

## Synchronous Atomic Broadcast Simulator Manual

### Underlying implementation:

`Processor.php`

Models individual processors and the network.

Contains global variable `$network`:

1. An array of instances of the `Processor` class
2. Represents the network

Defines the class `Processor`:

1. Private member variables
  - a. Boolean values
    - i. `$crashed`: is TRUE if the processor has crashed and FALSE otherwise.
  - b. Arrays
    - i. `$C`: Highest channels that messages have been received on so far, indexed first by time sent and then by processor id of sender
    - ii. `$H`: Incoming message buffer, indexed first by send time, and then by processor id of sender
    - iii. `$incoming`: Represents communication channels, and contains incoming messages. Indexed by channel number. Several messages can be headed towards a processor simultaneously on the same channel.
    - iv. `$schedule`: Pending scheduled events/tasks indexed by execution time
  - c. Integer values
    - i. `$bigDelta`: Worst case delay for all correct processors to receive message
    - ii. `$delta`: Point-to-point transmission delay parameter
    - iii. `$epsilon`: Maximum clock deviation between processors
    - iv. `$f`: Maximum number of faults to tolerate
    - v. `$forwardCrashChannel`: Channel after which to crash after the next forwarding attempt. If a new user specified crash point is designated when there is still a pending crash, then this variable takes on the smaller of the two values.
    - vi. `$myid`: Unique identifier for the processor
    - vii. `$pendingInFaults`: The number of remaining in-adapter faults that the user has told the processor to simulate so far. This value decreases to zero as faults occur, and can increase as a result of planned faults specified by the user of the simulation.
    - viii. `$T`: local time
  - d. String values

- i. `$summary`: Record of important events that happened to the processor during the simulation.
- 2. Constructor
  - a. `Processor($_myid,$_f,$_epsilon,$_delta,$_T)`: Initializes `$myid`, `$f`, `$epsilon`, `$delta` and `$T` with their underscore preceded inputs. Also initializes `$bigDelta`, and all other member variables.
- 3. Methods
  - a. Public methods
    - i. `breakInAdapter()`: Increases the number of pending in-adapter faults by one. As long as there are pending in-adapter faults, messages the processor attempts to read will be omitted.
    - ii. `channelOmission($c)`: Omits all messages incoming to the processor on channel `$c`.
    - iii. `crash()`: Crashes the processor, making it unable to respond to future commands.
    - iv. `crashOnForward($c)`: Indicates that the next time the processor is forwarding a message, that it should crash immediately after forwarding on channel `$c`, unless another, smaller value has been previously specified and is stored in `$forwardCrashChannel`, whose value is used instead.
    - v. `listen()`: Is called every step of the simulation so that the processor can see if incoming messages have arrived. Incoming messages have their predetermined travel times decremented until they equal zero, at which point the message is scheduled to be received that same time step, unless an in-adapter fault omits the message.
    - vi. `process()`: Called twice per simulation time step to execute scheduled tasks. The first call occurs before `listen()` is called, and assures that scheduled sends occur. The second call occurs after `listen` is called, thus allowing messages to be sent and received at the same global time.
    - vii. `scheduleAt($command,$time)`: Schedules the PHP command `$command` to be evaluated at the processor's local time `$time`. Corresponds to the `schedule` command used by Cristian.
    - viii. `summary()`: Returns value of `$summary`, a summary of important events that have been logged by the processor
    - ix. `tick()`: Called every time step to increment the local clock time.
  - b. Private methods
    - i. `appendSummary($message)`: Adds information about event in `$message` to the `$summary` variable.
    - ii. `bigReceive($sigma)`: Outputs that the processor has officially received the message `$sigma`. Corresponds to the `RECEIVE` command used by Cristian in the Deliver task.
    - iii. `bigSend($sigma,$crash = NULL)`: If `$crash` is `NULL`, as is the default, then this method broadcasts the message `$sigma` to each channel in increasing order using the `send` method, and then schedules the message to be delivered to the calling processor at

the appropriate time. If \$crash is set to have an integer value, then that value is the channel after which the current processor will crash in the send process. Corresponds to the SEND command used by Cristian in the Start task.

