Event-B Course

5. Bounded Retransmission Protocol

Jean-Raymond Abrial

September-October-November 2011

Purpose of this Lecture

- The Bounded Re-transmission Protocol is a file transfer protocol

- This is a problem dealing with fault tolerance

- We suppose that the transfer channels are unreliable

- We present classical solutions to handle that problem: timers.

- We would like to see how we can formalize such timers

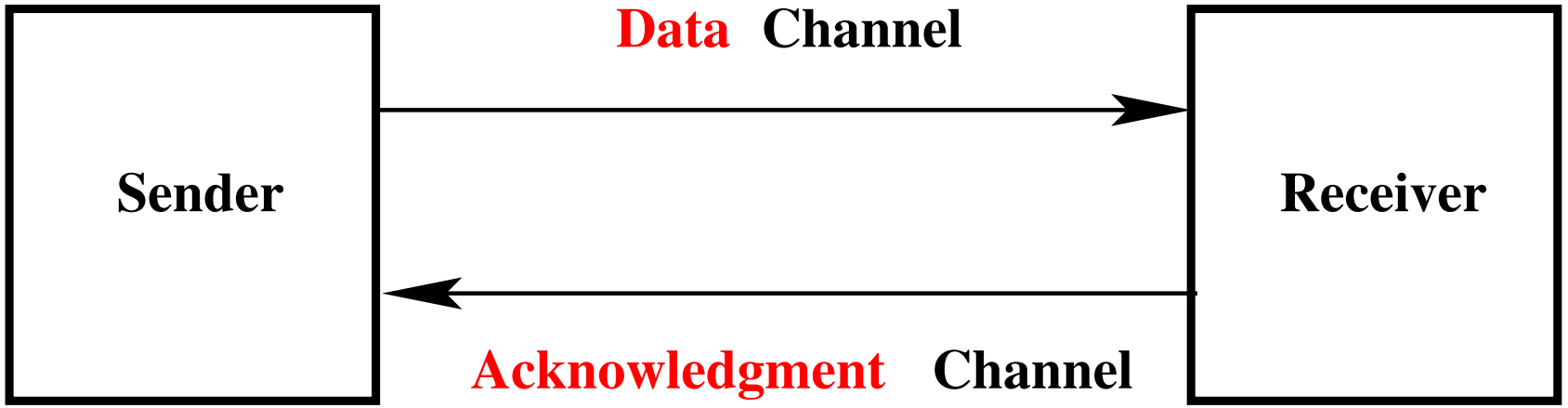
The Bounded Retransmission Protocol

- A sequential file is transmitted from a Sender to a Receiver

- The file is transmitted piece by piece through a Data Channel

- After receiving some data, the Receiver sends an acknowledgment

- After receiving it, the Sender sends the next piece of data, etc.



- Messages can be lost in the Data or Acknowledgment channels

Requirements (1)

The goal of the BRP is to totally or partially transfer a certain non-empty original sequential file from one site to another.	FUN_1
A total transfer means that the transmitted file is a copy of the original one.	FUN_2
A partial transfer means that the transmitted file is a genuine prefix of the original one.	FUN_3

Unreliability of the Communications (1)

- Messages can be lost in the Data or Acknowledgment channels

- The Sender starts a timer before sending a piece of data

- The timer wakes up the Sender after a delay dl

- This occurs if the Sender has not received an acknowledgment in the meantime

Unreliability of the Communications (2)

- dl is guaranteed to be greater than twice the transmission time

- When waken up, the Sender is then sure that the data or the acknowledgment has been lost

- When waken up, the Sender re-transmits the previous data

- The Sender sends an alternating bit together with a new data

- This ensures that the Receiver does not confuse (?) a new data with a retransmitted one.

Abortion of Protocol at the Sender Site

- The Sender can transmit the same data at most $MAX + 1$ times

- After this, the Sender decides to abort

- How does the Receiver know that the Sender aborted?

Abortion of Protocol at the Receiver Site

- Each time the Receiver receives a new piece of data, it starts a timer

- The timer wakes up the Receiver after a delay $(MAX + 1) \times dl$

- This occurs if the Sender has not received a new data in the meantime.

- After this delay, the Receiver is certain that the Sender has aborted

- Then the Receiver aborts too.

Final Situation of the Protocol

- At the end of the protocol, we might be in one of the three situations:

(1) The file has been transmitted entirely and the Sender has received the last acknowledgment

(2) The file has been transmitted entirely but the Sender has not received the last acknowledgment

(3) The file has not been transmitted entirely

Requirements (2)

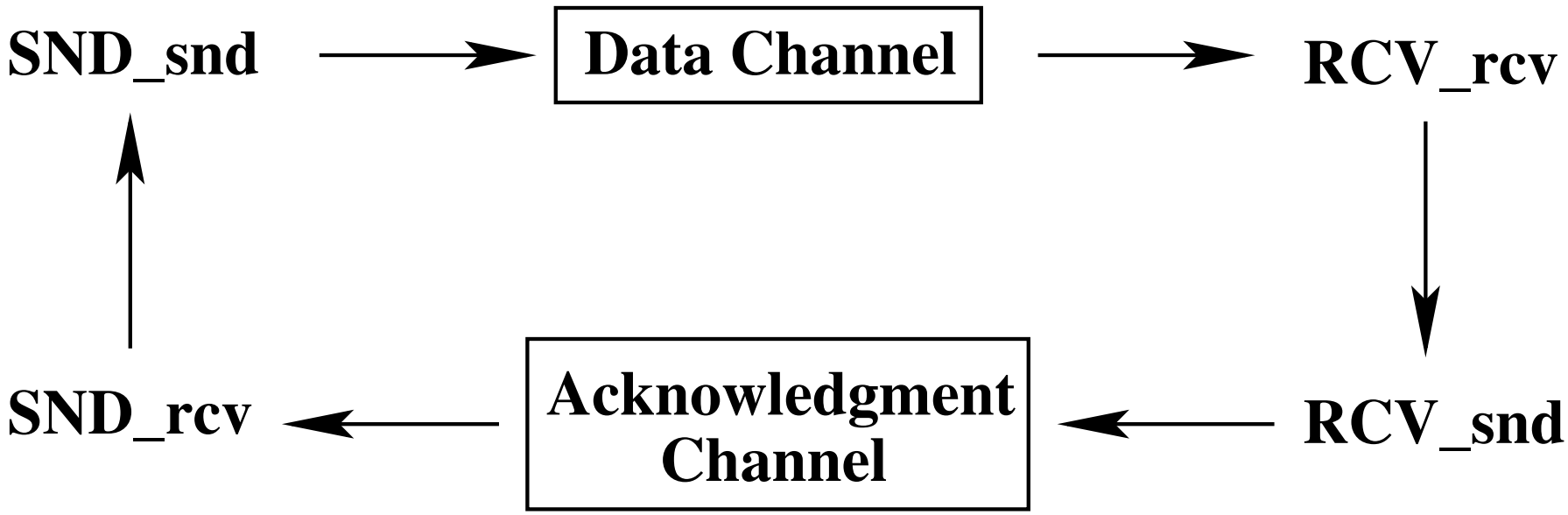
Each site may end up in any of the two situations: - either it believes that the protocol has terminated successfully, - or it believes that the protocol has aborted	FUN_4

When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.	FUN_5

Requirements (3)

However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.	FUN_6
When the Receiver believes that the protocol has terminated successfully, this is because the original file has been entirely copied on the Receiver's site.	FUN_7
When the Receiver believes that the protocol has aborted, this is because the original file has not been copied entirely on the Receiver's site.	FUN_8

Pseudo-code for the Protocol



The Sender sends Data

SND_snd
  when
    SND_snd is waken up
  then
    Acquire data from Sender's file;
    Store acquired data on Data Channel;
    Store Sender's bit on Data Channel;
    Start Sender's timer;
    Activate Data Channel;
  end

The Receiver Receives Data

```
RCV_rcv
   when
      Data Channel interrupt occurs
   then
      Acquire Sender's bit from Data Channel;
      if  Sender's bit = Receiver's bit  then
         Acquire Data from Data Channel;
         Store data on Receiver's file;
         Modify Receiver's bit;
         if  data is not the last one  then
            Start Receiver's timer;
         end
      end
      Reset Data Channel Interrupt;
      Wake up RCV_snd;
   end
```

The Receiver sends Acknowledgment

RCV_snd
when
RCV_snd is waken up
then
Activate Acknowledgment Channel;
end

The Sender Receives Acknowledgment

```
SND_rcv
   when
      Acknowledgment Channel interrupt occurs
   then
      Remove Data from Sender's file;
      Reset retry counter;
      Modify Sender's bit;
      Wake up event SND_snd;
      Reset Acknowledgment Channel interrupt;
      if  Sender's file is not empty  then
         Wake up event SND_snd
      end
   end
```

Timer Interrupt Occurs at Sender's Site


```
SND_timer
   when
      Sender's timer interrupt occurs
   then
      if  retry counter is equal to MAX+1  then
         Abort protocol on Sender's site;
      else
         Increment retry counter;
         Wake up event SND_snd;
      end
   end
```

Timer Interrupt occurs at Receiver's Site

RCV_timer
when
Receiver's timer interrupt occurs
then
Abort protocol on Receiver's site
end

About the Pseudo-code

- Quite often, protocols are "specified" by such pseudo-codes

- In fact, such a pseudo-code raises a number of questions:
  - Are we sure that this description is correct?
  - Are we sure that this protocol terminates?
  - What kinds of properties should this protocol maintain?

