6. The Bounded Re-transmission Protocol

Jean-Raymond Abrial

2009

Purpose of this Lecture

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

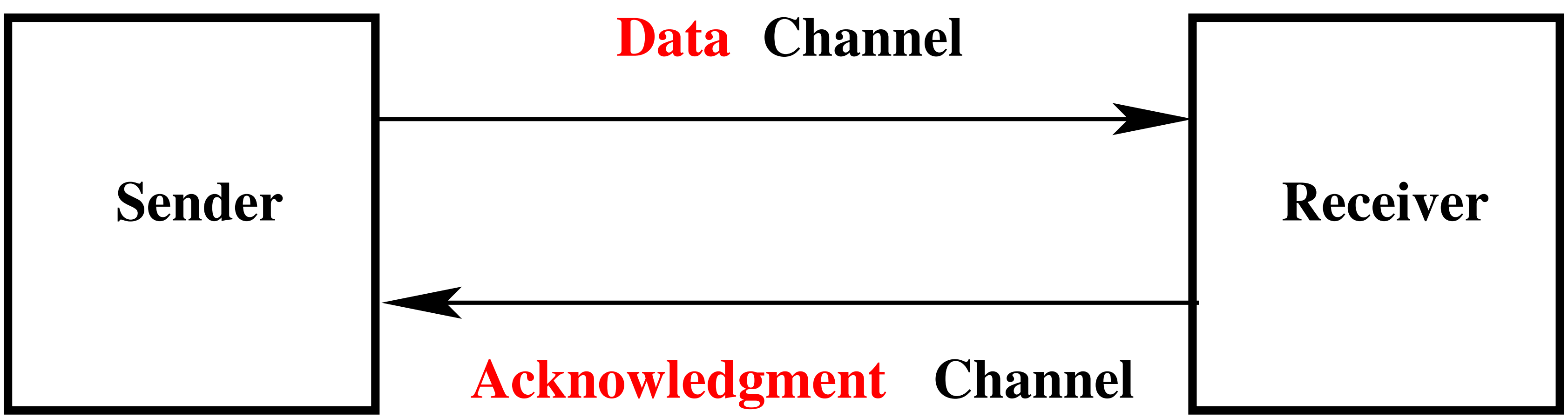
- This is a problem dealing with fault tolerance

- 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

- 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

- 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 re-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)

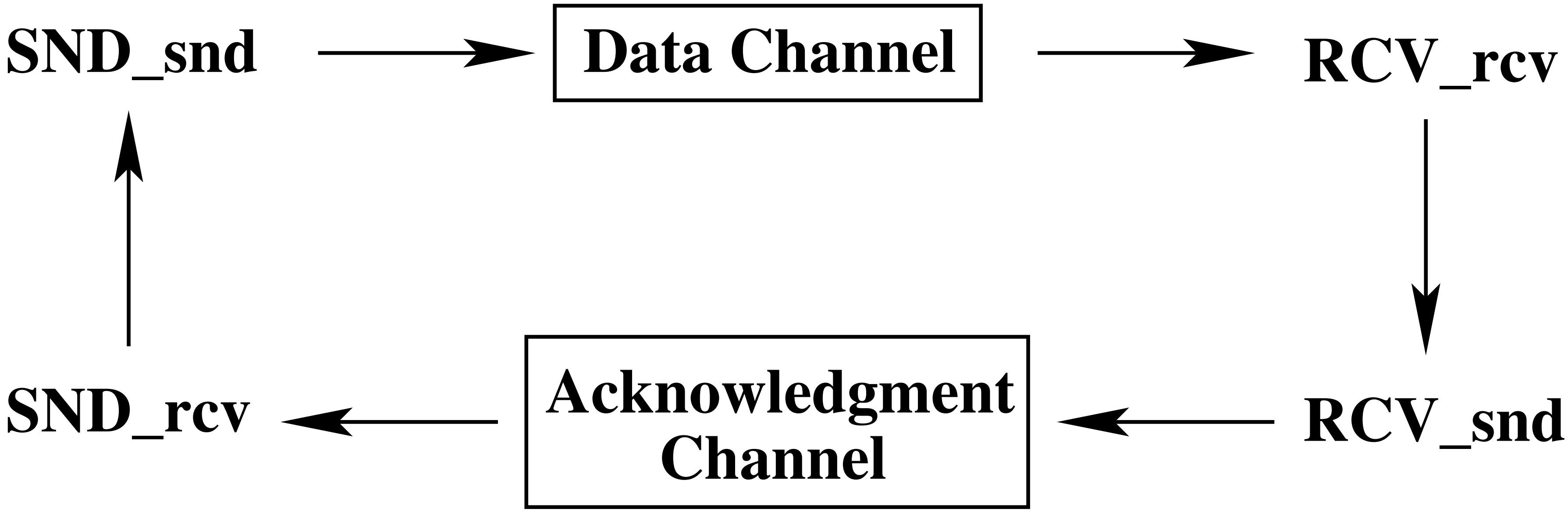
Each site may end up in any of the two situations: - either it believes that the protocol has terminated successfully, - or it believes that the protocol has aborted	FUN_4

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, protocol 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

(0) FUN_4: Defining the final "belief" situation

(1) and (2) FUN_5 and FUN_6: Connecting the "beliefs"

(3) FUN_1 to FUN_3, FUN_7 and FUN_8: Partial Transmission and final situation of the Receiver

(4) Introducing the Sender

(5) Introducing unreliable channels and timers.

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

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

Our initial model deals with requirements 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: Defining Constants

set: $STATUS$

constants: $working$
$success$
$failure$

axm0_1: $STATUS \;=\; \{working, success, failure\}$

axm0_2: $working \neq success$

axm0_3: $working \neq failure$

axm0_4: $success \neq failure$

Initial Model: Variables and Invariants

- Variables s_st and r_st denote the status of the participants (Sender and Receiver respectively).