- iv. circlePlusH(\$T,\$s,\$sigma): Updates the incoming message buffer \$H to store the message \$sigma, indexed by its send time of \$T and its sender id of \$s. Corresponds to the operation  $H \leftarrow H \oplus (T, myid, \sigma)$  used by Cristian.
- v. deliver(\$T): Checks for messages in the buffer \$H that were sent at time \$T, and officially receives them (with the bigReceive function) in increasing order of their sender ids. Corresponds to the Deliver task used by Cristian.
- vi. forward(\$T,\$s,\$h): First \$C is double checked to be sure that a forward is necessary for the message received from sender \$s sent at time \$T with hop count \$h, and if the forward is still necessary, then this message is immediately forwarded on the channels specified by Cristian's algorithm in increasing order. The only exception to this is if \$forwardCrashChannel has a non-NULL value, in which case the processor crashes after forwarding on the channel specified by \$forwardCrashChannel. Corresponds to the Forward task used by Cristian.
- vii. output(\$message,\$summarize = FALSE): Echoes out the \$message to the HTML full log output. By default, \$summarize has a value of FALSE, but if it is set to TRUE, then whatever message is printed out will also be appended to the \$summary variable.
- viii. randomTransmissionTime(): Using the value of \$delta, returns a randomly generated travel time for a message being sent by the processor.
- ix. receive(\$T,\$s,\$sigma,\$h,\$c): Receives a message (\$T,\$s,\$sigma, \$h) on channel \$c, where \$T is the send time, \$s is the id of the original sending processor, \$sigma is the message and \$h is the hop count. Depending on the values of \$T, \$h and \$c, several things can happen. The message may be rejected as late, be recognized as having been seen before, or be noticed for the first time. If this is the first receipt of the message, then it is scheduled for delivery and potentially scheduled for forwarding. If it has been seen before, then the maximum channel on which it has been received is updated in \$C. Corresponds to the Receive task used by Cristian.
- x. send(\$T,\$s,\$sigma,\$h,\$c): Using the global variable \$network, sets the incoming channel \$c of each processor in \$network (except for the sending processor) to have the message (\$T,\$s,\$sigma,\$h), where \$T is the send time, \$s is the id of the original sending processor, \$sigma is the message and \$h is the hop count. Calls the setIncoming method of each of the processors to which the

message is being broadcast. Corresponds to the send command done as part of a SEND within Cristian's Start task.

- xi. `setIncoming($T,$s,$sigma,$h,$c)`: Sets a processor's own `$incoming` array to have an incoming message (`$T,$s,$sigma,$h`) on channel `$c`, where `$T` is the send time, `$s` is the original sender's id, `$sigma` is the message, and `$h` is the hop count. This method also determines how long it will take the message to reach the processor by assigning it a travel time with the `randomTransmissionTime` method.

### User Interface:

`sab.php`

Provides a user interface for defining simulation parameters and fault scenarios. What follows is a description of the interface rather than the underlying implementation details.

Page 1:

Number of processors:	<input type="text"/>	(A positive integer)
Maximum faults:	<input type="text"/>	(A non-negative integer)
Max clock deviation:	<input type="text"/>	(A non-negative integer)
Point to point delay:	<input type="text"/>	(A positive integer)
Simulation time:	<input type="text"/>	(A positive integer)
Max messages:	<input type="text"/>	(A positive integer)
<input type="button" value="Continue"/>		

A number of the type described in the right hand column should be entered into each input field before pressing continue. The meaning of each of these inputs is as follows:

1. Number of processors: Quite simply the number,  $n$ , of processors in the network.
2. Maximum faults: Number of faults,  $f$ , you want to design the simulated network to tolerate. This also indirectly sets the number of communication channels in the network to  $f + 1$ .
3. Maximum clock deviation: The maximum different in time units allowed between local clocks of processors in the simulation, usually represented by  $\mathcal{E}$ . Time is discrete in the simulation, which is why this must be an integer.
4. Point to point delay: The maximum number of time units that a message can take to travel between processors, usually represented by  $\mathcal{D}$ . This must be a positive integer because all transmission times will be strictly less than this value. Therefore, setting this parameter to 1 will insure that all messages take 0 time to travel between processors.
5. Simulation time: The number of discrete time units to run the simulation for. A large enough value should be picked to allow all events of interest to finish execution within the specified time frame.
6. Max messages: During the simulation, processors will send synchronous atomic broadcasts to each other. The details of each broadcast are filled out on the next page, so the number of messages you intend to define should be entered here. You

can give a value larger than the actual number of messages that you end up entering, but at least one message must be specified so that the simulation has something to do.