- Hence the formal development which is presented now

Refinement Strategy

(0) The status (Success or Failure): FUN_4

(1) (2) Connections between the status: FUN_5 and FUN_6

(3) Partial transmission of the file in one shot: FUN_1, FUN_2, FUN_3

(4) Each participant has access to the other: FUN_7,FUN_8

(5) Introducing unreliable channels and timers.

(6) Optimize protocol

Reminder (1)

The goal of the BRP is to totally or partially transfer a certain non-empty original sequential file from one site to another.	FUN_1
A total transfer means that the transmitted file is a copy of the original one.	FUN_2
A partial transfer means that the transmitted file is a genuine prefix of the original one.	FUN_3

Reminder (2)

Each site may end up in any of the two situations: - either it believes that the protocol has terminated successfully, - or it believes that the protocol has aborted	FUN_4

When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.	FUN_5

Reminder (3)

However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.	FUN_6
When the Receiver believes that the protocol has terminated successfully, this is because the original file has been entirely copied on the Receiver's site.	FUN_7
When the Receiver believes that the protocol has aborted, this is because the original file has not been copied entirely on the Receiver's site.	FUN_8

- This model deals with a very abstract partial requirement: FUN-4.

Each site may end up in any of the two situations: - either it believes that the protocol has terminated successfully, - or it believes that the protocol has aborted	FUN_4

Initial Model: the Constants

We introduce the concept of status.

sets: $STATUS$

constants: $working$ $success$ $failure$

axm1_1: partition$(STATUS, \{working\}, \{success\}, \{failure\})$

- Mind the way enumerated sets are defined

Initial Model: the State

- There are two variables s_st and r_st

- They define the status of the two participants:

variables: s_st, r_st

inv0_1: $s\_st \in STATUS$

inv0_2: $r\_st \in STATUS$

Initial Model: the Events

- Initially, the participants are both $working$.

- We have then an observer event named brp.

- It is fired when both participants are not working any more.

init
$s\_st := working$
$r\_st := working$

brp
when
$s\_st \neq working$
$r\_st \neq working$
then
skip
end

Initial Model: the Anticipated Events

In what follows, we use the technique of anticipated events

SND_progress
status
anticipated
when
$s\_st = working$
then
$s\_st :\in \{success, failure\}$
end

RCV_progress
status
anticipated
when
$r\_st = working$
then
$r\_st :\in \{success, failure\}$
end

First and Second Refinements

- Taking account of requirement FUN-5 and FUN-6

When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.	FUN_5

However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.	FUN_6

inv1_1: $s\_st = success \Rightarrow r\_st = success$

- It makes more precise what is meant by the previous anticipated event.

Events of First Refinement

- We split now event SND_progress into success and failure events.

- Notice that event SND_success is cheating

- It contains the status of the other participant in its guards

SND_success
refines
SND_progress
status
convergent
when
$s\_st = working$
$r\_st = success$
then
$s\_st := success$
end

SND_failure
refines
SND_progress
status
convergent
when
$s\_st = working$
then
$s\_st := failure$
end

variant1: $\{success, failure\} \setminus \{s\_st\}$

Events of Second Refinement

- We split now events RCV_progress into success and failure events.

- Notice that event RCV_failure is cheating

- It contains the status of the other participant in its guards

RCV_success
refines
RCV_progress
status
convergent
when
$r\_st = working$
then
$r\_st := success$
end

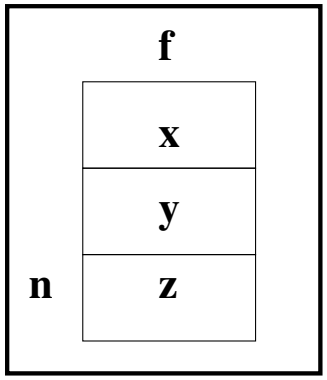
RCV_failure
refines
RCV_progress
status
convergent
when
$r\_st = working$
$s\_st = failure$
then
$r\_st := failure$
end

variant2: $\{success, failure\} \setminus \{r\_st\}$

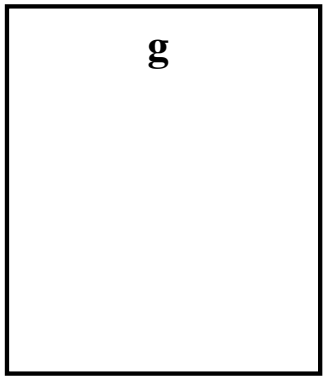
Third Refinement: a Global View

INITIAL SITUATION

SENDER

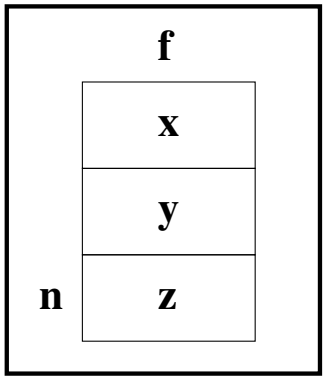


RECEIVER

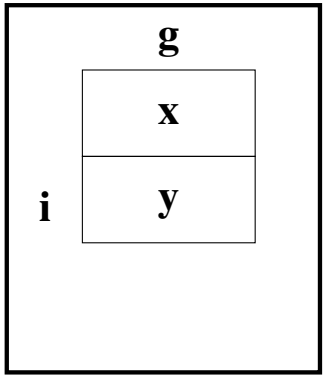


FINAL SITUATION

SENDER



RECEIVER



Third Refinement

- In this refinement, we consider requirements FUN-1 to FUN-3

The goal of the BRP is to totally or partially transfer a certain non-empty original sequential file from one site to another.	FUN_1

A total transfer means that the transmitted file is a copy of the original one.	FUN_2

A partial transfer means that the transmitted file is a genuine prefix of the original one.	FUN_3

- We also take account of requirement FUN-7 and FUN-8

The State (1)

- First, we define sequential file f to be transmitted

sets: D

constants: n, f

axm0_1: $0 < n$

axm0_2: $f \in 1 \mathinner{..} n \to D$

The State (2)

- The transmitted file is denoted by a variable g of length r.
- Invariant **inv3_2** states that the transmitted file is a prefix of f.
- Invariant **inv3_3** states when the receiver succeeds.

