Information Societies Technology (IST) Programme

Future and Emerging Technologies - Open Domain

Project acronym: Advance

Project full title:

Advanced Validation Techniques
for Telecommunication Protocols

Contract no: IST-1999-29082

Deliverable 12.1

Modeling and Verifying the PGM protocol

Year 2 : October 2001 - September 2002

Date of preparation: September 2002

Contributors: Paris, France-Telecom, Liège, Grenoble, Edinburgh

Contents

1 Introduction

The Advance project has begun the 1st of October 2000 for a duration of three years. Its goal is to develop a new generation of verification tools for telecommunication protocols. Those new tools should be able to tackle with problems like the resolution of timing constraints, unbounded numbers of data packets, unrestricted architectures and others. Since those problems comprise infinite components, previous verification tools based on the enumeration of all possibilities are unable to resolve those problems.

The project has a strong commitment to produce tools which could have a real impact on actual development methods in the telecommunication industry. It comprises mainly academic teams (from University of Edinburgh, University of Liège, University of Paris VII, University of Uppsala, VERIMAG and Weizmann Institute) but also teams from industrial R & D centers (Ericsson and France Telecom R & D). A significant part of the project is to validate the new tools by case studies upon problems big enough for ensuring that the tools would be useful later on real-size projects. This report gives a description of the work done in the second year of the project (October 2001 - September 2002) on one of those case studies.

The problem was submitted by France Telecom R & D. It consists in validating the PGM protocol, a multicast communication protocol designed by the IETF [1]. France Telecom, as an operator of telecommunication networks, has a natural interest in studying the behavior and performances of protocols which could be deployed in the future, and are proposed by network equipment builders, like Cisco. This study has been done by France Telecom R & D by designing a SDL[2] model of the protocol. This model enabled France Telecom to simulate the behavior of the protocol, but it didn't make it possible to prove properties of PGM, because this specification contains infinite parts, which can not be treated by the tools around the SDL language.

The sources of infinite behavior in PGM, and more generally in multicast telecommunication protocols come from several ways, but three of them seem to us particularly important to tackle because of their ubiquity in telecommunication protocols.

- Telecommunication protocols describe how to send data packets from a source to one or several receivers. The first source of infinity (or, more precisely, of intractable behaviors) comes from the number of those data packets. In PGM, the protocol numbers each data packets, and PGM can send 2^{32} different packets before coming back to the initial numbering. This implies that an exhaustive study of the protocol is out of the bounds of classical tools which enumerate all possibilities. It is necessary to use tools which treat the maximum number of packets as a parameter, and can then derive properties of the protocol whatever value this parameter is.

- A second source for infinite behavior is that communications protocols are designed to work in different configurations. PGM is a multicast protocol, that is, its configuration is tree-like: from one source, data are transmitted through several levels of nodes to receivers, which can be seen as tree leaves. Again, new methods have to be defined for deriving properties of the protocol which are true whatever the topology is.

- Last, telecommunication protocols make often use of timers. For example, PGM use timers for managing its policy of asking reemissions of lost data packets. And one of

[1]Internet Engineering Task Force, *i.e.* the body responsible for standardizing the communication protocols of the Internet.

[2]Specification and Description Language, a standardized language defined by the International Telecommunications Union for describing telecommunication systems and protocols.

the goals of PGM is to ensure that all losses can be recovered. This property depends on the relations between the timers used in the protocol and the delays introduced in the transfer of data packets. Since the timers can take any value, an interesting goal is to derive constraints between those values for ensuring that desired properties are fulfilled.

The work pursued in the Advance project has followed the lines described here. A first work was to model the PGM protocol in the language IF used and defined in the project. An SDL description given by France Telecom R & D served as a starting point, together with the IETF informal standards. This work was mainly done by the teams of Paris and Grenoble, with support from France Telecom R & D, and influenced the definition of the new version of the IF language. This modeling work is described in Section 2. It brings to some improvements of the common description language IF 2.0, which are presented in Section 3. Section 4 describes the results obtained by analyzing the untimed versions of the PGM. Section 5 describes work done on deriving and checking reliability constraint between the parameters of the protocol. The work reported in Sections 4 and 5 has been carried out by Marc Boyer from Paris, and use the verification tools of the IF-toolbox (by Grenoble) for untimed and timed systems. The work described in Section 6 has been done by Liège and uses the LASH tool which enables to cope with the parameterization of the number of data packets. Section 7 was written by Edinburgh. It shows that proving the correctness of the PGM for all possible topologies (with an arbitrary number of components) can be done by considering only a finite set of simple instances of these topologies. Yves-Marie Quemener from France Telecom R & D served as an editor for this report.

Acknowledgement The researchers from the ADVANCE project who participated to the work on the PGM case study are: Ahmed Bouajjani, Marc Boyer, and Mihaela Sighireanu from Paris, Yves-Marie Quemener from France-Telecom, Bernard Boigelot and Louis Latour from Liège, Marius Bozga and Yassine Lakhnech from Grenoble, Javier Esparza and Monika Maidl from Edinburgh.

2 The PGM protocol

The Pragmatic General Multicast (PGM) protocol is "a reliable multicast transport protocol for applications that require ordered or unordered, duplicate-free, multicast data delivery from multiple sources to multiple receivers" [10, Abstract, p. 1]. It is an "Experimental Protocol" [10, Status of this Memo, p. 1] whose "central design goal is simplicity of operation with due regard for scalability and network efficiency" designed for "multicast applications with basic reliability requirement" [10, Abstract, p. 1].

One specific aspect of this protocol is that it is "NAK-oriented", *i.e.* the reliability mechanism is not assured by acknowledging the received messages, but by sending a negative acknowledgment (`NAK`) when a message seems to be lost. The idea between this mechanism is to reduce the number of control messages respectively to data messages.

The protocol is fully described in [10].

2.1 Overview of the protocol

A PGM session has one single sender, a set of receivers and a set of network elements. From a topological point of view, it can be seen as a tree, with the sender as the root, the network elements as nodes and the receivers as leaves.

The source "multicasts sequenced data paquets (`ODATA`)" downstream in the tree. When a receiver detect that an `ODATA` has been lost (receiving out of sequence `ODATA` for example), it sends upstream a `NAK` with the sequence number of the `ODATA` paquet send. When the source gets a `NAK`, if the matching data paquet is still in memory, the same data is sent in response, but in a repair data paquet (`RDATA`).