variables: s_st
r_st

inv0_1: $s\_st \in STATUS$

inv0_2: $r\_st \in STATUS$

Initial Model: Events

- Initially, both participants are working

- Event "brp" is an "oserver" fired when both participants are not working

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

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

Next are two 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

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

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

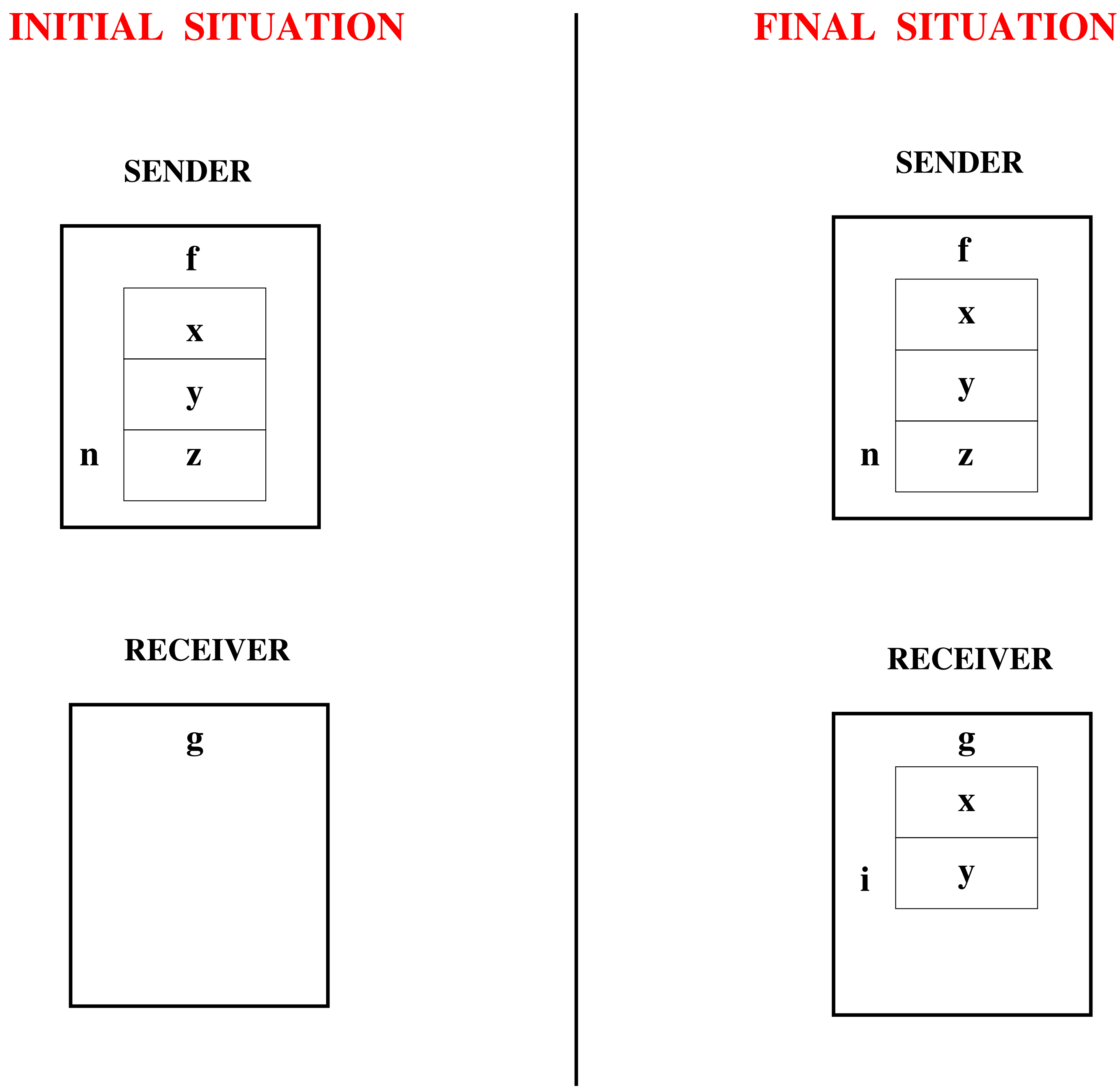
variant1: $\{success, failure\} \setminus \{s\_st\}$

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

The Sender and the Receiver: a First View



Third Refinement: More Constants

- Set D denotes the objects in the files

- Constant n denotes the size of the non-empty file

- Constant f denotes the original file.

set: D

constants: n
f

axm1_1: $0 < n$

axm1_2: $f \in 1 \,..\, n \rightarrow D$

Third Refinement: the Variables

- Variable r denotes the size of file g

- Variable g denotes the transmitted file.

variables: r
g

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

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

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

Third Refinement: the Events

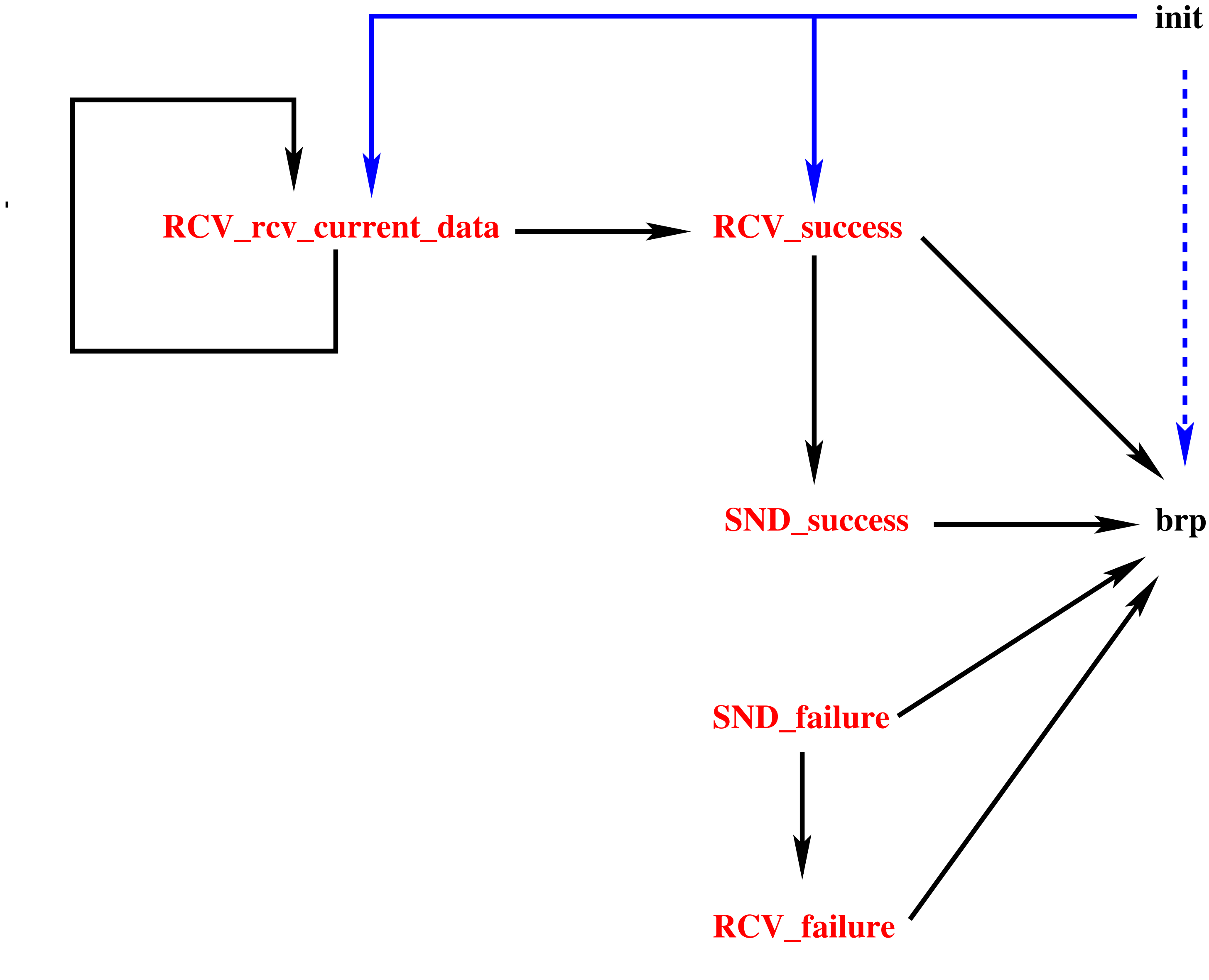
Both these events are cheating: they have access to $f(r+1)$, $f(n)$, and n.