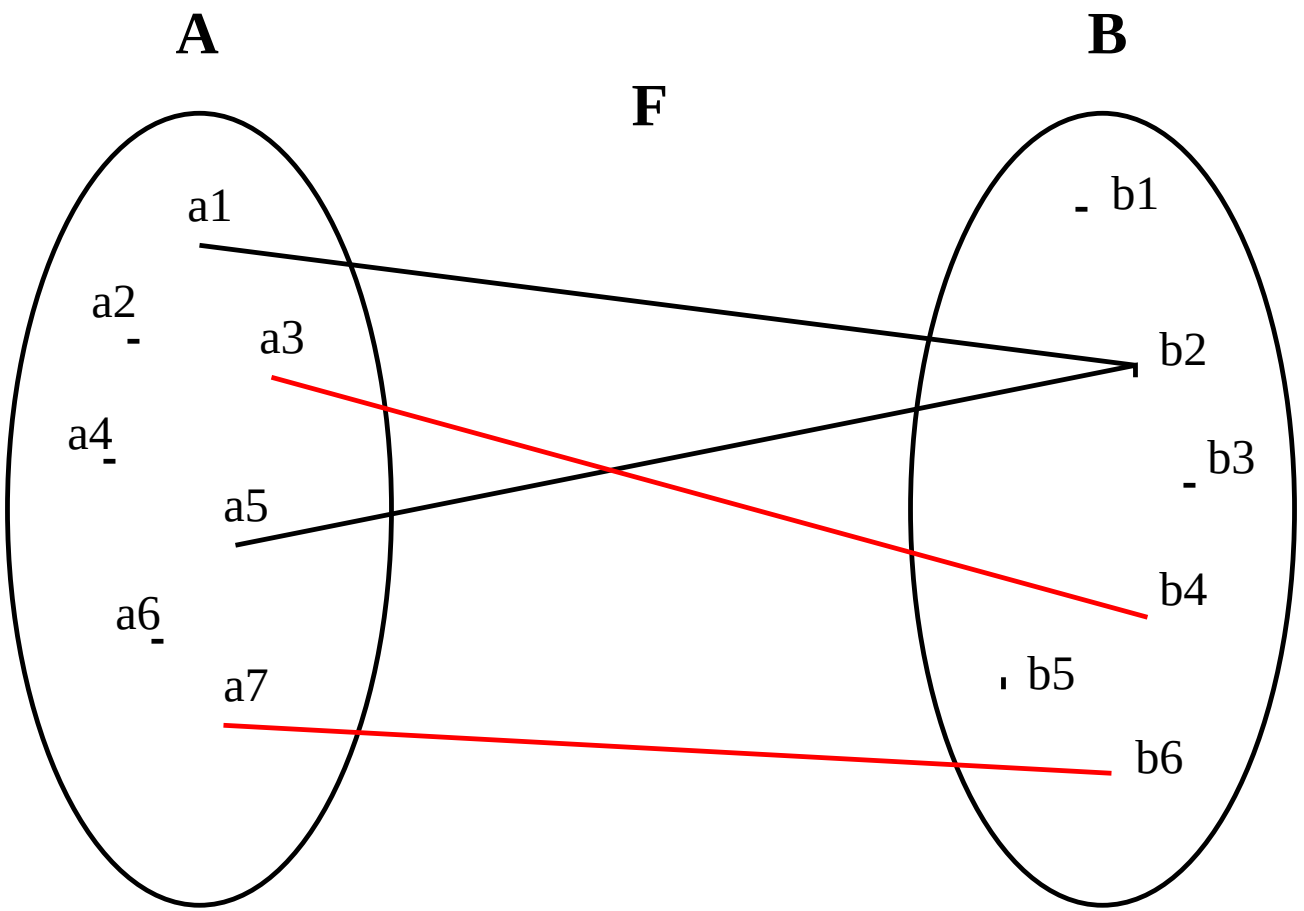
variables: r, g

inv3_1: $r \in 0 \,..\, n$

inv3_2: $g = 1 \,..\, r \lhd f$

inv3_3: $r\_st = success \;\Leftrightarrow\; r = n$

Refresher: The Domain Restriction Operator



$$\{a3,\ a7\} \lhd F$$

The Events

- New Event RCV_rcv_current_data and refined event RCV_success both cheat

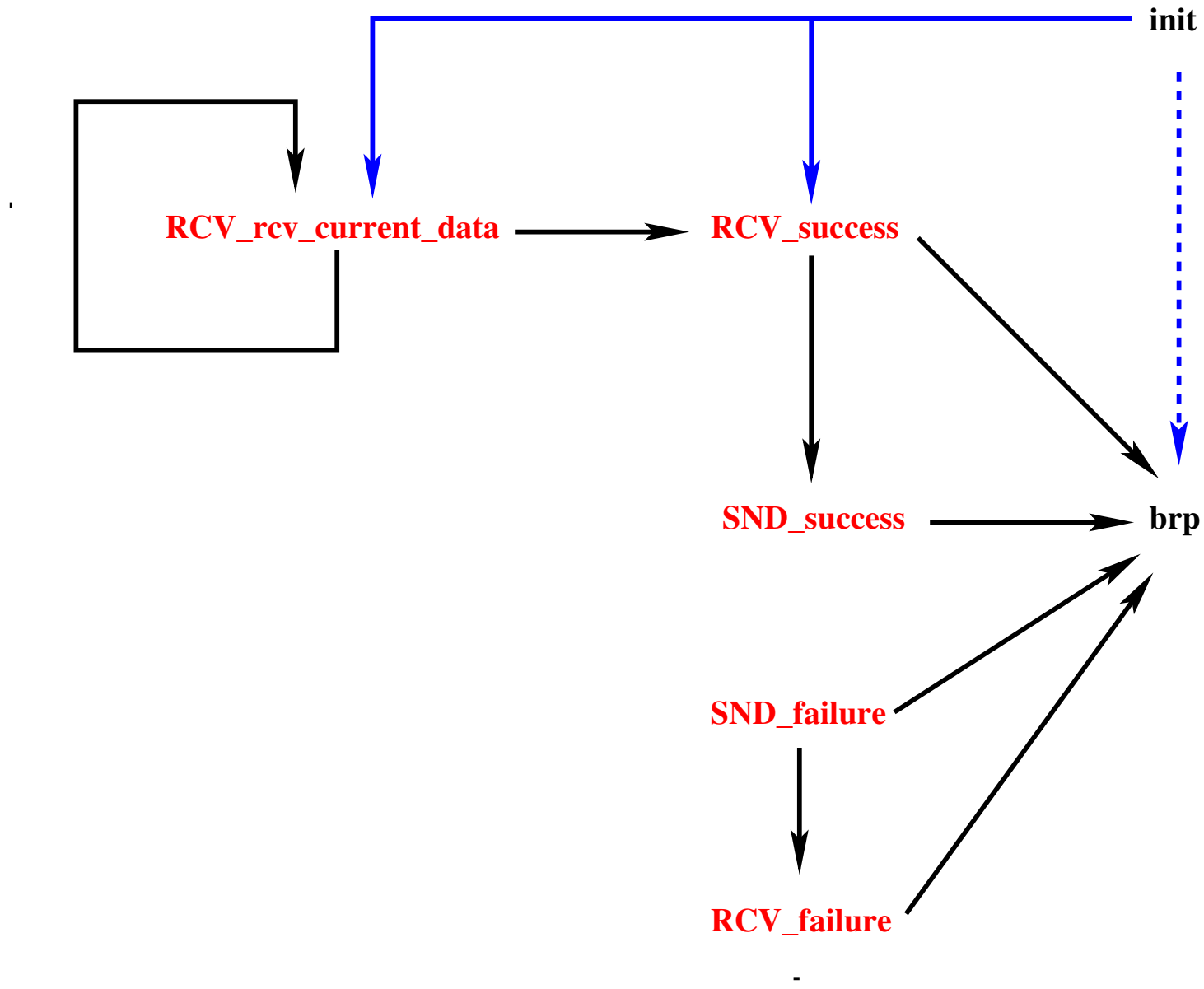
- They access the original file f and its length n (situated on the sender site)

RCV_rcv_current_data
status
convergent
when
$r\_st = working$
$r + 1 < n$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto f(r + 1)\}$
end

RCV_success
when
$r\_st = working$
$r + 1 = n$
then
$r\_st := success$
$r := r + 1$
$g := g \cup \{r + 1 \mapsto f(n)\}$
end

variant3: $n - r$

Synchronization of the Events



Fourth Refinement: the State

- The state is first enlarged with an activation bit w.
- When w is TRUE, it means that the sender event can be activated.
- The state is also enlarged with the sender pointer s
- It is such that $s+1$ points the next item to be sent, $f(s+1)$
- The state is further enlarged with the data container d
- d is equal to $f(s+1)$ when the data channel is active ($w = \mathrm{FALSE}$)

variables: $\ldots$, w, s, d

inv4_1: $s \in 0 \,..\, n-1$

inv4_2: $r \in s \,..\, s+1$

inv4_3: $w = \mathrm{FALSE} \Rightarrow d = f(s+1)$

Fourth Refinement: the Initialisation Event

Events brp, SND_failure, and RCV_failure are not modified.

init

$r := 0$

$g := \varnothing$

$r\_st := working$

$s\_st := working$

$w := \mathrm{TRUE}$

$s := 0$

$d :\in D$

Fourth Refinement: a New Event

- The next event SND_snd_data is new.

- It corresponds to the main action of the sender,

- It prepares the information to be sent through: d and s.

SND_snd_data
when
$s\_st = working$
$w = \mathrm{TRUE}$
then
$d := f(s+1)$
$w := \mathrm{FALSE}$
end

Fourth Refinement: Events on the Receiver Side

RCV_rcv_current_data
when
$r\_st = working$
$w = \mathrm{FALSE}$
$r = s$
$r + 1 < n$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
end

RCV_success
when
$r\_st = working$
$w = \mathrm{FALSE}$
$r = s$
$r + 1 = n$
then
$r\_st := success$
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
end

Notice that the receiver is still cheating: it accesses n and w

Fourth Refinement: Events on the Sender side

- SND_rcv_current_ack is a new event

SND_rcv_current_ack
when
$s\_st = working$
$w = \mathrm{FALSE}$
$s + 1 < n$
$r = s + 1$
then
$w := \mathrm{TRUE}$
$s := s + 1$
end

