliable \land p2 \in ack\_channel\_unreliable \land (p1 \% MAX\_SEQ\_NUM = p2 \% MAX\_SEQ\_NUM) \Rightarrow p1 = p2$

EVENTS

Initialisation

begin

act1 : $r\_b := 0$
act2 : $h := \varnothing$
act3 : $r\_st := working$
act4 : $s\_st := working$
act5 : $s\_b := 0$
act6 : $s\_n := 0$
act7 : $w := 1 \mathinner{..} n \times \{not\_sent\}$
act13 : $data\_channel\_unreliable := \varnothing$
act12 : $ack\_channel\_unreliable := \varnothing$
act10 : $packet\_life\_indicator := 1 \mathinner{..} n \times \{zombie\}$
act11 : $number\_of\_retries := 1 \mathinner{..} n \times \{0\}$
act14 : $ack\_channel\_modulo := \varnothing$
act15 : $data\_channel\_modulo := \varnothing$

end

Event $brp \;\widehat{=}$
extends brp

when
grd1 : s_st ≠ working
grd2 : r_st ≠ working
then
skip
end

Event *RCV_buffer_current_data* $\widehat{=}$
refines *RCV_buffer_current_data*
any
q packet number in modulo data channel
p sequence number in the file within the receiver window to which q maps This notion of p and q carries further.
where
grd1 : $r\_st = working$
grd2 : $q \in dom(data\_channel\_modulo)$
grd3 : $p \leq r\_b + RWS$
grd4 : $p \neq r\_b + 1$
grd5 : $p \notin dom(h)$
grd6 : $(p\%MAX\_SEQ\_NUM) = q$
grd7 : $p \in 1 \mathrel{..} n$
then
act1 : $h := h \cup \{p \mapsto data\_channel\_modulo(q)\}$
act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{p\}$
act4 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$
act6 : $ack\_channel\_modulo := ack\_channel\_modulo \cup \{q\}$
act7 : $data\_channel\_modulo := \{q\} \lhd data\_channel\_modulo$
end

Event *RCV_rcv_at_base* $\widehat{=}$
refines *RCV_rcv_at_base*
any
p
q
where
grd1 : $r\_st = working$
grd2 : $r\_b + 1 < n$
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $q \in dom(data\_channel\_modulo)$
grd9 : $q = (r\_b + 1)\%MAX\_SEQ\_NUM$
grd5 : $p \in 1 \mathrel{..} n - 1$
grd6 : $\forall k \cdot k \neq r\_b + 1 \land k \leq p \Rightarrow k \in dom(h)$
grd7 : $p + 1 \notin dom(h)$
grd8 : $p \neq r\_b$
then
act1 : $r\_b := p$
act2 : $h := h \cup \{r\_b + 1 \mapsto data\_channel\_modulo(q)\}$
act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{r\_b + 1\}$
act6 : $data\_channel\_unreliable := \{r\_b + 1\} \lhd data\_channel\_unreliable$
act7 : $ack\_channel\_modulo := ack\_channel\_modulo \cup \{q\}$
act8 : $data\_channel\_modulo := \{q\} \lhd data\_channel\_modulo$
end

Event *RCV_success_on_rcv* $\widehat{=}$
refines *RCV_success_on_rcv*
any
q
where
grd1 : $r\_st = working$

grd2 : $r\_b + 1 = n$
grd3 : $r\_b + 1 \notin dom(h)$
grd4 : $q \in dom(data\_channel\_modulo)$
grd6 : $q = (r\_b + 1)\%MAX\_SEQ\_NUM$
grd5 : $\forall k \cdot k \neq r\_b + 1 \wedge k \leq n \Rightarrow k \in dom(h)$

then

act1 : $r\_st := success$
act2 : $r\_b := r\_b + 1$
act3 : $h := h \cup \{n \mapsto data\_channel\_unreliable(n)\}$
act6 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{r\_b + 1\}$
act7 : $data\_channel\_unreliable := \{r\_b + 1\} \lhd data\_channel\_unreliable$
act8 : $ack\_channel\_modulo := ack\_channel\_modulo \cup \{q\}$
act9 : $data\_channel\_modulo := \{q\} \lhd data\_channel\_modulo$

end

Event $RCV\_rcv\_just\_ack \mathrel{\widehat{=}}$

send ack for any modulo packet which maps to an already received packet within [r_b-SWS,r_b+RWS)