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: $n - r$

Third Refinement: Synchronization



Fourth Refinement: Introducing the Sender (More Variables)

- Variable s is the Sender pointer sent to the Receiver
- Variable d is the data sent to the Receiver
- Variable w is the Sender activation bit
- When w is TRUE it means the Sender has just received the acknowledgement
- When w is FALSE it means the Sender has sent the information to the Receiver

variables: $\ldots$
w
s
d

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

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

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

The Main Communication

SND_snd_current_data $\longrightarrow$ (d, s) $\searrow$

$\uparrow$ RCV_rcv_current_data

SND_rcv_current_ack $\longleftarrow$ (r) $\swarrow$

Fourth Refinement: the Events (1)

init
- $r := 0$
- $g := \varnothing$
- $r\_st := working$
- $s\_st := working$
- $s := 0$
- $d :\in D$
- $w := \mathrm{TRUE}$

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

Fourth Refinement: the Events (2)

- New Events: the Sender prepares data d to be sent

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

- These events clearly refine skip and maintain invariant **inv4_3**

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

Fourth Refinement: the Events (3)

- The Receiver receives data d and pointer s. It sends pointer r.

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

- The Receiver still cheats: it accesses constant n and boolean w

Refinement of RCV_rcv_current_data

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

(concrete-)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

- Observe guard strengthening

- This invariant helps proving event refinement

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

Refinement of RCV_success

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

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

- Observe guard strengthening

- This invariant helps proving event refinement

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

Fourth Refinement: the Events (4)

- The first event is new. It clearly refines skip

- The activation bit is set to TRUE (activating the event "SND_snd_data")

- The Sender receives acknowledgment (pointer r)

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

Refinement of SND_success

(abstract-)SND_success
when
$s\_st = working$
$r\_st = success$
then
$s\_st := success$
end

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

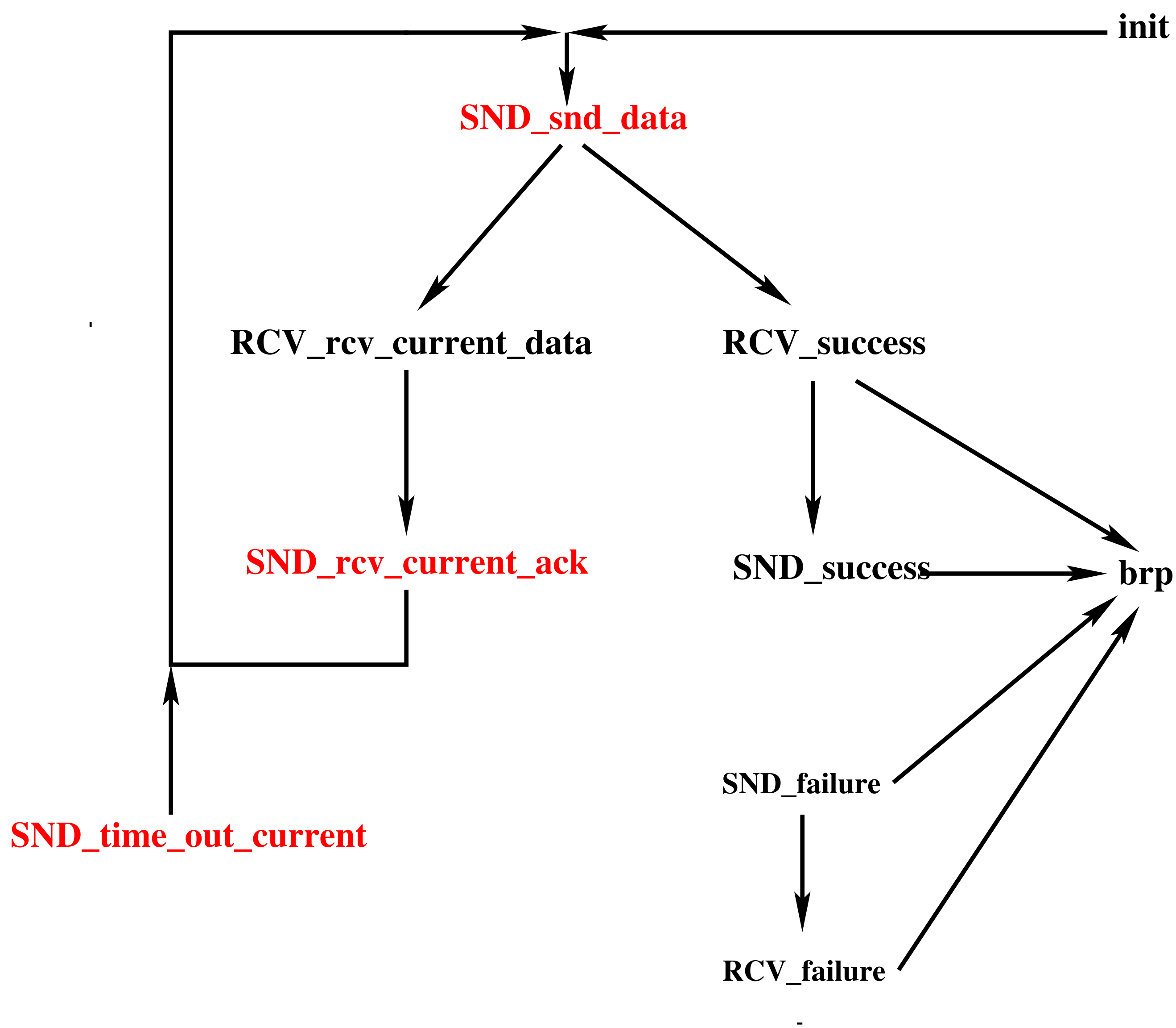
- The presence of **inv1_3** ensures that the guard is strengthen

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

- This new events will receive a full 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