This basic behavior is very dependent upon the reliability of the `NAK`. Hence, a hop-by-hop reliability for `NAK` is ensured: each element of the tree (leaf or node) will repeat the emission of a given `NAK`, until the reception of a matching `NAK` confirmation (`NCF`) sent by the upper element in the tree (node or root).

Moreover, some source-path-message (`SPM`) are sometimes sent. Their main goal is to maintain some routing information for the protocol (the topology of the PGM tree). Moreover, they carry some informations like the last message sent by the source, which helps the receivers to detect losses.

2.2 The protocol in the ADVANCE project

This PGM protocol is a case study suggested by France Telecom in the ADVANCE project. The PGM protocol had been studied by one of their team, looking at a draft version of the protocol, and modeled in SDL [3].

Hence, there existed three main sources of specification for this protocol: the drafts (the final RFC has been published a few months after the start of the project), a France Telecom specification document [3] and the SDL source code.

2.3 More details on the PGM protocol

The full description of the protocol in [10] is 111 pages long, and even only the main mechanisms need 30 pages to be described.

Only some details useful to understand the studies in this document will be presented in this section.

2.3.1 The transmit window

The transmit window is the set of data packets that have been sent by the source and that are still in memory. The trailing edge (called `TXW_TRAIL`) is the oldest data packet available for repair and the leading edge (called `TXW_LEAD`) is "the sequence number of the most recent data packet the source has transmitted" [10, § 3.3].

2.3.2 Detection of losses

The PGM protocol is NAK-oriented: a receiver does not acknowledge the received packets, but sends a `NAK` to signal the losses. The question is: how can it detect losses? It should have a (partial) knowledge on the transmit window: having a lower bound of `TXW_LEAD` to `NAK` packets not received, and an approximation of `TXW_TRAIL`, to avoid to send `NAK` for messages that are no more in the source buffers.

To achieve it, each `ODATA` and `RDATA` packet will be sent with the current value of `TXW_TRAIL` (and its sequence number is a lower bound of the current `TXW_LEAD`), and each `SPM` will be sent with the current value of `TXW_TRAIL` and `TXW_LEAD`.

The PGM works with the assumption that the underlying network deliver the packets mostly in sequence. Then, if a hole in the sequence of received packets is detected, it is considered as a loss.

Hence, by comparing the sequence number of the last message received with the sequence number of the `ODATA` (or `RDATA`) or with the `TXW_LEAD` of a `SPM`, the receiver can detect losses.

It will also detect unrecoverable losses by looking at the `TXW_TRAIL` [3].

2.3.3 `NAK` and `NCF`

The reliability of the `NAK` mechanism is ensured hop-by-hop: when a receiver sends a `NAK`, its reception is acknowledged by the node by sending a `NCF`, and then the `NAK` is forwarded upstream.

When the source receives a `NAK`, it replies with a `NCF`, and then, if the `NAK` sequence number is in the transmit window, the source repairs the loss with the adequate `RDATA`.

2.3.4 Avoiding `NAK` explosion: filtering and random backoff

One problem with multicast protocol is the explosion of signaling messages at the source: if every receiver acknowledges each received message or `NAK` each loss, with thousands of receivers, the receiver can get a storm of signaling messages.

To avoid `NAK` explosion, the nodes have some mechanisms to "filter" the `NAK`: if a `NAK` is received from downstream or a `NCF` from upstream, all `NAK`s with the same sequence number received a few time after (see [10, § 7.4]) will be dropped.

Moreover, each receiver waits a "random backoff" before sending a `NAK`, and cancels this sending if a matching `NCF` is received while waiting. That is, if the same packet is detected as lost by all the receivers under a given node, if the random backoffs are different enough, only one receiver will send a `NAK`, which will be acknowledged to all the receivers (using multicast), and all (except the one that have sent the first `NAK`) will cancel the `NAK` emission. This is a rough description: the full mechanism is in [10, § 6.3].

After this short overview of PGM, we present what are the properties interesting to verify, and the complexities introduced by the protocol.

2.4 Properties of PGM to be verified

The protocol is unreliable, hence, we can not verify in the general case that all messages will be delivered. In fact, in the specification of the protocol, the only property that is expected from the protocol is "PGM guarantees that a receiver in the group either receives all data packets from transmissions and repairs, or is able to detect unrecoverable data packet loss." [10, Abstract, p. 1]. This property is called "*Losses-signaled*" in this paper.

This property implies that a receiver that will receive all messages from the network and drop all of them, just signaling to the application level that all messages have been lost, such a receiver will be compliant with the specification of the protocol. Of course, such a receiver is of no interest. This could mean that the property *Losses-signaled* itself has no interest. In fact, it only reflects that, because of the unreliability of the communication medium, it is difficult to impose strong properties upon PGM, but only a weak one like *Losses-signaled*.

However, there are some other properties that could be interesting to study, to see the efficiency of the mechanisms of the protocol.

[3] In fact, there is no requirement in the [10] to declare a loss as unrecoverable by just looking at this information, but it will be an assumption of our models, that seems reasonable.

Reliability efficiency. Even if there is no requirement in the protocol of a total reliability, a retransmission mechanism exists. It could be interesting to see the efficiency of this mechanism, *i.e.* which are the parameters of the systems that have an influence upon the reliability of the system, and for example under which assumption are all messages delivered.

In the PGM protocol, some parameters can be controlled by the protocol itself (like the throughput of the source, or the size of the window and the `SPM` rate), and other ones that are imposed by the environment (like the communication delays, the loss rate, etc.). Then, the question could be: in a given environment, what are the parameters that the protocol can change to increase the reliability?

Protocol load. Another important aspect in a protocol is the ratio payload/bandwidth, *i.e.* how many bytes are sent in the network and how many were really needed? To completely address this issue, a model should take into account the size of the headers and the size of the packets, and such details will be very hard for the tools.

Nevertheless, a simplest way is to count the messages without any useful data: how many signaling messages (`NAK`, `NCF`, `SPM`) and how many useless `ODATA` or `RDATA` (such a packet is useless if it is discarded by the receiver, *i.e.* if a `ODATA`/`RDATA` with the same sequence number has already been received, or if the receiver has already declared this message as definitely lost). Even in signaling messages, there are useless messages, like duplicated `NAK`s.