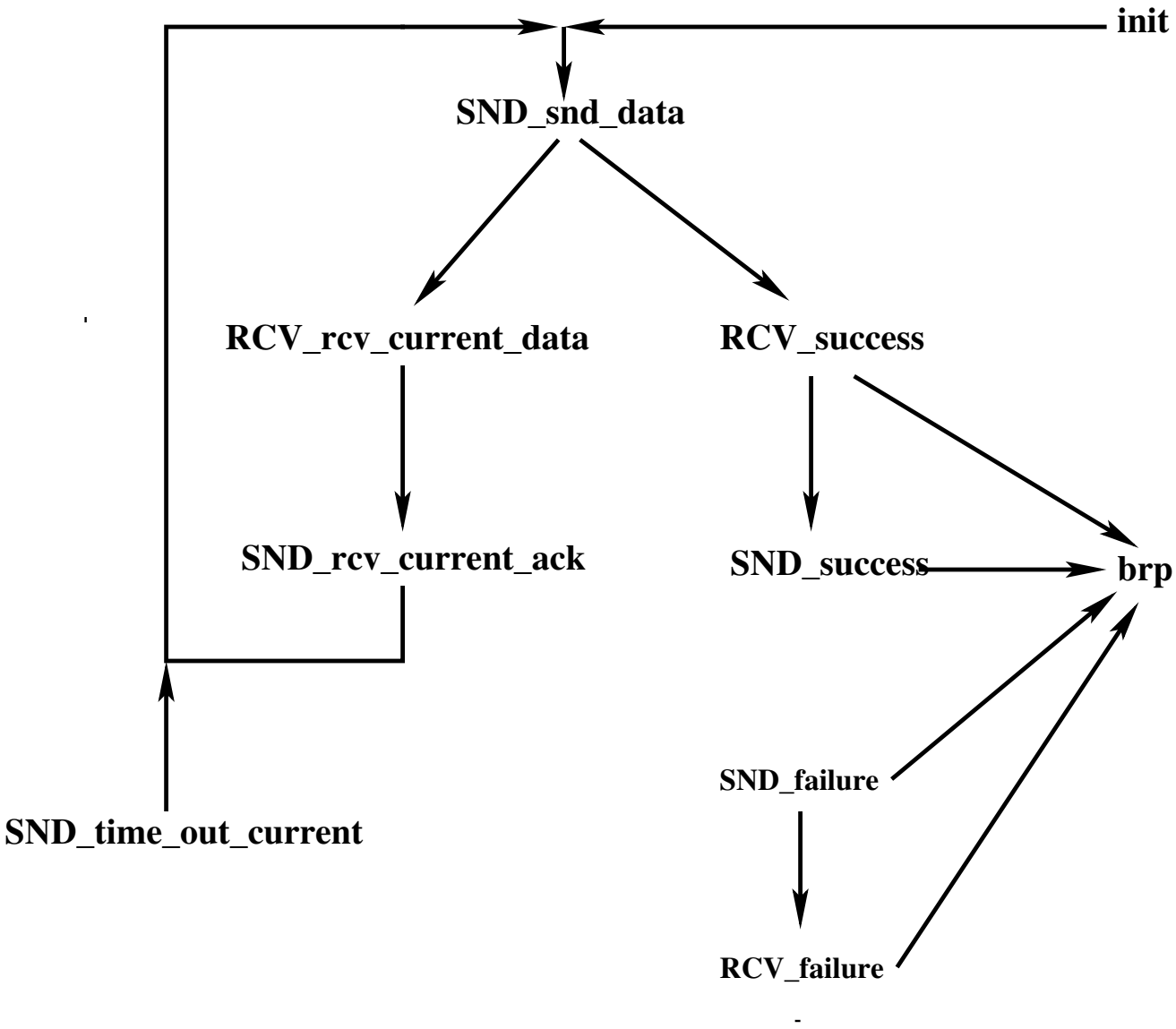
SND_success
when
$s\_st = working$
$w = \mathrm{FALSE}$
$s + 1 = n$
$r = s + 1$
then
$s\_st := success$
end

Fourth Refinement: a Special Event

- This event will receive an explanation in the next refinement

SND_time_out_current
when
$s\_st = working$
$w = \mathrm{FALSE}$
then
$w := \mathrm{TRUE}$
end

Fourth Refinement: Synchronization of the Events



Fifth Refinement: Introducing more Activation Bits

- We introduce more activation bits: db, ab, v.

- At most one activation bit is TRUE at a time

variables: $\ldots$, db, ab, v

inv3_1: $w = \mathrm{TRUE} \Rightarrow db = \mathrm{FALSE}$

inv3_2: $w = \mathrm{TRUE} \Rightarrow ab = \mathrm{FALSE}$

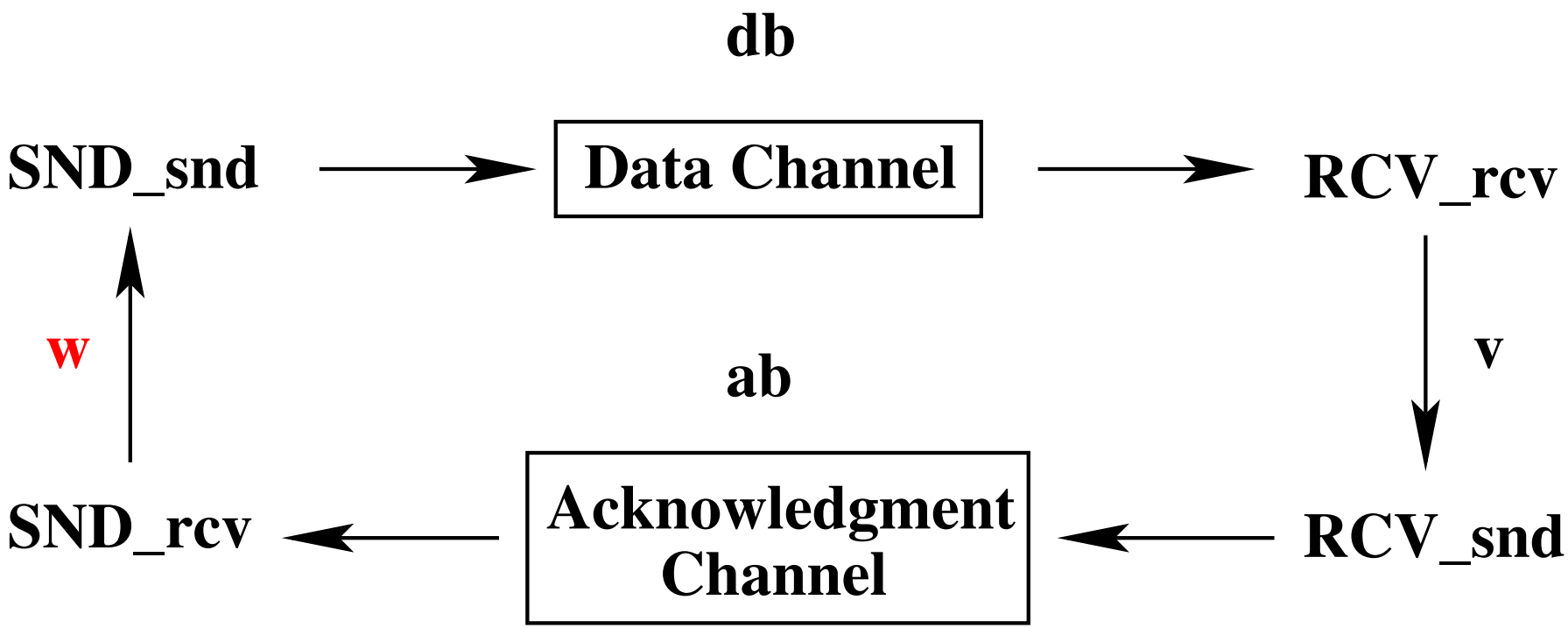
inv3_3: $w = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

inv3_4: $db = \mathrm{TRUE} \Rightarrow ab = \mathrm{FALSE}$

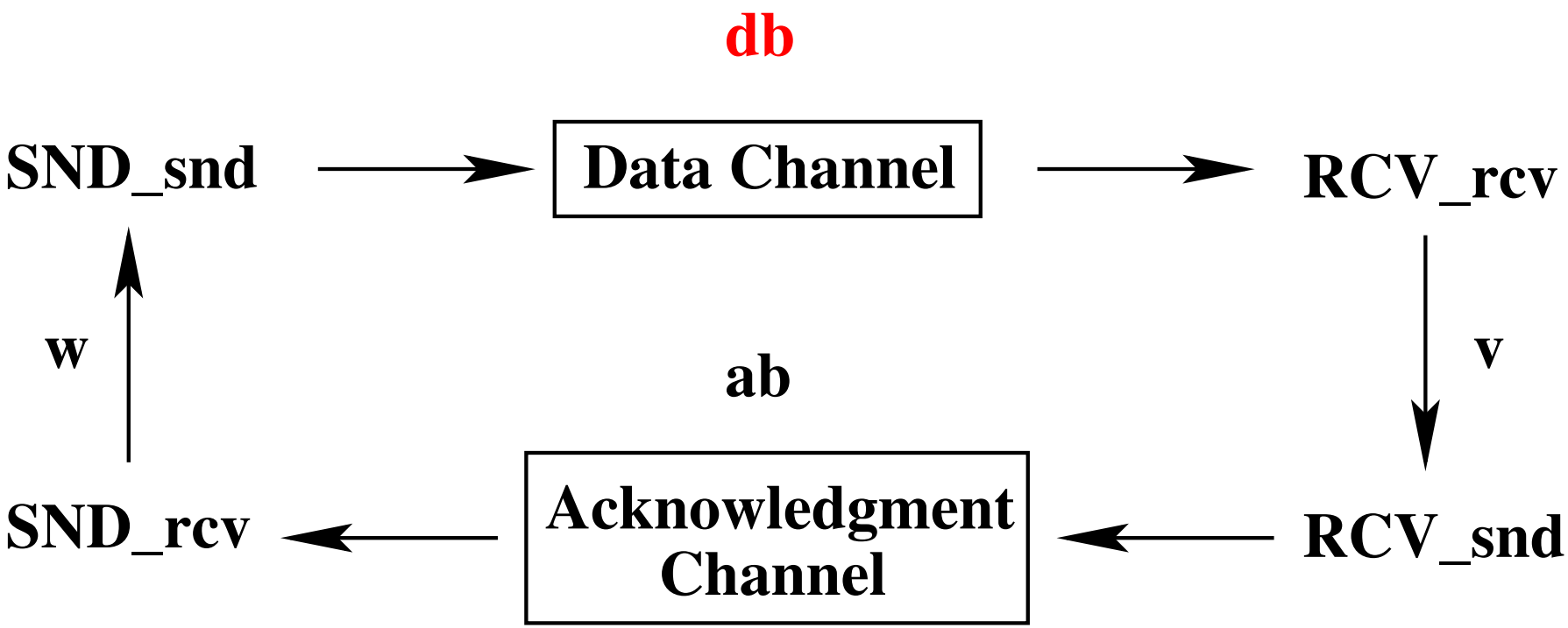
inv3_5: $db = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