refines $RCV\_rcv\_just\_ack$

any

q
p

where

grd1 : $r\_st = working$
grd2 : $q \in dom(data\_channel\_modulo)$
grd3 : $p \leq r\_b + RWS$
grd6 : $p > r\_b - SWS$
grd4 : $p \in dom(h)$
grd5 : $q = (p\%MAX\_SEQ\_NUM)$
grd7 : $p \in 1 \,..\, n$

then

act3 : $ack\_channel\_unreliable := ack\_channel\_unreliable \cup \{p\}$
act4 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$
act5 : $ack\_channel\_modulo := ack\_channel\_modulo \cup \{q\}$
act6 : $data\_channel\_modulo := \{q\} \lhd data\_channel\_modulo$

end

Event $RCV\_rcv\_ignore \mathrel{\widehat{=}}$

do not send ack for any modulo packet which does not map to any packet within [r_b-SWS,r_b+RWS)

refines RCV_rcv_ignore

any

q
p

where

grd1 : $r\_st = working$
grd2 : $q \in dom(data\_channel\_modulo)$
grd3 : $p > r\_b + RWS \vee p \leq r\_b - SWS$
grd4 : $q = (p\%MAX\_SEQ\_NUM)$
grd5 : $p \in 1 \,..\, n$
grd6 : $\forall k \cdot k > r\_b - SWS \wedge k \leq r\_b + RWS \Rightarrow q \neq (k\%MAX\_SEQ\_NUM)$

then

act2 : $data\_channel\_unreliable := \{p\} \lhd data\_channel\_unreliable$
act3 : $data\_channel\_modulo := \{q\} \lhd data\_channel\_modulo$

end

Event $RCV\_failure \mathrel{\widehat{=}}$

extends $RCV\_failure$

any

p

where

grd1 : $\mathtt{r\_st} = \mathtt{working}$

grd4 : $\mathtt{p} \in 1 \,..\, \mathtt{n}$

grd3 : $\mathtt{number\_of\_retries(p)} > \mathtt{MAX\_RETRANSMIT}$

if any packet is retransmitted more than MAX_RETRANSMIT times, receiver also fails. It is like receiver does not receive anything for dl*(MAX_RETRANSMIT +1) time

then

act1 : $\mathtt{r\_st} := \mathtt{failure}$

end

Event *SND_snd_data* $\widehat{=}$

refines *SND_snd_data*

when

grd1 : $s\_st = working$

grd2 : $s\_n < n$

grd3 : $s\_n < s\_b + SWS$

then

act2 : $w(s\_n + 1) := not\_acked$

act3 : $s\_n := s\_n + 1$

act4 : $packet\_life\_indicator(s\_n + 1) := alive$

act5 : $data\_channel\_unreliable := data\_channel\_unreliable \cup \{s\_n + 1 \mapsto f(s\_n + 1)\}$

act6 : $data\_channel\_modulo := data\_channel\_modulo \cup \{(s\_n + 1\%MAX\_SEQ\_NUM) \mapsto f(s\_n + 1)\}$

in the modulo channel, instead of inserting the packet with its original sequence number, insert it with packet number modulo MAX_SEQ_NUM

end

Event *SND_rcv_current_ack* $\widehat{=}$

accept an ack if there is a packet on sender which maps to this modulo ack and is not yet acked.

refines *SND_rcv_current_ack*

any

p

q

where

grd1 : $s\_st = working$

grd2 : $q \in ack\_channel\_modulo$

grd3 : $p \neq s\_b + 1$

grd4 : $q = (p\%MAX\_SEQ\_NUM)$

grd5 : $w(p) = not\_acked$

then

act1 : $w(p) := acked$

act3 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{p\}$

act4 : $ack\_channel\_modulo := ack\_channel\_modulo \setminus \{q\}$

end

Event *SND_ignore_current_ack* $\widehat{=}$

ignore an ack if there is no packet on sender which maps to this modulo ack and is not yet acked.

refines *SND_rcv_current_ack*

any

p

q

where

grd1 : $s\_st = working$

grd2 : $q \in ack\_channel\_modulo$

grd3 : $p \neq s\_b + 1$

grd4 : $q = p\%MAX\_SEQ\_NUM$
grd5 : $w(p) = acked$
grd6 : $\forall k \cdot k \leq s\_n \wedge (k\%MAX\_SEQ\_NUM = q) \Rightarrow w(k) = acked$
then
act3 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{p\}$
act4 : $ack\_channel\_modulo := ack\_channel\_modulo \setminus \{q\}$
end