2.5 Sources of complexity in PGM

For deciding what are the best abstractions to do when modeling PGM, our first work was to list the dimensions of the complexity of the system.

Topology. If we want to consider the fully general case, the topology of the system is a general tree. Hence, it should be possible to derive properties whatever the topology of the system is, as soon as it is a tree. But the verification tools typically used need to consider one given topology. Then, more and more complex trees can be taken into account, like illustrated in Figure 1. We will also see in section 7 how to derive properties for infinitely many topologies.

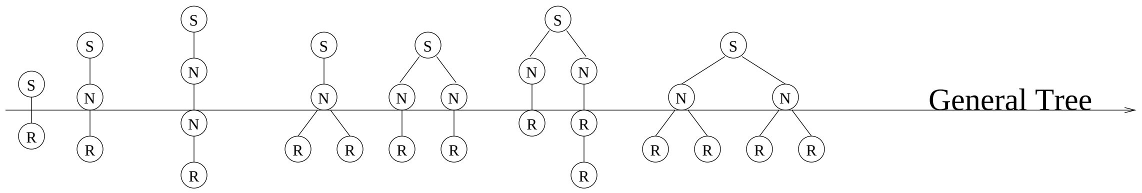


Figure 1: Increasing complexity of topology

Loss policy. If no assumption is done upon the loss policy, *i.e.* if the more general case is assumed (every message can be lost), it will be impossible to prove anything about the protocol.

Nevertheless, there are different more or less realistic assumptions that can be done. The simplest one is, obviously, to assume that there is no loss, and then to study what are the properties of PGM. But this assumption is also the less realistic one. The next is "one fixed loss" *i.e.* one fixed paquet is lost. A more complex is "one random loss" (it is more complex for verification tools, since they have to test, for each packet, what happens if this packet is lost or not). Then, more complex assumptions can be considered, like "at most N losses" or "at most N losses and never more than S in sequence" or "at most N over M packets are lost", with a special case when this M is the window size.

But this dimension of complexity is on the number of losses. Another dimension is: what kind of messages can be lost ? The most simple assumption is that only `ODATA` can be lost. Then, the signaling of the loss with `NAK` and `NCF` messages won't be affected by losses, and the protocol can repair the loss with `RDATA` messages. But in a real network, the signaling `NAK` messages and the `RDATA` ones could also be lost. To handle those potential losses, the protocol includes a loop of `NAK` emissions, and the system of `NAK` confirmations with `NCF` messages.

Communication mediums. What is the assumption for the communication between processes? In the worst case, a IP network is an unordered, unreliable channel, with unbounded transmission delay. Nevertheless, the IP network in general works, and assuming that it can be modeled as a FIFO queue is realistic.

For our tools, the most simple case is when processes are communicating through a one-bounded queue (only one shared variable). After that, delays can be added, or bounded queues or even more unbounded ones.

Transmission length. What is the size of the transmission (*i.e.* the number of packets sent during a session)? How many messages are the tools able to handle? For testing all the mechanisms of the protocol, the transmission length should be at least more than 2-windows long (to have some messages in the first window, some in the last, and some neither in one or the other). And a window should contain at least 3 messages, to have a first message, a last and also one that is neither first neither last.

But this is a simplification, because the PGM has not been designed to send only 9 messages but more realistically thousands of data packets.

An abstraction simplifying the work for the verification tools is to assume an infinite transmission (they do not have to take into account any ending).

The most difficult is a parametric one, *i.e.* being able to consider the length as a parameter, and making verification without any assumption on the value. This kind of work is described in section 6.

Traffic shape. Another dimension of complexity is the shape of the traffic: in [10, § 5.1.2], it is assumed that "a source MUST use a token bucket scheme or any other traffic management scheme that yields equivalent behavior" and, "in addition, a source MUST bound the maximum rate at which successive packets may be transmitted using a leaky bucket scheme drained at a maximum transmit rate, or equivalent mechanism."

Even with this restriction, it opens a wide range of traffics' shapes, considering the values of the token bucket size, the token bucket rate and the leaky bucket rate...

Moreover, the protocol is also sensitive to a local absence of data to transmit (because data packets carry signaling informations), and a special mechanism has been defined for this purpose [10, § 5.1.5 Heartbeat SPMs].

The most simple case is when the source emits at constant rate (the token bucket size is one) and never stop to transmit until the end of the communication.

Group membership (joining and leaving). "PGM has no notion of group membership" [10, § 1] and receivers can join or leave the session at any time, without any signaling.

Nevertheless, there is a special mechanism in the receiver to handle the arrival in a session. When a receiver joins, it had not received the messages sent before its arrival. Then, which messages can be considered as lost, and requested to be retransmitted ? A mechanism is presented in [10, § 6.1]. There is also an option of the protocol (`OPT_JOIN`, [10, § 9.4]) that defines a more "clever" algorithm to deal with joining.

Another difficulty in modeling the arrival of receivers is that this implies that the topology of the protocol is dynamic.

The model can be simplified, assuming that all receivers are in the group from the very beginning (and then, not implement the mechanism of initial data reception), or consider one late join at a given moment, or consider random join and leave.

Figure 2: Joining complexity

Removing mechanisms. Another solution to decrease the complexity of the system is to remove some of the mechanisms, that are designed to improve the mean performances of the protocol, but are not very relevant in a verification context, like the complex `NAK` generation mechanisms.

More roughly, mechanisms like the heartbeat `SPM` [10, § 5.1.5] or the filtering in nodes of `NAK`s [10, § 7.4] and `RDATA` [10, § 7.8] can also be removed.

Implementation choices. The specification of the protocol leaves the programmer with a lot of freedom about how to implement a lot of mechanisms, like the rate of ambient `SPM` [4], or the progress policy of the sender window[5].

A nondeterministic policy could be modeled, that simulates the best as well as the worst policy. This solution has two major drawbacks: first, it can be hard to verify since it increases the number of potential behaviors; second, the performances of the protocols will be very bad (as an example, the transmit window policy that drops a message from memory as soon as it had been sent will never allow to recover any loss).