inv3_6: $ab = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

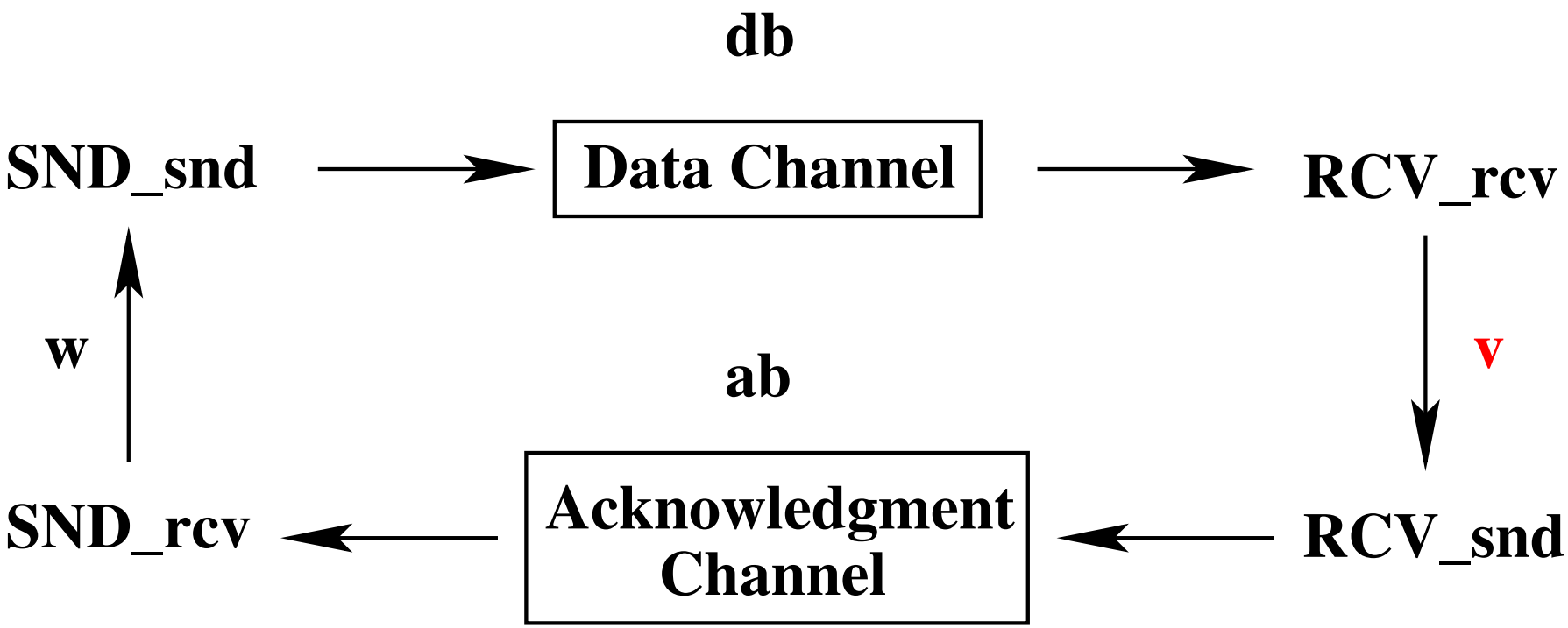
Activation bits at work



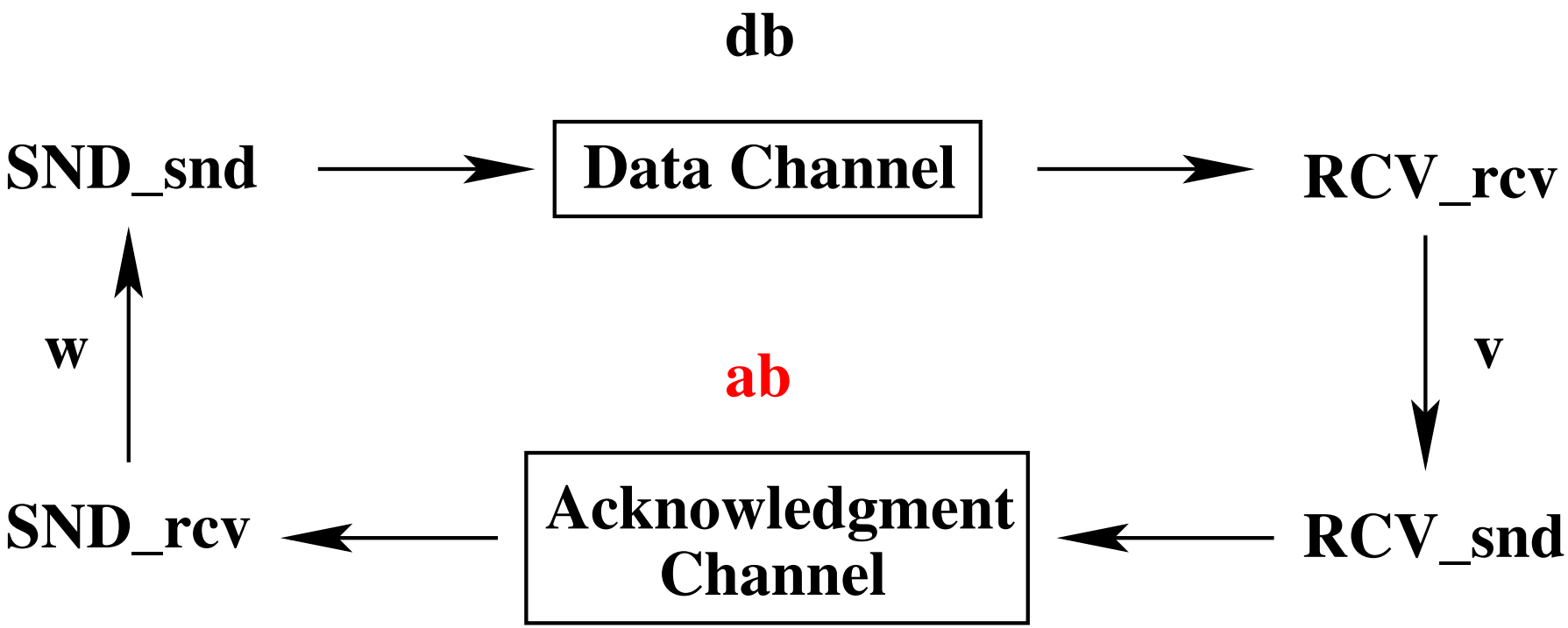
Activation bits at work



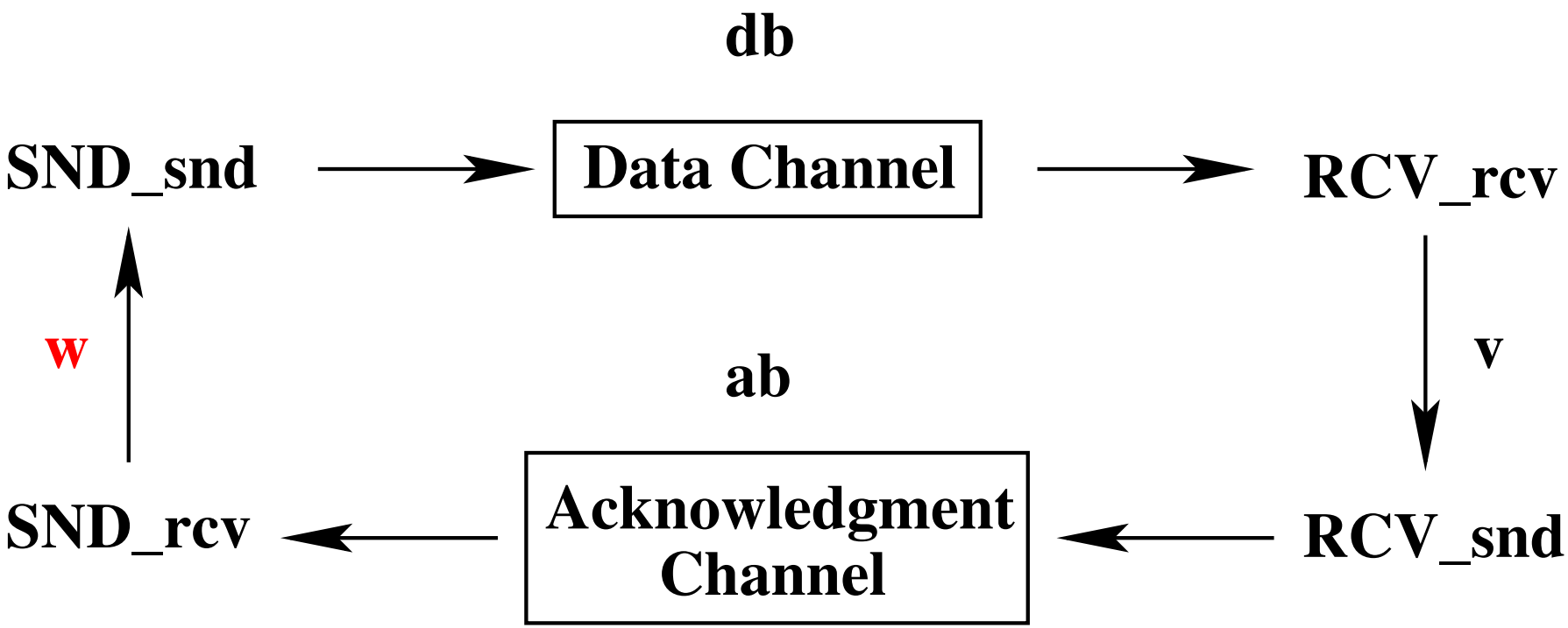
Activation bits at work

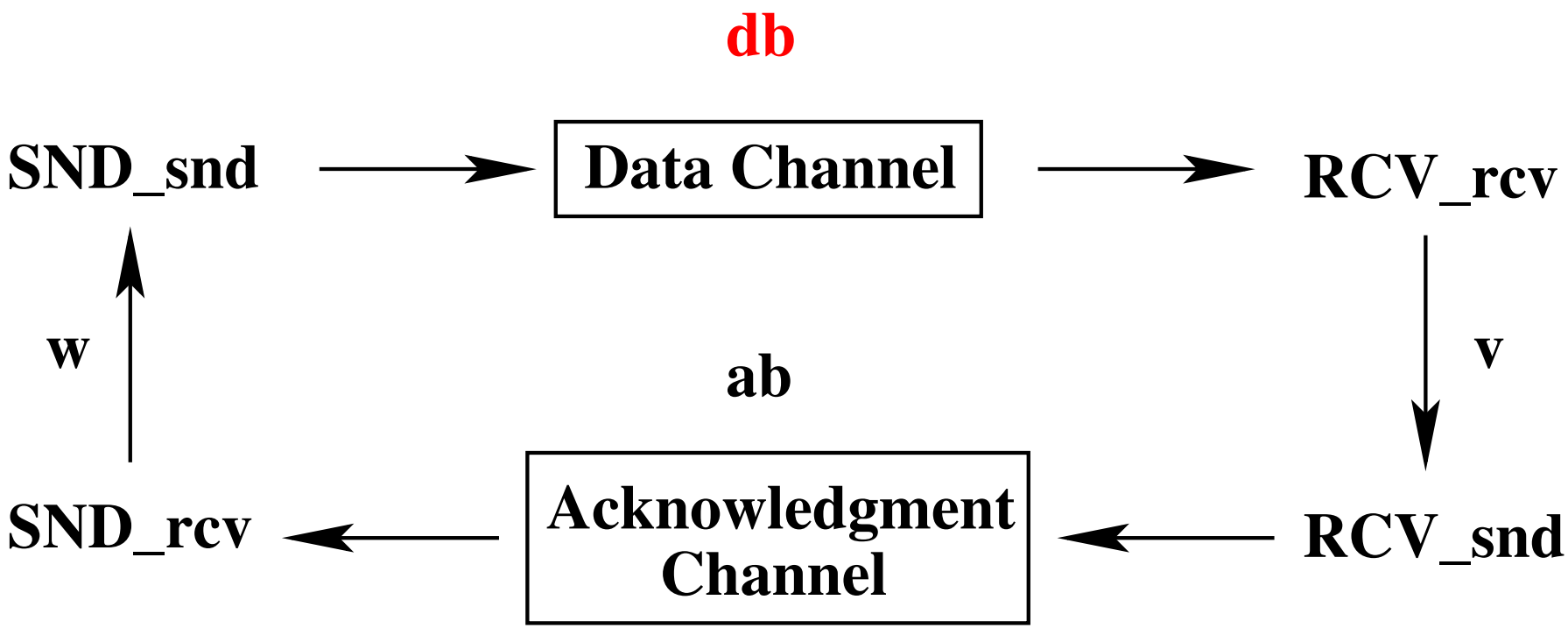


Activation bits at work

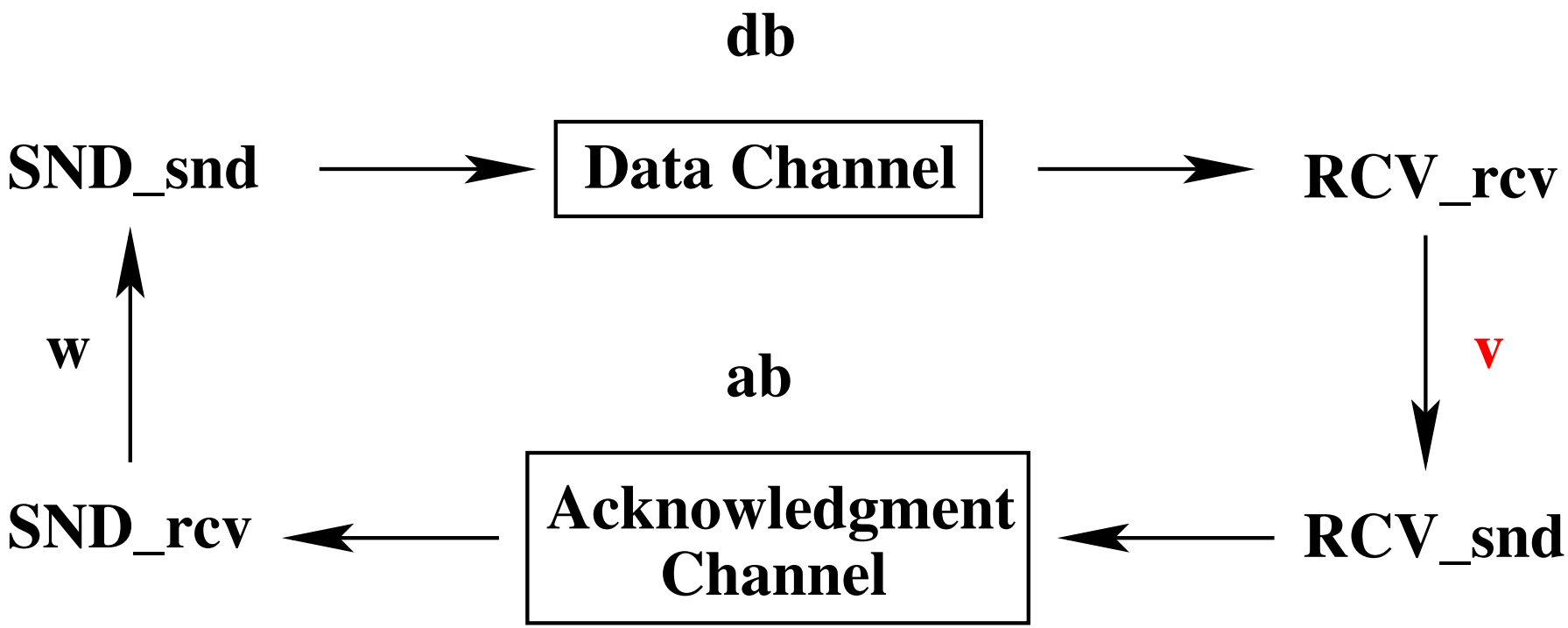


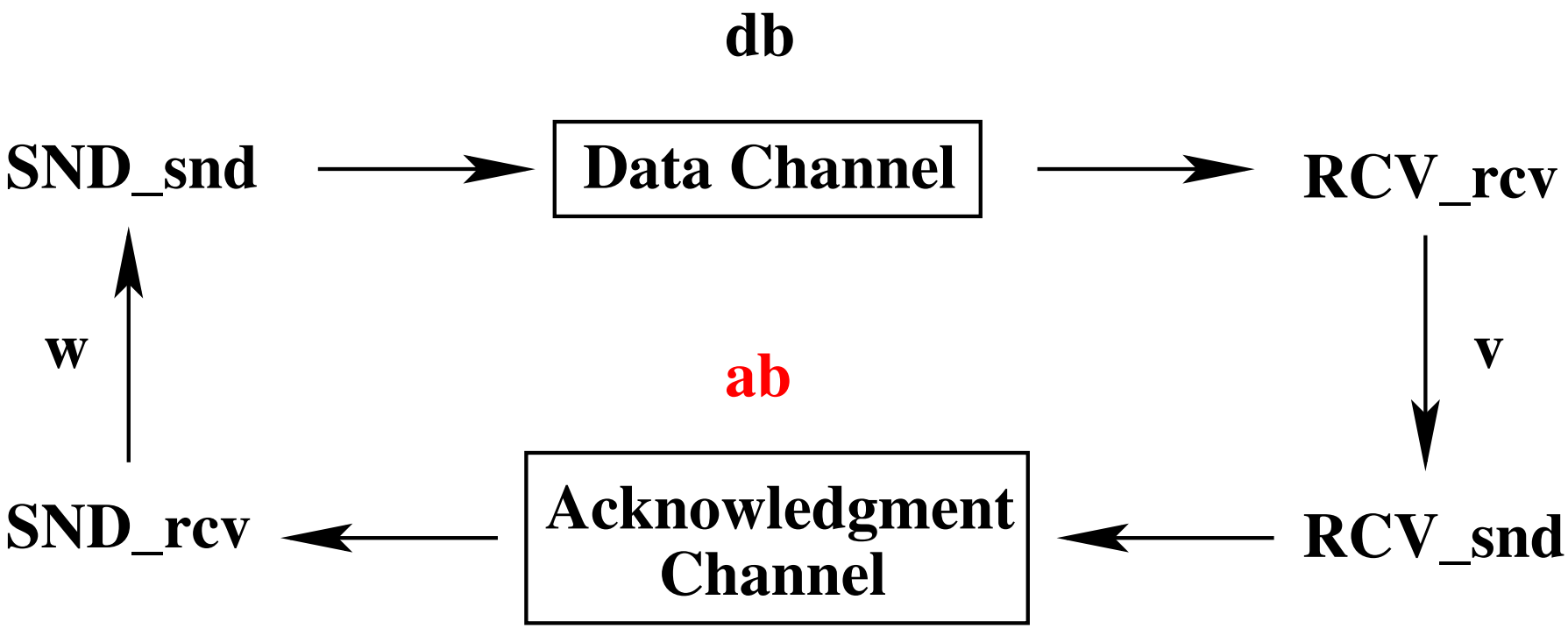
Activation bits at work




db
SND_snd
Data Channel
RCV_rcv
w
ab
v
SND_rcv
Acknowledgment Channel
RCV_snd

Activation bits at work




db
SND_snd
Data Channel
RCV_rcv
w
ab
v
SND_rcv
Acknowledgment Channel
RCV_snd

Fifth Refinement: Introducing the Last Item Indicator

- We introduce the last data indicator

variables: $\ldots$, l

- Together with these invariants (more explanation to come):

inv3_7: $db = \mathrm{TRUE} \ \wedge \ r = s \ \wedge \ l = \mathrm{FALSE} \ \Rightarrow \ r + 1 < n$

inv3_8: $db = \mathrm{TRUE} \ \wedge \ r = s \ \wedge \ l = \mathrm{TRUE} \ \Rightarrow \ r + 1 = n$

- This bit is sent by the Sender to the Receiver

- When equal to TRUE, this bit indicates that the sent item is the last one

Fifth Refinement: Introducing the Retry Counter c

- Constant MAX denotes the maximum number of retries

- The sender fails iff the retry counter c exceeds MAX (**inv3_10**)

constants: $\ldots$
MAX

axm3_1: $MAX \in \mathbb{N}$

variables: $\ldots$
c

inv3_9: $c \in 0 \mathinner{..} MAX + 1$

inv3_10: $c = MAX + 1 \Leftrightarrow s\_st = failure$

Fifth Refinement: the Events (1)

init

$r := 0$
$g := \varnothing$
$r\_st := working$
$s\_st := working$
$s := 0$
$d :\in D$
$w := \mathrm{TRUE}$
$db := \mathrm{FALSE}$
$ab := \mathrm{FALSE}$
$v := \mathrm{FALSE}$