- At most one activation bit is TRUE at a time

variables: $\ldots$
db
ab
v

inv5_1: $w = \text{TRUE} \Rightarrow db = \text{FALSE}$

inv5_2: $w = \text{TRUE} \Rightarrow ab = \text{FALSE}$

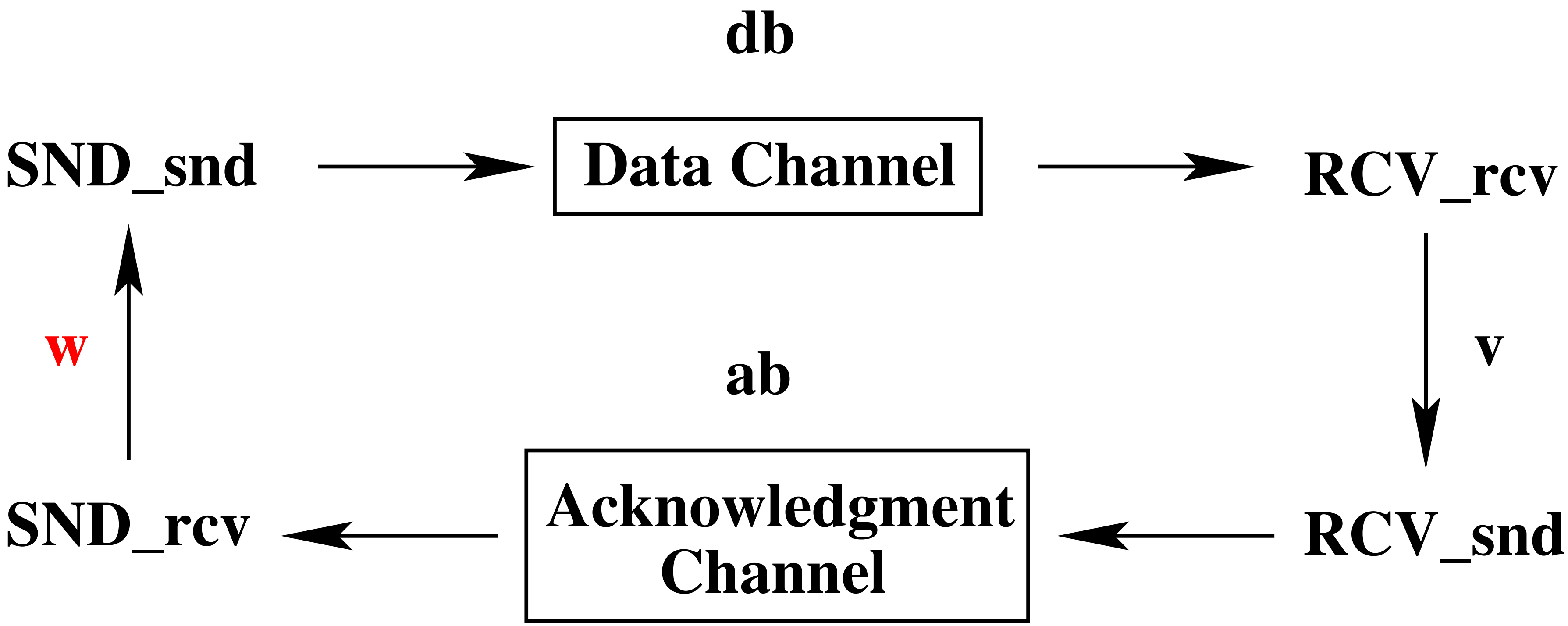
inv5_3: $w = \text{TRUE} \Rightarrow v = \text{FALSE}$

inv5_4: $db = \text{TRUE} \Rightarrow ab = \text{FALSE}$

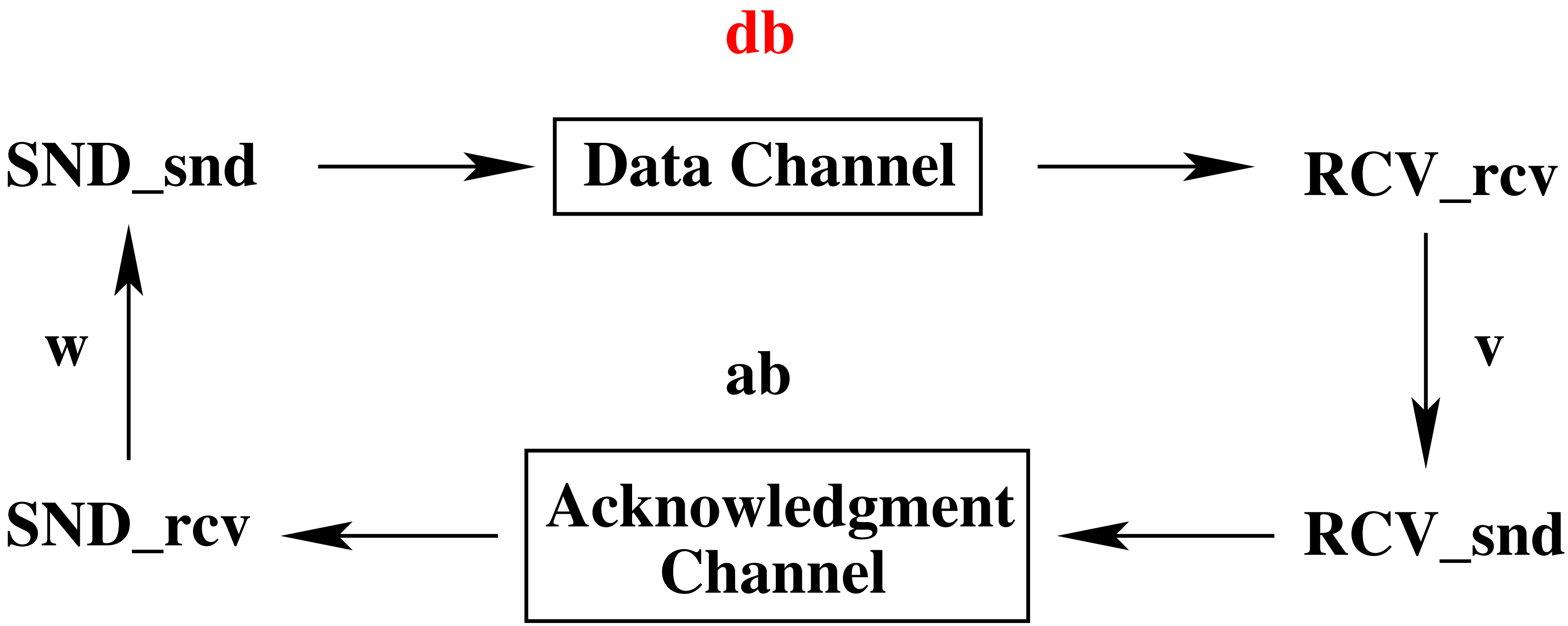
inv5_5: $db = \text{TRUE} \Rightarrow v = \text{FALSE}$