2.6 Simplifications and assumptions done in the project

The presentation of the complexity dimensions was done during the sixth meeting of the ADVANCE project (Paris, December 2001).

During this meeting, after a discussion with the France Telecom team about the realism of different options, it had been decided not to take into account the joining/leaving of receiver, and to focus on the three following points: parametric verification regarding the number of sent packets; deriving properties which are true whatever the topology is; and deriving temporal constraints. But if those tasks needed advanced verification tools, it

[4]The ambient `SPM` rate must be "at least sufficient to maintain current source path state in PGM network elements." [10, § 5.1.4]

[5]Even if an appendix in the RCF ([10, Appendix F]) is devoted to this mechanism, it is given "for illustrative purpose only".

can be shown that classical (finite state) verification tools can already bring interesting results.

3 Influence of modeling PGM upon the IF language

As the entry point of the Advance verification toolset, the common language IF needs enough expressive power to allow for a sound and direct mapping from complex SDL specifications. Therefore, the PGM protocol, and in particular its SDL specification written by France Telecom, has had a strong influence on the design of the common language IF-2.0.

In addition to the fact that PGM is the major Advance case study, we found the PGM specification extremely interesting mainly for two reasons.

First, this specification was developed by people working outside the Advance project and moreover outside the formal verification area. Therefore, the emphasis was on *modeling* i.e, remaining closer as possible to the IETF description, and not on verification. That is, no verification tool or method had been a priori targeted and no simplifications were made.

Second, given its complexity, as well as the large number of SDL concepts used, the PGM specification represents a typical medium-sized case study in the telecommunication area. We recall here that the whole description is about 3400 lines of textual SDL/PR code, and uses features such as abstract data types, dynamic creation and destruction, asynchronous communication, procedures, timers, etc.

Therefore, the IF-2.0 language includes (but is not limited to) most of the widely used concepts of the SDL language, which are also present in the PGM specification (see above). Nevertheless, the IF-2.0 language offers, in addition to SDL, a much more powerful timed-model based on urgencies, parameterization at different levels (i.e, data, timing or configurations), structured control on transitions, hierarchical states, etc.

All these features combined have been made possible the complete rewriting of the whole SDL specification in IF-2.0. The IF-2.0 description is about 1300 lines of IF code, which is 2.5 times smaller than the SDL one. Moreover, some aspects of the specification get improved, for example, the multicast tree structure become parameterized. Furthermore, the rewriting process pointed out several inconsistencies in the original SDL specification such as wrong initializations, dead code, useless communications, etc.

That is, the existence of the PGM case study guarantees for the continuation of the project that the modeling language we decided to use is able to cope with realistic telecommunication applications.

4 Preliminary experiments

The first attempt for analysis of PGM has been to consider finite models and to analyses them using classical tools, *i.e.* simulation and model-checking tools which work by exhaustively enumerating all the possible behaviors of the studied system. Such tools are not able to fulfill the more complicated verification tasks we had in mind but can nevertheless bring interesting results.

4.1 Tools used

Since these experiments take place during the first years of the project, when the version 2.0 of IF language and the tools associated was not yet available, we used as modeling language IF 1.0. Roughly speaking, the IF 1.0 underlying theoretical model is finite state automata with clocks and finite domain variables, communicating trough shared variables and buffers.

To generate the labelled transition system corresponding to the IF 1.0 model we used the `generator` tool of the IF toolset. The output of this tool is a graph in the ".aut" format. This format can be handled by the CADP tools [5].

We use CADP toolset [5] in order to verify off-line temporal logic properties or to minimize and compare graphs.

4.2 Untimed model for PGM

The full PGM protocol is out of the scope of existing verification tools: a complete modelization implies to be able to handle thousands of processes, dynamic creation and destruction of processes, dynamic routing, dozens of counters and timers per process, and transmissions of length greater than 2^{32}...

Nevertheless, some interesting aspects of the protocol should be studied, doing the right assumptions and abstractions, simplifying some complexities we have already discussed.

A first model was produced by the VERIMAG team, as a translation of the SDL code given by FRANCE TELECOM, using the IF 2.0 language.

This code had some drawbacks for our use: it was written in IF 2.0 and made an intensive use of lists and external procedures (*i.e.* procedures written in `C` language), that can not be handled by the verification tools (and are very resource consuming for the IF `generator`).

Hence, there was a need to define other models for the project.

The two first works were not exactly new models, but common material for the members of the project (in order to share the same knowledge of the protocol): the IF 2.0 file result of the translation has been commented, and an "automaton oriented" description of the [10] has been written. All files have been uploaded on the WEB server [1, Case Studies].

The first code written was the "very simple pgm" with the following features:

Topology: 1 source – 1 network node – 1 receiver

Policy of loss: 1 random `ODATA` lost

Communication medium: 1-bounded buffers (without delay), one per type of message, reliable

Transmission length: parameter of the code

Traffic shape: none (untimed model, but limited by the size of the communication buffers[6])

Group membership: the receiver is part of the group since the first time

Removed mechanisms: no heartbeat `SPM`, no filtering in nodes, no repeat of `NAK` (useless since `NAK` communication is reliable), untimed model

Implementation choices: undeterministic transmit window policy, and ambient rate driven by the number of send messages (one `SPM` is send each time `AMB_RATE` `ODATA` have been sent, with `AMB_RATE` a parameter of the code)

The architecture (*i.e.* topology and communication buffers) of this "very simple pgm" is illustrated in Figure 3.

[6] The production rate is limited by the consuming rate and buffer size. Since the communication is reliable, the sender can not write in a buffer if the data has not been transmitted.

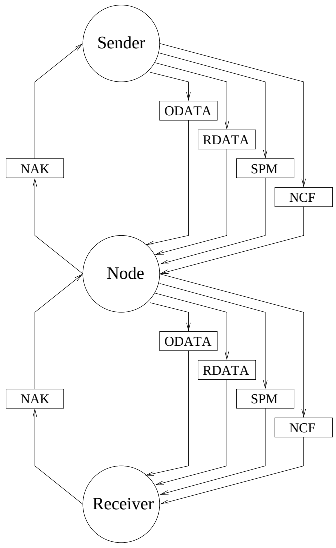


Figure 3: Topology and communication buffers of the "very simple pgm" model

This was a first code, the "starting point" for the project. The aim was to use it as a first input for the tools, and to see if more complex models could be taken into account by the tools, or if some more simplifications and abstractions had to be done.