$l := \mathrm{FALSE}$
$c := 0$

brp

when
$r \neq working$
$s \neq working$
then
skip
end

Fifth Refinement: the Events (2)

SND_snd_current_data

when

$s\_st = working$

$w = \mathrm{TRUE}$

$s + 1 < n$

then

$d := f(s + 1)$

$w := \mathrm{FALSE}$

$db := \mathrm{TRUE}$

$l := \mathrm{FALSE}$

end

SND_snd_last_data

when

$s\_st = working$

$w = \mathrm{TRUE}$

$s + 1 = n$

then

$d := f(s + 1)$

$w := \mathrm{FALSE}$

$db := \mathrm{TRUE}$

$l := \mathrm{TRUE}$

end

Fifth Refinement: New Events

- Daemons are breaking the channels

DMN_data_channel
when
$db = \mathrm{TRUE}$
then
$db = \mathrm{FALSE}$
end

DMN_ack_channel
when
$ab = \mathrm{TRUE}$
then
$ab = \mathrm{FALSE}$
end

- A failure is characterized by all activation bits being FALSE

Fifth Refinement: the Events (3)

SND_time_out_current
when
$s\_st = working$
$w = \mathbf{FALSE}$
$ab = \mathbf{FALSE}$
$db = \mathbf{FALSE}$
$v = \mathbf{FALSE}$
$c < MAX$
then
$w := \mathbf{TRUE}$
$c := c + 1$
end

SND_failure
when
$s\_st = working$
$w = \mathbf{FALSE}$
$ab = \mathbf{FALSE}$
$db = \mathbf{FALSE}$
$v = \mathbf{FALSE}$
$c = MAX$
then
$s\_st := failure$
$c := c + 1$
end

- Sender aborts after $MAX + 1$ tries

Fifth Refinement: the Events (4)

RCV_rcv_current_data
when
$r\_st = working$
$db = \text{TRUE}$
$r = s$
$l = \text{FALSE}$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
$db := \text{FALSE}$
$v := \text{TRUE}$
end

RCV_success
when
$r\_st = working$
$db = \text{TRUE}$
$r = s$
$l = \text{TRUE}$
then
$r\_st := success$
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
$db := \text{FALSE}$
$v := \text{TRUE}$
end

Reminder: l is the last data indicator

Fifth Refinement: Guard Srengthening (1)

(abstract-)RCV_rcv_current_data