Page 2:

Simulation will involve 6 processors,  
tolerate up to 4 faults,  
allow for a maximum clock deviation of 5,  
involve a point to point delay of 4,  
last for 30 time units,  
and involve up to 4 messages

Initial Clock Values:  
(Without loss of generality, values are between 0 and 5)

Processor 1:   
Processor 2:   
Processor 3:   
Processor 4:   
Processor 5:   
Processor 6:

---

Message 1: Sender  Local send time  Message  Crash During Send   
Message 2: Sender  Local send time  Message  Crash During Send   
Message 3: Sender  Local send time  Message  Crash During Send   
Message 4: Sender  Local send time  Message  Crash During Send

---

Other Faults

---

Faults Used  
0  
Faults Available  
4

---

This is the general appearance of the next page, each portion of which is described in detail below:

Summary of previously entered values:

Simulation will involve 6 processors,  
tolerate up to 4 faults,  
allow for a maximum clock deviation of 5,  
involve a point to point delay of 4,  
last for 30 time units,  
and involve up to 4 messages

This is a summary of the values chosen on the previous page.

Initial clock times:

Initial Clock Values: (Without loss of generality, values are between 0 and 5)	
Processor 1:	<input type="text" value="0"/>
Processor 2:	<input type="text" value="0"/>
Processor 3:	<input type="text" value="0"/>
Processor 4:	<input type="text" value="0"/>
Processor 5:	<input type="text" value="0"/>
Processor 6:	<input type="text" value="0"/>

The simulation starts at global time 0, and in order to simplify matters, every processor is assumed to start out synchronized not only within  $\epsilon$  units of each other, but also within  $\epsilon$  units of 0. For each processor specified on the previous page there will be an input field whose default value is 0. Text above the fields defines the range of integer clock values allowed for these values (based on previously entered value for  $\epsilon$ ).

Messages to send:

Message 1: Sender	<input type="text"/>	Local send time	<input type="text"/>	Message	<input type="text"/>	Crash During Send	<input type="checkbox"/>
Message 2: Sender	<input type="text"/>	Local send time	<input type="text"/>	Message	<input type="text"/>	Crash During Send	<input type="checkbox"/>
Message 3: Sender	<input type="text"/>	Local send time	<input type="text"/>	Message	<input type="text"/>	Crash During Send	<input type="checkbox"/>
Message 4: Sender	<input type="text"/>	Local send time	<input type="text"/>	Message	<input type="text"/>	Crash During Send	<input type="checkbox"/>

There will be as many message entry rows as specified on the previous page. For each row, you have the option of specifying a message. A message requires the following information:

1. Sender: Integer id of sending processor.
2. Local send time: The local time on the sending processor at which it will attempt to send the message.
3. Message: A string value specifying the actual content of the message. Since this is only a simulation, these values can be arbitrary, but clear labels make the log output on the next page easier to follow.
4. Crash During Send: If there are still faults remaining, then this checkbox will appear. Check it if you wish for the processor to suffer a crash failure during its send. Every fault added will decrease the number of available faults, and effect the visibility of parts of the interface that allow faults to be added. Clicking the box will also cause the following input field to appear:

Crash During Send <input checked="" type="checkbox"/> After Channel <input type="text"/>
--

- a. After Channel: A drop down box of all available channels allows you to specify at what point during the send the processor will crash. The processor will successfully send the message on all channels up to and including the one specified, and then crash immediately afterward.

Other faults:

Other Faults

None

None

None

If faults are still available, then drop-down boxes will appear for each potential fault. These boxes allow for the specification of processor crashes and communication faults. Each option will cause its own interface to become visible:

1. Processor Crash:

a. At Global Time:

Processor Crash Crash processor 1 at Global Time of

This option causes the processor specified in the drop-down box to crash at the global simulation time specified in the input field after “of”.

b. At Forward Attempt

Processor Crash Crash processor 1 at Forward Attempt after global time after channel 1

This option causes the processor specified in the drop-down box to crash in the middle of the first forward attempt it makes after the global time specified in the input field after “after global time”. If, after the specified global time, the processor actually does attempt to forward a message, it will crash immediately after successfully sending a message on the channel specified in the “after channel” drop-down box.

2. Communications Fault:

a. In-Adapter:

Communications Fault Component In-Adapter Omit first message to processor 1 after global time

This causes the specified processor to suffer an in-adapter failure on the first message it receives after the global time specified in the input field. More accurately, at the global time specified, the number of pending in-adapter faults for the processor will be incremented. If there are already pending in-adapter faults, then this simply adds another pending fault. These faults only ever occur if the processor actually has a chance to receive a message.

b. Channel:

Communications Fault Component Channel Omit any message on channel 1 at global time

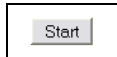
This fault kills all messages heading to all processors on the channel specified in the drop-down box at the specified global time. Thus, it has the potential to kill several messages at once.