4.3 First experiment: *Losses-signaled*

The first experiment was simply to verify the *Losses-signaled* property on the very simple model.

Since this property is very basic, it was more a test for our model and the performances of the verification tools that a real test of the property. Actually, an interesting behavior have been found, that was unforeseen, even if, once the computer has given a counter-example, the mistake appears obvious. But is it not the case for most of the bugs?

The context of the experiment was the very simple protocol, described in Section 4.2, with a few messages.

The results were that the property *Losses-signaled* was verified for all messages, except for those of the last window.

The experiments are presented in the Table 1, looking at the lines with configuration "SNR 1p" and "Closing SPM" value "false" in the rows of *Losses-signaled*.

By scanning the counter examples, the reason why the messages in the last window can be neither delivered neither signaled as lost, can be explained as follows: if the first message of the last window is lost, and if the transmit window policy makes the window advance by trying to keep as much messages in memory than is allowed, and closing it once the transmission is ended, all `ODATA` messages will be sent with a `TXW_TRAIL` value meaning that the lost message can be recovered. But the `NAK` sent by the receiver can be received too late, and discarded. Then, the receiver is not able to detect that the message has been lost and waits indefinitely (the mechanism of loss detection is presented in Section 2.3.2).

This behavior is illustrated in Figure 4, considering a very simple communication of 2 data packets, with 2 as window size.

This highlights the fact that there is no special mechanism in the protocol to identify the closing of a session.

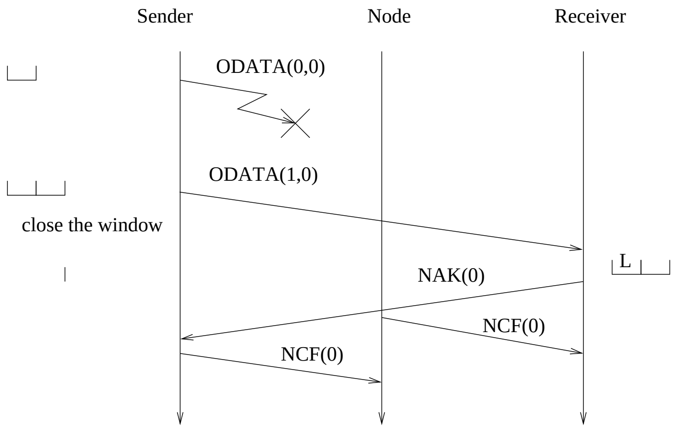


Figure 4: Example of last window frozen

Nevertheless, in the protocol, there is an "heartbeat SPM" mechanism, designed to send some `SPM` when no `ODATA` or `RDATA` are sent. In our model, a "closing SPM" option has been added: if this option is set, when all the `ODATA` packets have been sent, a `SPM` is sent (with the value of the transmit window). This option is illustrated in Figure 5, considering the same communication as in Figure 4.

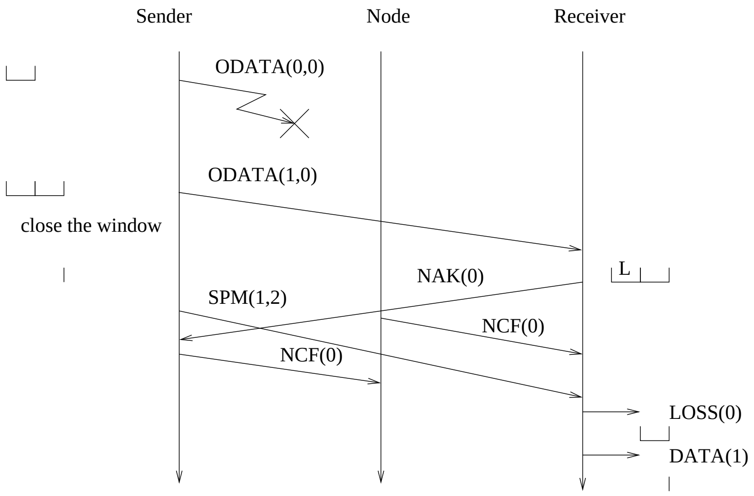


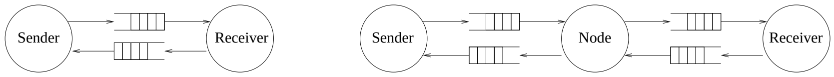
Figure 5: Example of effect of the "closing SPM"

Then, if this option is set, the property *Losses-signaled* is verified for all messages, as presented in the result Table 1 in lines with "Closing SPM" value "true" (for "SNR 1p" configurations).

4.4 Second experiment: removing nodes

The second experiment was designed to handle a problem about the architecture of the protocol.

What is the role of a node in the model? In the real protocol, it is used for routing and filtering, but, as our models do not consider routing problems, if we do not care about the filtering aspect, can we remove the nodes? This would simplify the verification work without loss of generality if we could show that removing nodes have no effect.



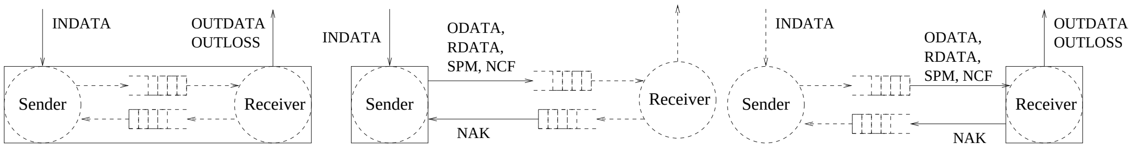
(a) The SR config (b) The SNR config

Figure 6: Architecture of the two configurations compared

Hence, two topologies of the protocol have been considered: one with one source, one node and one receiver (designed by SNR), and one with only one source and one receiver (designed by SR), communicating through unbounded FIFO queues[7], like illustrated on Figure 6.

If we are to make comparisons, the main question is what is the comparison criterion? Three point of views have been defined for the problem:

1. the *blackbox* point of view: looking the system from the application point of view, hiding all actions except for the inputs and outputs of the session (Figure 7(a));
2. the *source* point of view: looking only at the inputs and outputs of the sender process (Figure 7(b));
3. the *receiver* point of view: looking only at the inputs and outputs of the receiver process (Figure 7(c)).