when
$r\_st = working$
$w = \text{FALSE}$
$r = s$
$r + 1 < n$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
end

(concrete-)RCV_rcv_current_data
when
$r\_st = working$
$db = \text{TRUE}$
$r = s$
$l = \text{FALSE}$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
$db := \text{FALSE}$
$v := \text{TRUE}$
end

inv3_1': $db = \text{TRUE} \Rightarrow w = \text{FALSE}$

inv3_7: $db = \text{TRUE} \wedge r = s \wedge l = \text{FALSE} \Rightarrow r + 1 < n$

Fifth Refinement: Guard Srengthening (2)

(abstract-)RCV_success
when
$r\_st = working$
$w = \mathrm{FALSE}$
$r = s$
$r + 1 = n$
then
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
end

(concrete-)RCV_success
when
$r\_st = working$
$db = \mathrm{TRUE}$
$r = s$
$l = \mathrm{TRUE}$
then
$r\_st := success$
$r := r + 1$
$g := g \cup \{r + 1 \mapsto d\}$
$db := \mathrm{FALSE}$
$v := \mathrm{TRUE}$
end

inv3_1': $db = \mathrm{TRUE} \Rightarrow w = \mathrm{FALSE}$

inv3_8: $db = \mathrm{TRUE} \wedge r = s \wedge l = \mathrm{TRUE} \Rightarrow r + 1 = n$

Fifth Refinement: the Events (5)

RCV_rcv_retry
when
$db = \mathrm{TRUE}$
$r \neq s$
then
$db := \mathrm{FALSE}$
$v := \mathrm{TRUE}$
end

RCV_snd_ack
when
$v = \mathrm{TRUE}$
then
$v := \mathrm{FALSE}$
$ab := \mathrm{TRUE}$
end

RCV_failure
when
$r\_st = working$
$c = MAX + 1$
then
$r\_st := failure$
end

Fifth Refinement: the Events (6)

SND_rcv_current_ack

when

$s\_st = working$

$ab = \mathrm{TRUE}$

$s + 1 < n$

then

$w := \mathrm{TRUE}$

$s := s + 1$

$c := 0$

$ab := \mathrm{FALSE}$

end

SND_success

when

$s\_st = working$

$ab = \mathrm{TRUE}$

$s + 1 = n$

then

$s\_st := success$

$c := 0$

$ab := \mathrm{FALSE}$

end

Fifth Refinement: Guard Strengthening (1)