inv5_6: $ab = \text{TRUE} \Rightarrow v = \text{FALSE}$

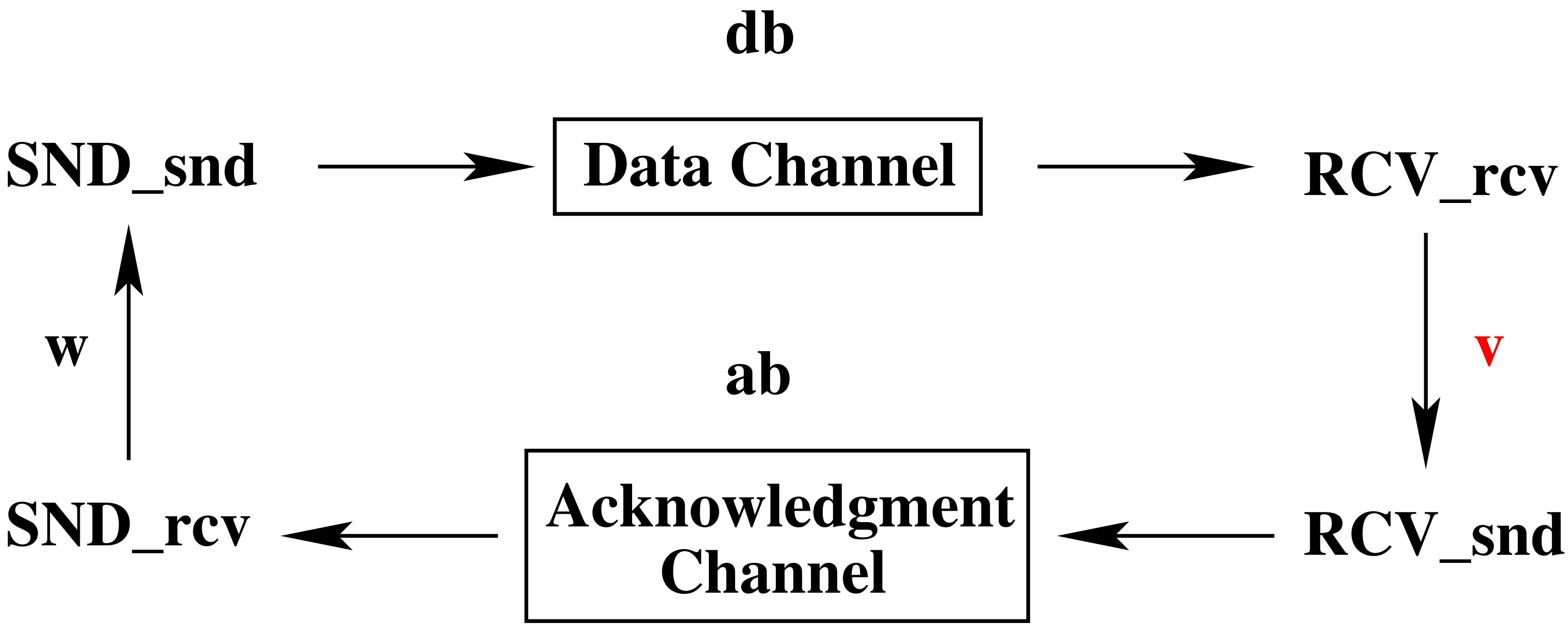
Activation bits at work



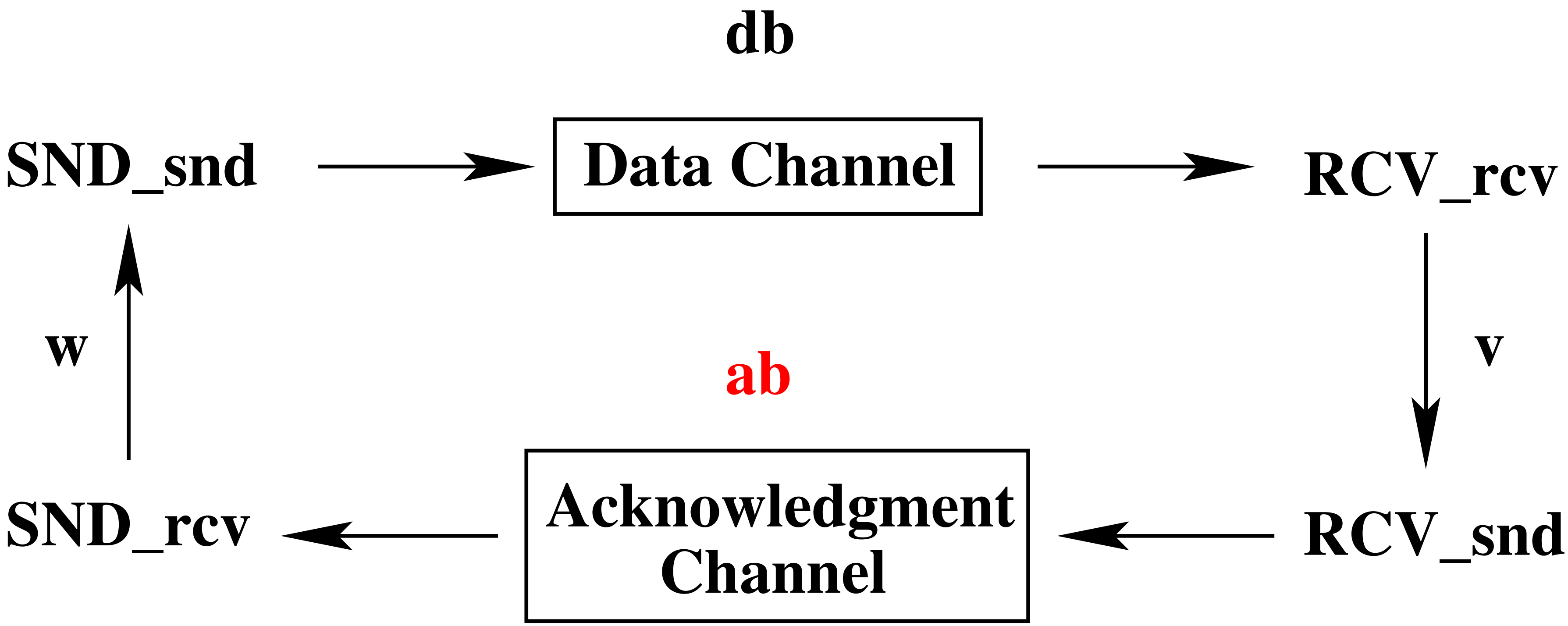
Activation bits at work



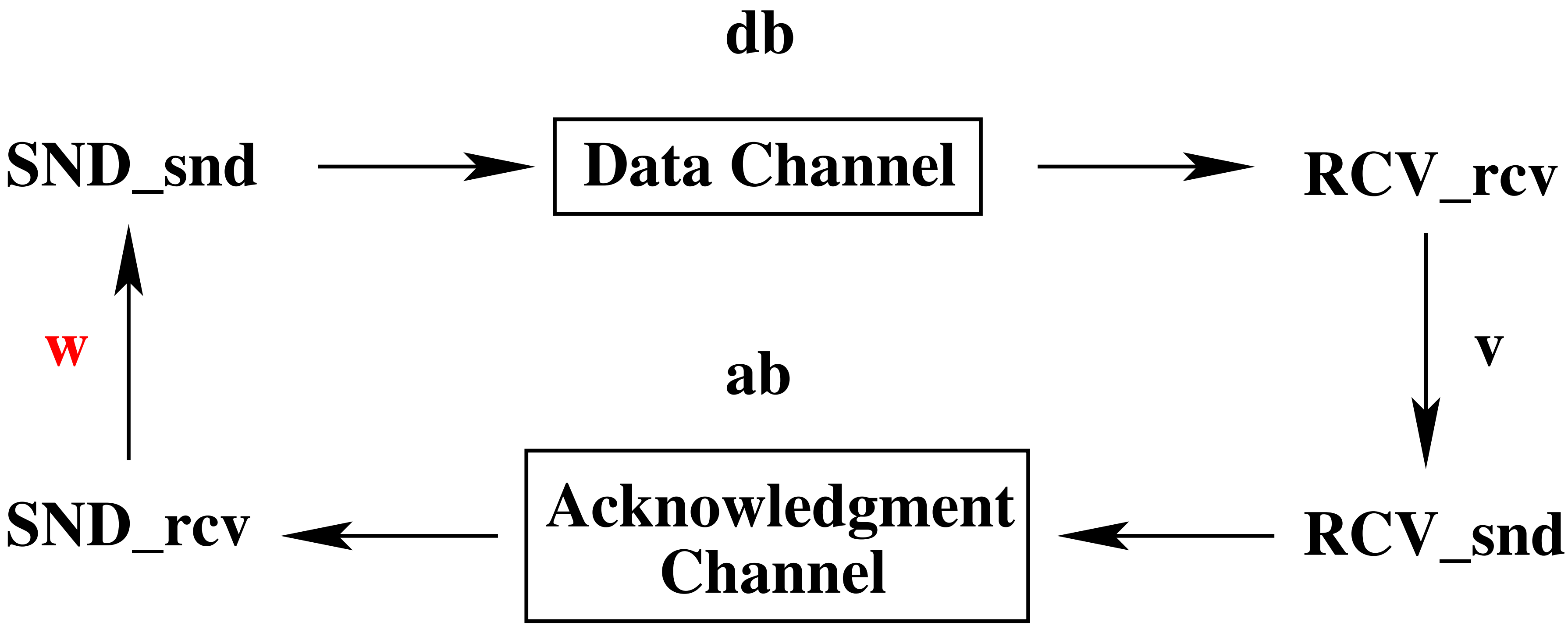
Activation bits at work



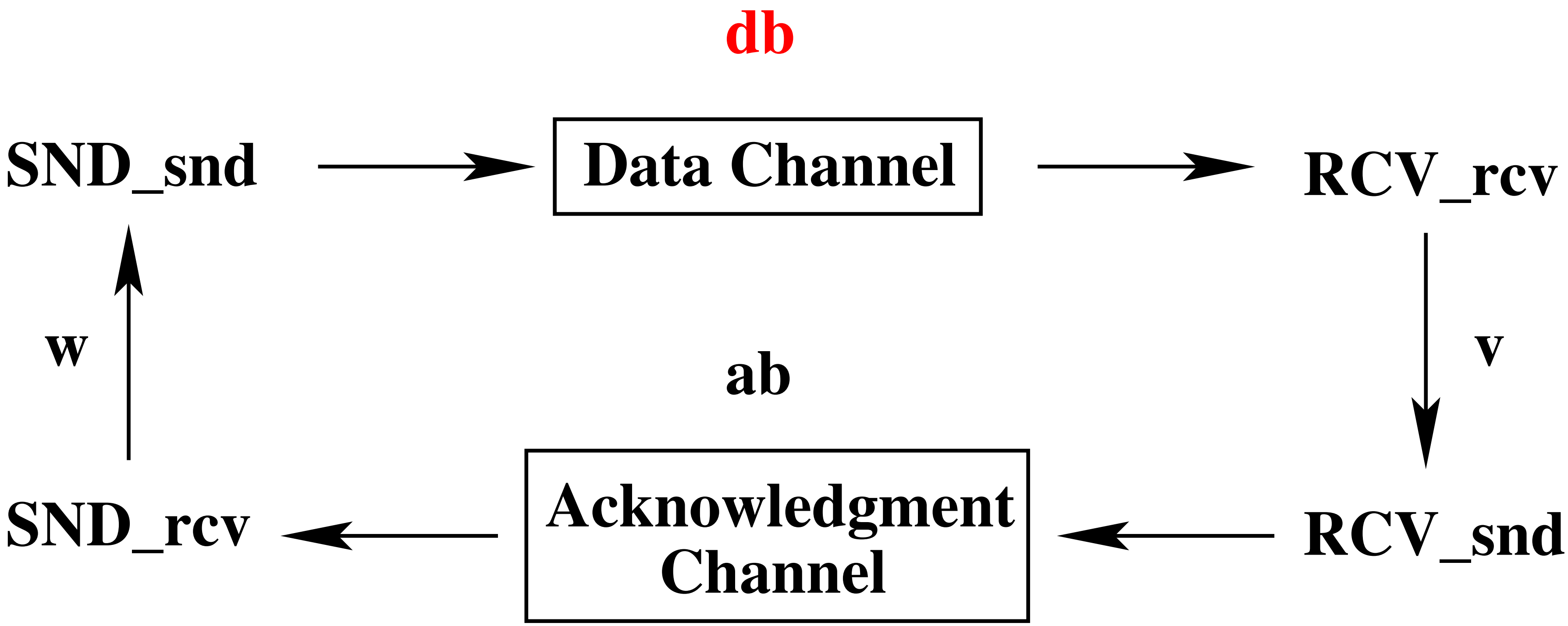
Activation bits at work



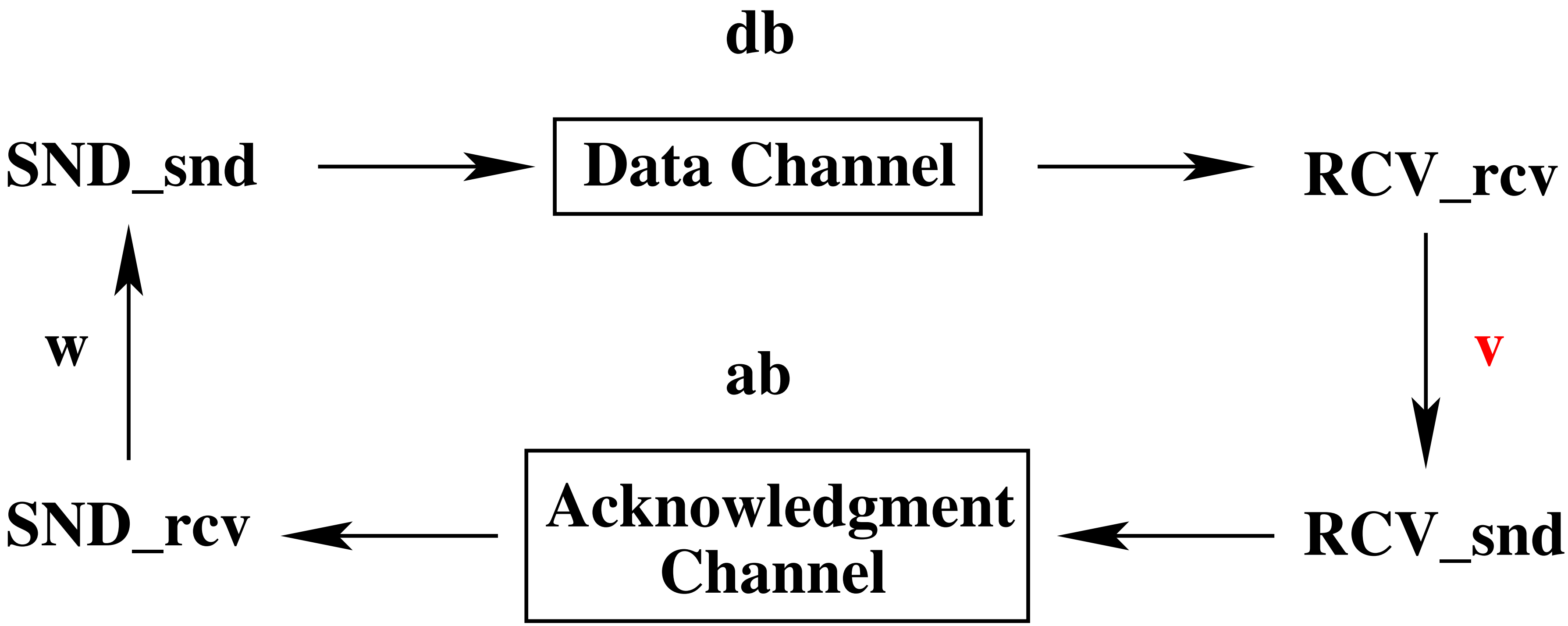
Activation bits at work



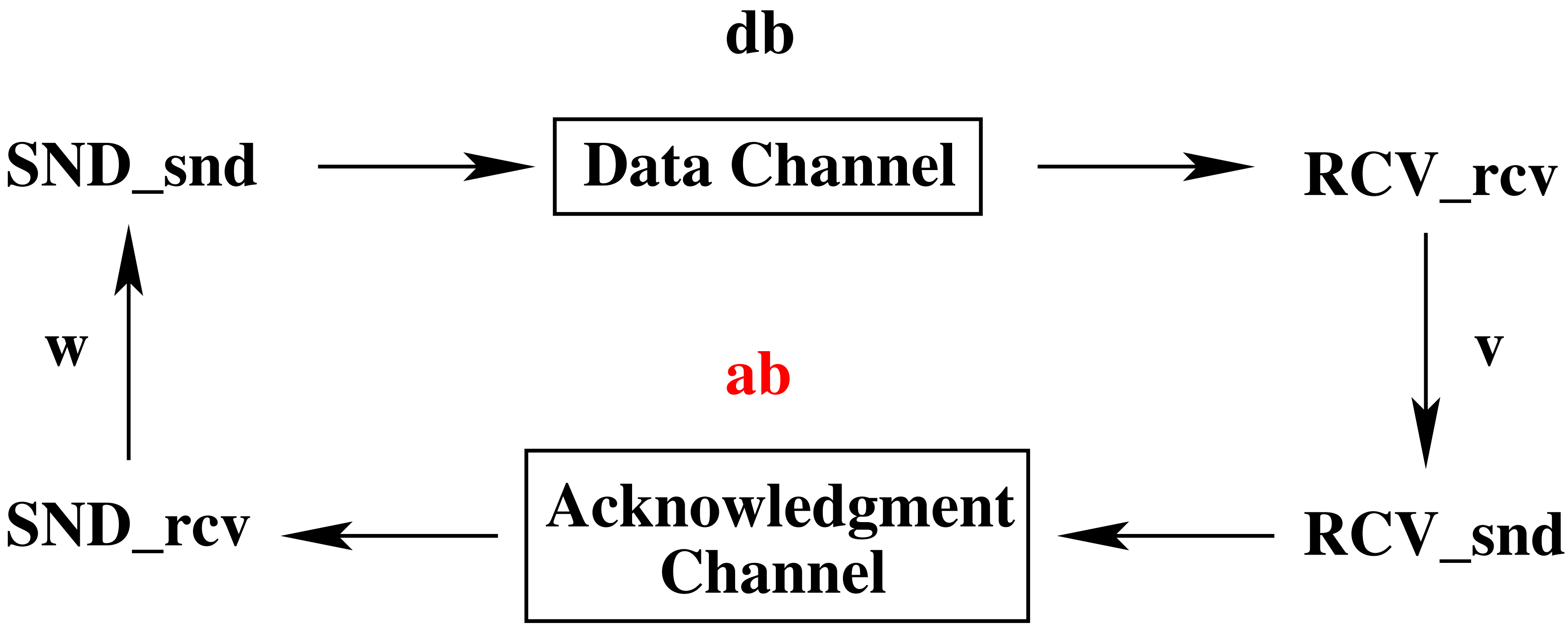
Activation bits at work



Activation bits at work



Activation bits at work



Fifth Refinement: Introducing the Last Item Indicator

- These invariants define the last data indicator

variables: $\ldots$
l

inv5_7: $db = \text{TRUE} \ \wedge \ r = s \ \wedge \ l = \text{FALSE} \ \Rightarrow \ r + 1 < n$

inv5_8: $db = \text{TRUE} \ \wedge \ r = s \ \wedge \ l = \text{TRUE} \ \Rightarrow \ r + 1 = n$

- 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 (**inv5_10**)

constants: $\ldots$
MAX

axm5_1: $MAX \in \mathbb{N}$

variables: $\ldots$
c

inv5_9: $c \in 0\,..\,MAX+1$

inv5_10: $c = MAX+1 \Leftrightarrow s\_st = failure$

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
refines
SND_snd_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
refines
SND_snd_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

- Daemons are breaking the channels

```
DMN_data_channel
  when
    db = TRUE
  then
    db = FALSE
  end
```

```
DMN_ack_channel
  when
    ab = TRUE
  then
    ab = FALSE
  end
```

- A failure is characterized by all activation bits being $\mathbf{FALSE}$

Fifth Refinement: the Events (3)

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

- Sender aborts after $MAX + 1$ tries

Fifth Refinement: the Events (4)