(a) BlackBox (b) Source (c) Receiver

Figure 7: The different points of view

Then, the comparisons will be to test the "τ^*-bisimulation" relation ([7], the `-iequ` option of the `aldebaran` tool [5]) between the behaviors graphs, with only a given set of visible actions (different for each point of view).

Some configurations have been tested, as presented in the Table 1, looking at the lines "SNR q" and "SR q". The rows "Blackbox Equiv.", "Source Equiv." and "Receiver Equiv." give the results for this test of equivalence.

A first result, that confirms the previous study, is that the effect of the "Closing SPM" option is the same for these models than if the system was communicating through 1-place buffers.

But the main results are that, in all the tested cases, both systems are equivalent from the application and sender point of view, but not from the receiver point of view.

[7]We could not compare the two systems using the 1-bounded queues, like with the model used in section 4.3, since it is obvious that the configuration SNR with 1-bounded buffers could perhaps be equivalent to a SR configuration with 2-bounded buffers, but never to a SR with 1-bounded buffers. And, with the tools used, it was easier to consider unbounded FIFO than 2-bounded FIFO.

# of mess	Window Size	AMB rate	Closing SPM	Conf.	Trans.	States	*Losses-signaled*							BlackBox Equiv.	Source Equiv.	Receiver Equiv.
							0	1	2	3	4	5	6			
5	1	2	**ff**	SNR 1p	17,642	7,443	**tt**	**tt**	**tt**	**tt**	**ff**	-	-	-	-	-
				SNR q	36,944	20,802	**tt**	**tt**	**tt**	**tt**	**ff**	-	-	**tt**	**tt**	**ff**
				SR q	5,501	3,189	**tt**	**tt**	**tt**	**tt**	**ff**	-	-			
5	1	2	**tt**	SNR 1p	19,821	8,338	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	-	-	-
				SNR q	49,141	27,849	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	**tt**	**tt**	**ff**
				SR q	6,555	3,812	**tt**	**tt**	**tt**	**tt**	**tt**	-	-			
5	3	2	**ff**	SNR 1p	1,089,462	349,270	**tt**	**tt**	**ff**	**ff**	**ff**	-	-	-	-	-
				SNR q	2,002,554	1,018,918	**tt**	**tt**	**ff**	**ff**	**ff**	-	-	**tt**	**tt**	**ff**
				SR q	181,185	89,933	**tt**	**tt**	**ff**	**ff**	**ff**	-	-			
5	3	2	**tt**	SNR 1p	1,139,092	367,235	**tt**	**tt**	**ff**	**ff**	**ff**	-	-	-	-	-
				SNR q	2,594,217	1,345,093	**tt**	**tt**	**ff**	**ff**	**ff**	-	-	**tt**	**tt**	**ff**
				SR q	214,434	108,176	**tt**	**tt**	**ff**	**ff**	**ff**	-	-			
5	∞	∞	**ff**	SNR 1p	567,709	189,912	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	-	-	-
				SNR q	328,118	168,336	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	**tt**	**tt**	**ff**
				SR q	55,883	28,176	**ff**	**ff**	**ff**	**ff**	**ff**	-	-			
5	∞	∞	**tt**	SNR 1p	643,660	217,471	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	-	-	-
				SNR q	442,742	233,676	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	**tt**	**tt**	**ff**
				SR q	69,970	35,163	**ff**	**ff**	**ff**	**ff**	**ff**	-	-			
7	2	2	**ff**	SNR 1q	1,704,342	556,170	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	-	-	-
				SNR q	21,182,556	11,073,929	†	†	†	†	†	†	†	†	†	†
				SR q	1,016,579	527,475	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**			
7	2	2	**tt**	SNR 1q	1,735,130	567,283	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	-
				SNR q	26,333,252	13,944,848	†	†	†	†	†	†	†	†	†	†
				SR q	1,182,067	620,581	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**			
7	2	∞	**ff**	SNR 1p	152,298	58,718	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	-	-	-
				SNR q	994,660	531,116	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	**tt**	**tt**	**ff**
				SR q	95,214	51,551	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**			
7	2	∞	**tt**	SNR 1p	170,734	66,051	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	-
				SNR q	1,287,932	689,024	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**
				SR q	112,146	61,401	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**			

†= out of memory ;

SNR 1p= Configuration with topology Sender-Node-Receiver, communicating through 1-place buffers ; SNR q= Configuration with topology Sender-Node-Receiver, communicating through FIFO queues ; SR q= Configuration with topology Sender-Receiver, communicating through FIFO queues

Table 1: Experiment results

The results for the blackbox (application) and sender point of view were expected, but not the one for the receiver.

To find the differences between both configurations for the receiver point of view, we have tried to hide more and more actions, randomly, until getting some new results.

The interesting results, presented in the Table 2 are that:

- adding the `NCF` messages to the list of hidden actions, the receiver points of view are equivalent for all tested configurations (except the case with 7 messages sent: the tool runs out of memory);
- with only `NAK` and `NCF` messages as visible actions, the receiver points of view are equivalent for all tested configurations (with the same restriction on memory).

Looking at the counter-examples and those new results, the explanation of the non-equivalence comes from the atomicity of `NCF` reply (the `NAK/NCF` mechanism is described in Section 2.3.3): when the receiver is plugged directly to the sender, a `NCF` is immediately followed by a `RDATA` (if the matching messages is still in transmit window), because the behavior of the source is to retransmit the `NCF` and just after the matching `RDATA` (in our model); whereas when the receiver is plugged to the node, it will receive a `NCF` when the node has received the `NAK`, but the `RDATA` is not sent immediately after, as the node will then transmit the `NAK` to the source, and the `RDATA` will be sent later.