(abstract-)SND_rcv_current_ack
when
$s\_st = working$
$w = \text{FALSE}$
$s + 1 < n$
$r = s + 1$
then
$w := \text{TRUE}$
$s := s + 1$
end

(concrete-)SND_rcv_current_ack
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 < n$
then
$w := \text{TRUE}$
$s := s + 1$
$c := 0$
$ab := \text{FALSE}$
end

inv3_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

- In order to prove guard strengthening we need invariant **inv3_11**

inv3_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

inv3_12: $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv3_12** is needed to prove **inv3_11**

Fifth Refinement: Guard Strengthening (2)

(abstract-)SND_success
when
$s\_st = working$
$w = \text{FALSE}$
$s + 1 = n$
$r = s + 1$
then
$s\_st := success$
end

(concrete-)SND_success
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 = n$
then
$s\_st := success$
$c := 0$
$ab := \text{FALSE}$
end

inv3_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

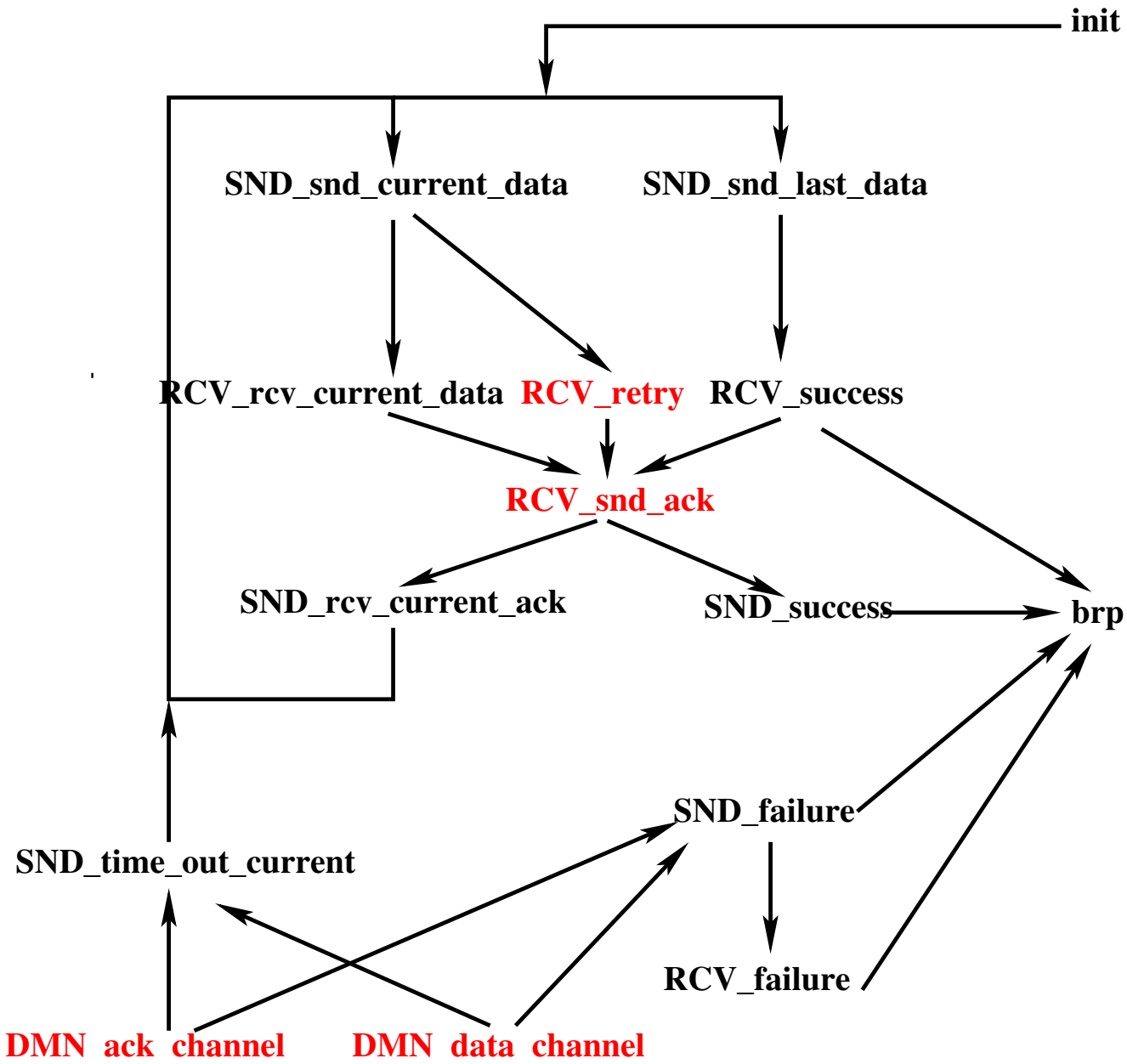
- In order to prove guard strengthening we need invariant **inv3_11**

inv3_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

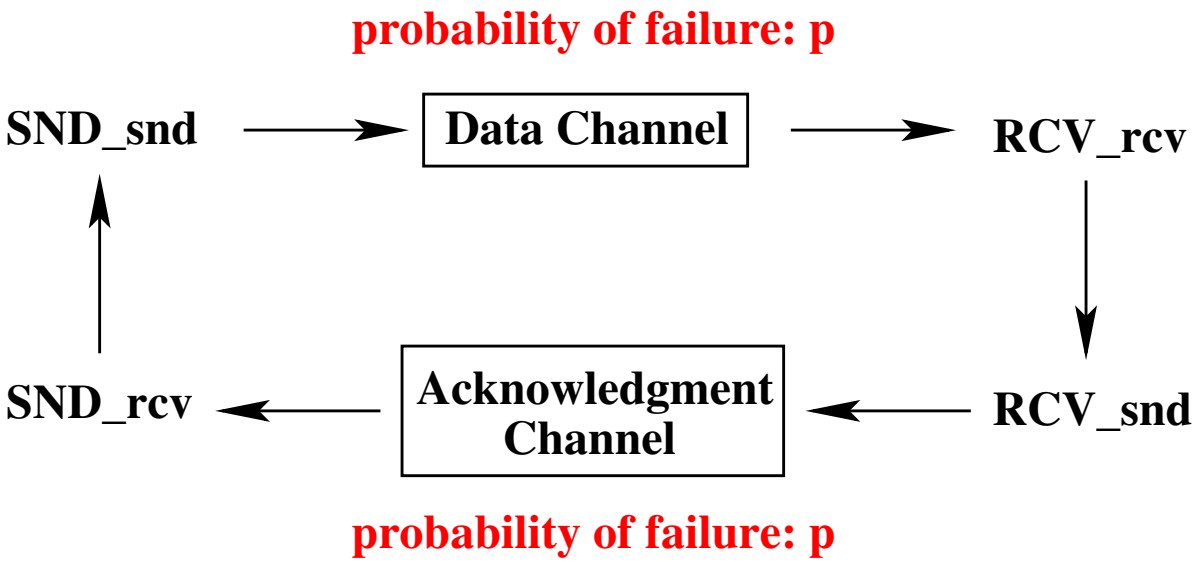
inv3_12: $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv3_12** is needed to prove **inv3_11**

Final Synchronization of the Events



Computing Probabilities



- We would like to compute the probability of success
- It is a function of:
  - p: probability of failure for one channel
  - n: size of the file
  - $MAX + 1$: number of re-tries

Failure on one channel p

Computing Probabilities

Failure on one channel	p
Success on one channel	$1 - p$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Fails on one try	$1 - (1 - p)^2$
Fails on $MAX + 1$ tries	$(1 - (1 - p)^2)^{MAX+1}$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1-p$
Success on both channels	$(1-p)^2$
Fails on one try	$1-(1-p)^2$
Fails on $MAX+1$ tries	$(1-(1-p)^2)^{MAX+1}$
Succ. on $MAX+1$ tries	$1-(1-(1-p)^2)^{MAX+1}$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1-p$
Success on both channels	$(1-p)^2$
Fails on one try	$1-(1-p)^2$
Fails on $MAX+1$ tries	$(1-(1-p)^2)^{MAX+1}$
Succ. on $MAX+1$ tries	$1-(1-(1-p)^2)^{MAX+1}$
Success for n data	$(1-(1-(1-p)^2)^{MAX+1})^n$

Computing Probabilities

Failure on one channel	p
Success on one channel	$1-p$
Success on both channels	$(1-p)^2$
Fails on one try	$1-(1-p)^2$
Fails on $MAX+1$ tries	$(1-(1-p)^2)^{MAX+1}$
Succ. on $MAX+1$ tries	$1-(1-(1-p)^2)^{MAX+1}$
Success for n data	$(1-(1-(1-p)^2)^{MAX+1})^n$

$p = .1$
$MAX = 5$
$n = 100$

$.995$