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

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

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

inv5_1': $db = \text{TRUE} \;\Rightarrow\; w = \text{FALSE}$

inv5_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

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

end

(concrete-)RCV_success

when

$r\_st = working$

$db = \mathrm{TRUE}$

$r = s$

$l = \mathrm{TRUE}$

then

$r\_st := success$

$r := r + 1$

$h := h \cup \{r + 1 \mapsto d\}$

$db := \mathrm{FALSE}$

$v := \mathrm{TRUE}$

end

inv5_1’: $db = \mathrm{TRUE} \;\Rightarrow\; w = \mathrm{FALSE}$

inv5_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

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

inv5_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

- In order to prove guard strengthening we need invariant **inv5_11**

inv5_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

inv5_12: $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv5_12** is needed to prove **inv5_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

inv5_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

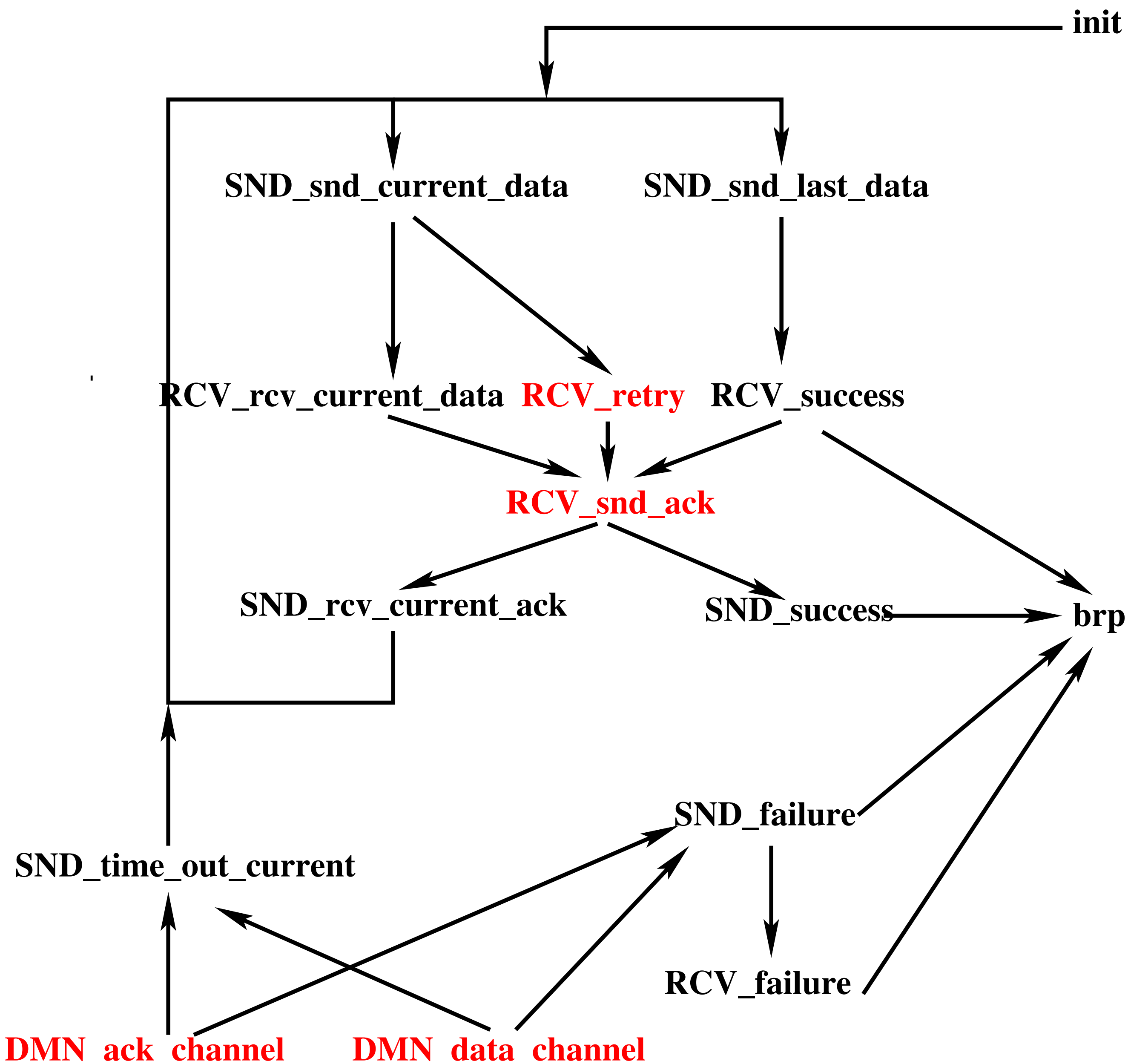
- In order to prove guard strengthening we need invariant **inv5_11**

inv5_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

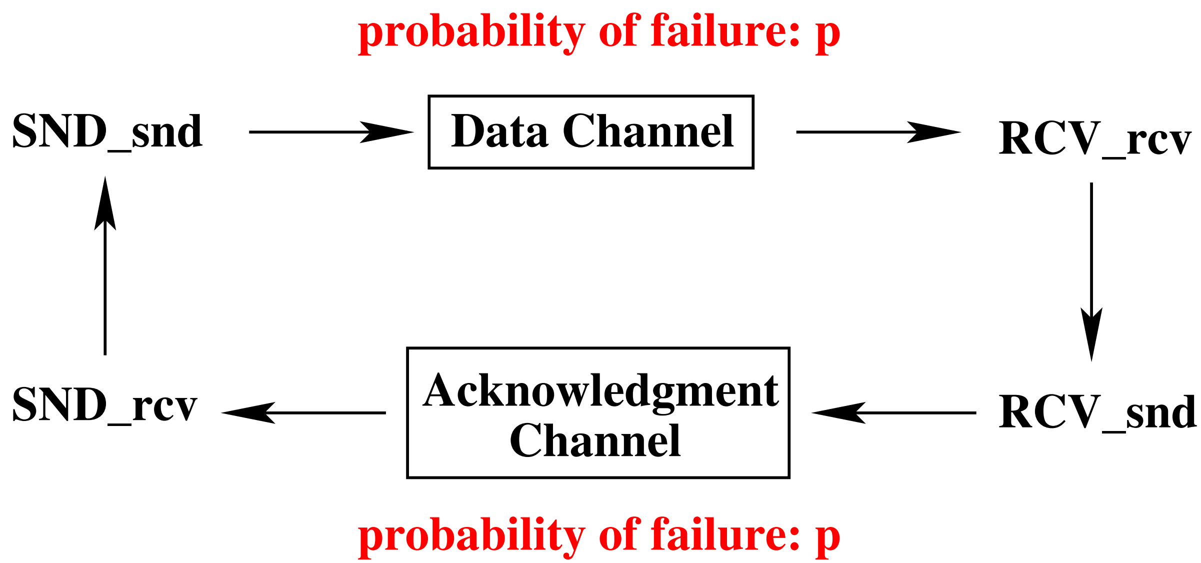
inv5_12: $v = \text{TRUE} \Rightarrow r = s + 1$

- Invariant **inv5_12** is needed to prove **inv5_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

Failure on one channel p

Success on one channel $1 - p$

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

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

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	$(1 - (1 - p)^2)^{MAX+1}$

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	$1 - (1 - (1 - p)^2)^{MAX+1}$

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	$(1-(1-(1-p)^2)^{MAX+1})^n$

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	$(1 - (1 - (1 - p)^2)^{MAX+1})^n$

$p = .1$
$MAX = 5$
$n = 100$ $\quad$ $.995$