# of mess	Window Size	AMB rate	Closing SPM	Without NCF	Only NAK and NCF
5	1	2	**ff**	**tt**	**tt**
5	1	2	**tt**	**tt**	**tt**
5	3	2	**ff**	**tt**	**tt**
5	3	2	**tt**	**tt**	**tt**
5	∞	∞	**ff**	**tt**	**tt**
5	∞	∞	**tt**	**tt**	**tt**
7	2	2	**ff**	†	†
7	2	2	**tt**	†	†
6	2	∞	**ff**	**tt**	**tt**
6	2	∞	**tt**	**tt**	**tt**

†= out of memory

Table 2: Equivalence between SNR and SR configurations with queues, from the receiver point of view

5 Deriving and Checking a Reliability Constraint

The PGM protocol is unreliable, and it can not be guaranteed that all messages will be recovered. Nevertheless, the "efficiency" of the protocol can be studied, *i.e.* looking at the performances under "reasonable" assumptions.

So, we tried to answer to the following question: is there a relation between the parameters of the system that implies that the system will recover all losses, under the assumption that only `ODATA` messages can be lost?

We first studied this question "manually", by looking at the reliability mechanism of PGM. We derived a non linear relation that relies five parameters of the protocol, and we verified it using IF toolset. This section describes in detail our work.

5.1 Condition for recovering all losses

What is the mechanism that allows to recover a loss? First (obvious) step, a message must be lost. Second, the receiver must get a message that signals that a previous message has been lost: it can be a `ODATA`, a `RDATA` or a `SPM` sent after the loss. Third, the `NAK` signaling the loss must be received before the corresponding data have been dropped from the sender memory. Then, the matching `RDATA` is sent and must be received.

Hence, in the property are involved the following parameters:

- the loss assumption on `ODATA`, `RDATA`, `SPM` and `NAK`;
- the transit delay of messages;
- the rate of `ODATA` (sender traffic policy) and `SPM` (ambient and heartbeat `SPM`); and,
- the transmit window policy.

Notice that the two first points are driven by the environment (we could make some assumption while modeling, but the implementation can not control them) and the two last are parameters of the protocol (that can be changed, up to the computer limits).

The most general relation is out of the scope of this study: it seems a first interesting step to consider one given "reasonable" configuration and to find the associated relation.

5.2 The model tested

The model tested is based on the SNR configuration of the section 4.4:

Topology: 1 source – 1 network node – 1 receiver

Policy of loss: N random `ODATA` lost (N is a parameter in the code)

Communication medium: FIFO queues with fixed delay

Transmission length: parameter of the code

Traffic shape: `ODATA` are sent at constant rate, one `ODATA` every `PROD_RATE` time unit

Group membership: the receiver is part of the group since the first time

Removed mechanisms: no heartbeat `SPM`, no filtering in nodes, no repeat of `NAK` (useless since `NAK` communication is reliable), no backoff before sending a `NAK`

Implementation choices:

> **Transmit window policy:** the sender tries to keep the maximum number of packets in memory, *i.e.* to keep "Window size" messages, minus one (keep free to receive data from the application). When the transmission is ended, drop one packet every `PROD_RATE` time unit.
>
> **ambient `SPM`:** the ambient rate is driven by the advance of the window. One `SPM` is sent each time `AMB_RATE` packets have been dropped out of the transmit window, (with `AMB_RATE` a parameter of the code).
>
> **closing `SPM`:** the "closing SPM" option is always turned on.

Note on "rate". There are two opposite views on the notion of rate. Formally, a rate is a number of actions per time unit. Nevertheless, while modeling or programming, it is easier to manipulate the waiting time between two actions. Since this waiting time is a way to encode the rate, the variables used in the model uses the suffix "`RATE`" instead of "`INTER_ACTION_WAITING_TIME`". But one consequence of this convention is that, to increase the production rate of `ODATA`, the variable "`PROD_RATE`" has to be decreased.

5.3 Deriving the relation

As presented in section 5.1, all losses are recovered if the `NAK` is received before the end of the lifetime of the data in the transmit window.

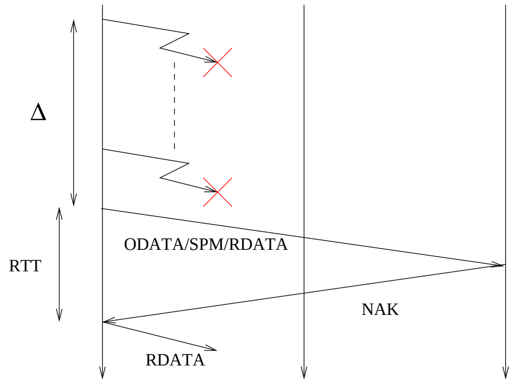


Figure 8: Pattern of loss recovering

The Figure 8 gives the pattern of a loss recovering in our model. The delay between the instant a message (that will be lost) is sent and the instant that a message (that will not be lost) is sent is called Δ. The delay between the sending of this message and the reception of the matching `NAK` is the Round Trip Time (RTT), assuming that a `NAK` is sent as soon as a loss is detected (it is the case in our model, but in a more realistic one, we should add the random backoff).

A message is recovered if this $\Delta + RTT$ is lesser than the lifetime (Lf) of the data.

What are the values of those three variables?

The value of Δ depends on the kind of the first non-lost message: `ODATA`, `RDATA` or `SPM`, *i.e.* $\Delta = \min(\Delta_{\texttt{ODATA}}, \Delta_{\texttt{RDATA}}, \Delta_{\texttt{SPM}})$.

We are not able to give an upper bound of $\Delta_{\texttt{RDATA}}$ (this could be ∞ if no message has been lost before).

For $\Delta_{\texttt{ODATA}}$, considering that no more that `NLOSS` messages can be lost, its maximum bound is $\texttt{NLOSS} * \texttt{PROD\_RATE}$.

Notice that this maximum bound is no more true at the end of the transmission: it is false for the `NLOSS` last messages (that can all be lost and not followed by any `ODATA`).

For $\Delta_{\texttt{SPM}}$, the upper bound is simply $\texttt{AMB\_RATE} * \texttt{PROD\_RATE}$, except for the first window (as the ambient `SPM` are only related to the advance of the window, and there is no regular advance of the window in the first window).

The lifetime of a data is also simple: $Lf = (\text{Window size} - 1) * \texttt{PROD\_RATE}$.