Event $SND\_rcv\_ack\_at\_base \widehat{=}$
refines $SND\_rcv\_ack\_at\_base$
any
q
where
grd3 : $s\_st = working$
grd1 : $s\_b + 1 < n$
grd2 : $q \in ack\_channel\_modulo$
grd4 : $q = (s\_b + 1)\%MAX\_SEQ\_NUM$
then
act2 : $w(s\_b + 1) := acked$
act3 : $s\_b := s\_b + 1$
act4 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{s\_b + 1\}$
act5 : $ack\_channel\_modulo := ack\_channel\_modulo \setminus \{q\}$
end

Event $SND\_shift\_base \widehat{=}$
extends SND_shift_base
when
grd1 : $\mathtt{s\_st} = \mathtt{working}$
grd2 : $\mathtt{s\_b} + 1 < \mathtt{n}$
grd3 : $\mathtt{w}(\mathtt{s\_b} + 1) = \mathtt{acked}$
then
act1 : $\mathtt{s\_b} := \mathtt{s\_b} + 1$
end

Event $SND\_success\_on\_ack \widehat{=}$
refines $SND\_success\_on\_ack$
any
q
where
grd1 : $s\_st = working$
grd2 : $s\_b + 1 = n$
grd3 : $q \in ack\_channel\_modulo$
grd4 : $q = (s\_b + 1)\%MAX\_SEQ\_NUM$
then
act1 : $s\_st := success$
act2 : $w(n) := acked$
act4 : $s\_b := s\_b + 1$
act5 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{s\_b + 1\}$
act6 : $ack\_channel\_modulo := ack\_channel\_modulo \setminus \{q\}$
end

Event $SND\_success\_on\_shift\_base \widehat{=}$
extends $SND\_success\_on\_shift\_base$
when
grd1 : $\mathtt{s\_st} = \mathtt{working}$
grd2 : $\mathtt{s\_b} + 1 = \mathtt{n}$
grd3 : $\mathtt{w}(\mathtt{s\_b} + 1) = \mathtt{acked}$
then
act1 : $\mathtt{s\_st} := \mathtt{success}$
act2 : $\mathtt{s\_b} := \mathtt{s\_b} + 1$

end
Event *SND_failure* $\widehat{=}$
extends *SND_failure*
any
p
where
grd1 : $\mathtt{s\_st} = \mathtt{working}$
grd3 : $\mathtt{p} \in \mathtt{1} \,..\, \mathtt{n}$
grd2 : $\mathtt{number\_of\_retries(p)} = \mathtt{MAX\_RETRANSMIT}$
grd4 : $\mathtt{packet\_life\_indicator(p)} = \mathtt{dead}$
then
act1 : $\mathtt{s\_st} := \mathtt{failure}$
act2 : $\mathtt{number\_of\_retries(p)} := \mathtt{number\_of\_retries(p)} + 1$
end
Event *DMN_data_channel* $\widehat{=}$
refines *DMN_data_channel*
any
p
q
where
grd1 : $q \in dom(data\_channel\_modulo)$
grd2 : $q = p\%MAX\_SEQ\_NUM$
grd3 : $p \in dom(data\_channel\_unreliable)$
then
act1 : $data\_channel\_unreliable := \{p\} \lhd\!\!\!- \, data\_channel\_unreliable$
act2 : $packet\_life\_indicator(p) := dead$
act3 : $data\_channel\_modulo := \{q\} \lhd\!\!\!- \, data\_channel\_modulo$
end
Event *DMN_ack_channel* $\widehat{=}$
refines *DMN_ack_channel*
any
p
q
where
grd1 : $q \in ack\_channel\_modulo$
grd2 : $q = p\%MAX\_SEQ\_NUM$
grd3 : $p \in ack\_channel\_unreliable$
then
act1 : $ack\_channel\_unreliable := ack\_channel\_unreliable \setminus \{p\}$
act2 : $packet\_life\_indicator(p) := dead$
act3 : $ack\_channel\_modulo := ack\_channel\_modulo \setminus \{q\}$
end
Event *SND_resend_data* $\widehat{=}$
refines *SND_resend_data*
any
p
where
grd3 : $s\_st = working$
grd4 : $p \in 1 \,..\, n$
grd1 : $packet\_life\_indicator(p) = dead$
grd2 : $number\_of\_retries(p) < MAX\_RETRANSMIT$
then
act1 : $data\_channel\_unreliable := data\_channel\_unreliable \cup \{p \mapsto f(p)\}$
act2 : $packet\_life\_indicator(p) := alive$
act3 : $number\_of\_retries(p) := number\_of\_retries(p) + 1$
act4 : $data\_channel\_modulo := data\_channel\_modulo \cup \{(p\%MAX\_SEQ\_NUM) \mapsto f(p)\}$
end
END