Fault use:

Faults Used
3
Faults Available
1

These values track the used and available faults as information in the above sections is filled out.

Start button:



Once all above information is filled out, click this button to start the simulation.

Page 3:

This page runs the simulation and displays the results. At the top, the simulation parameters specified on the first page are displayed. Then there is a message specifying how many processors are being made, and then a line of text for each message that is being scheduled to be sent on a given processor.

Then there is a table with two columns. The left side is a full log of the simulation, and the right side has shortened, processor-centric summaries.

1. Full log: Each section of the full log starts by specifying the global time at which the logged events occurred. Then there is an ordered list of the events that occurred on each processor. Here is an example:

```
Global Time is 0
-----
P6 (T = 0): bigSend('message 1',1)
P6 (T = 0): Send "message 1" at time 0
P6 (T = 0): Sending out (0,6,"message 1",1) on channel 1
P1 (T = 3): The message (0,6,"message 1",1) is on its way to me on channel 1 (travel time is 3)
P2 (T = 1): The message (0,6,"message 1",1) is on its way to me on channel 1 (travel time is 1)
P3 (T = 2): The message (0,6,"message 1",1) is on its way to me on channel 1 (travel time is 3)
P4 (T = 4): The message (0,6,"message 1",1) is on its way to me on channel 1 (travel time is 3)
P5 (T = 0): The message (0,6,"message 1",1) is on its way to me on channel 1 (travel time is 1)
P6 (T = 0): crashed!
P4 (T = 4): Will crash on next forward attempt after sending on channel 2
P1 (T = 3): listens on channel 1
P2 (T = 1): listens on channel 1
P3 (T = 2): listens on channel 1
P4 (T = 4): listens on channel 1
P5 (T = 0): listens on channel 1
```

Each line of text first specifies the processor by “P#” where # is its unique id number. This is followed by “(T = t)” where t is the local time on the specified processor at which the event occurred. Then there is a message indicating what

happened. In the example above, we see that processor 6 broadcasted the message “message 1” at its own local time of 0, which happened to match the global time. It successfully sent the message out on channel 1 to processors 1 through 5, but then crashed. Processor 4 also receives a directive from the simulation indicating that it should simulate a crash failure the next time it forwards a message on channel 2. Finally, since there is a message on the way to processors 1 through 5, they explicitly indicate in the log that they are waiting for the message to arrive. Each of these log messages corresponds to a decrease in the remaining travel time of the incoming message.

2. Summary: The summary is organized by processor, and lists only the most important events that occurred to the given processor in the course of the simulation. Here is an example showing the summaries for three processors.

---

P1 Summary:

T = 7: (0,6,"message 1",1) received first on channel 1  
T = 7: (5,3,"message 2",1) received first on channel 1  
T = 10: Forward (0,6,"message 1",2) on channel 2  
T = 10: Forward (0,6,"message 1",2) on channel 3  
T = 10: Forward (0,6,"message 1",2) on channel 4  
T = 30: message delivered: "message 1"  
T = 35: message delivered: "message 2"

---

P2 Summary:

T = 5: (0,6,"message 1",1) received first on channel 1  
T = 6: (5,3,"message 2",1) received first on channel 1  
T = 8: Fault on channel 3 caused omission of incoming messages  
T = 10: Forward (0,6,"message 1",2) on channel 2  
T = 10: crashed!  
T = 10: Crashed after forwarding on channel 2

---

P3 Summary:

T = 3: (0,6,"message 1",1) received first on channel 1  
T = 5: Send "message 2"  
T = 30: message delivered: "message 1"  
T = 35: message delivered: "message 2"

---

Each section starts with “P# Summary:”, where # is the unique id number of the processor. Each line afterwards starts with “T = t:”, where t is the local time at which the processor experiences the event. Notice that the correctness of the synchronous broadcast can be seen in that correct processors (P1 and P3 in this example) will have all delivered the same messages at the same local times (though these may have been different global times). Both P1 and P3 delivered “message 1” at local time 30, and “message 2” at local time 35. P2 did not deliver

either of these messages because it crashed after a forward at local time 10. It crashed between receiving the messages and delivering them.

---

Having described the underlying functionality of the Processor class used to model processors in this simulation, and having described the user interface for the simulation, this manual is at an end.