Then, the condition on parameters to recover all losses should be $Lf \geq \Delta + RTT$. Nevertheless, it is partially false. There is a problem of interleaving of actions at the date when a data is dropped: a data is not accessible during all its lifetime. Because in our model, the dropping action takes no time, the data is "first" ready and "after" lost. The condition is then $Lf > \Delta + RTT \iff$:

$$\text{Window Size} - 1 - \min\left(\texttt{NLOSS}, \texttt{AMB\_RATE}\right) > \frac{RTT}{\texttt{PROD\_RATE}} \tag{1}$$

Keep in mind that this relation is false on the first window (the evaluation of the $\Delta_{\texttt{SPM}}$ assumes a "constant" rate of SPM, and in our model, the SPM messages are sent only in case of window advance) and on the NLOSS last messages (the evaluation of the $\Delta_{\texttt{ODATA}}$ assumes that a ODATA will be sent in the future, and it is false at the end of the transmission).

5.4 Testing the relation on some configurations

The relation on parameters has been tested on some configurations: different series of experiments have been made, looking at the evolution of the losses while increasing or decreasing each of the parameters involved in the relation.

The results are presented in Tables 3, 4 and 5.

The Tables 3 and 4 are devoted to the study of the recovering based on the $\Delta_{\texttt{ODATA}}$ (assuming $\Delta_{\texttt{SPM}} = \infty$) and Table 5 studies the rate of ambient SPM.

Considering recovery based on ODATA In Tables 3 and 4, the row $|w| - 1 - l$ is the value Window Size $-$ 1 $-$ NLOSS, and the row $\frac{RTT}{Prod}$ is $\frac{RTT}{\texttt{PROD\_RATE}}$ *i.e.* the left and right part of the Equation 1 (because AMB_RATE $= \infty$ in all the configurations presented there).

The first three configurations tested just increase the size of the transmission. It shows that the last message is always lost. Basically, in all configurations in Tables 3 and 4, the NLOSS last messages can always be definitely lost (where NLOSS is the maximum number of messages that can be lost, given in the first row).

The second, third and fourth series of experiments (Table 3) just increase the production rate (they decrease the time between two emissions of ODATA, then increase the number of ODATA per time unit).

In the second series, the window size is 3 and the maximum number of losses is 1: the expression Window Size $-$ 1 $-$ NLOSS is 1. Then, for all configurations where $\frac{RTT}{\texttt{PROD\_RATE}} < 1$, the messages (except the last) are all delivered, never lost. When the threshold 1 is overtaken, all messages can be definitively lost.

The third series is the same as the second, except that both the window size and the number of losses have been incremented by one. As the difference is the same as in previous series, we have the same threshold behavior with 1.

The fourth series assume 2 losses and a window of size 5, *i.e.* the difference Window Size $-$ 1 $-$ NLOSS is 2. Then, the threshold behavior exists at value 2.

The fifth, sixth, and seventh series are in Table 4. They just decrease the window size, to show that a wide window allows to recover all losses and a small one does not. The relation given in Equation 1 is always verified.

The eighth experiment, the last in Table 4 increases the number of losses. For the 0, 1 and 2 losses, only the last messages can be definitely lost. With 3 losses, the Equation 1 is no more verified, and all messages can be lost.

Considering recovery based on SPM The Table 5 presents some experiment to test the effect of the ambient SPM rate, without the recovering mechanism driven by ODATA. To do so, in all configurations tested, the value of expression Window Size $-$ 1 $-$ NLOSS is 0 (and then can never be greater than the ratio RTT/Prod).

The first series increases the SPM ambient rate. In the first line, with no SPM (AMB_RATE $= \infty$), all messages can be definitively lost. On the opposite, with AMB_RATE $= 1$, all messages not in the first window will be delivered. The Equation 1 is confirmed.

# of losses	# of mess.	AMB rate	Window Size	RTT	Prod. rate	States	Trans.	Mess. always delivered						$\lvert w\rvert -$ $1-l$	$\frac{RTT}{Prod}$
								0	1	2	3	4	5		
Increase the size of the transmission															
1	3	∞	3	8	9	709	750	**tt**	**tt**	**ff**	-	-	-	1	0.89
1	4	∞	3	8	9	1,363	1,446	**tt**	**tt**	**tt**	**ff**	-	-	1	0.89
1	5	∞	3	8	9	2,107	2,240	**tt**	**tt**	**tt**	**tt**	**ff**	-	1	0.89
Evolution of the production rate															
1	6	∞	3	8	15	2,716	2,896	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	1	0.53
1	6	∞	3	8	10	5,027	5,935	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	1	0.80
1	6	∞	3	8	9	2,935	3,122	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	1	0.89
1	6	∞	3	8	8	8,671	10,535	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.00
1	6	∞	3	8	7	2,858	3,053	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.14
1	6	∞	3	8	5	3,324	3,656	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.60
1	6	∞	3	8	3	9,550	11,531	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	2.67
Evolution of the production rate (with wider window but same diff,)															
2	6	∞	4	8	15	9,577	10,932	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	1	0.53
2	6	∞	4	8	10	19,891	24,762	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	1	0.80
2	6	∞	4	8	9	10,632	12,069	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	1	0.89
2	6	∞	4	8	8	60,187	78,667	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.00
2	6	∞	4	8	7	11,211	12,796	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.14
2	6	∞	4	8	5	19,173	23,198	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1.60
2	6	∞	4	8	3	43,028	54,838	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	2.67
Evolution of the production rate (with wider window and bigger diff,)															
2	6	∞	5	8	15	8,153	9,281	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	2	0.53
2	6	∞	5	8	10	15,543	19,173	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	2	0.80
2	6	∞	5	8	5	18,494	22,726	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	2	1.60
2	6	∞	5	8	4	198,971	270,047	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	2	2.00
2	6	∞	5	8	3	45,027	57,698	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	2	2.67
2	6	∞	5	8	2	732,344	1,015,141	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	2	4.00

Table 3: Are all losses recovered? Effects of the production rate

# of losses	# of mess.	AMB rate	Window Size	RTT	Prod. rate	States	Trans.	Mess. always delivered						$\|w\|-1-l$	$\frac{RTT}{Prod}$
								0	1	2	3	4	5		
Evolution of the window size															
1	6	∞	5	8	9	2.118	2.235	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	3	0.89
1	6	∞	4	8	9	2.558	2.712	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	2	0.89
1	6	∞	3	8	9	2.935	3.122	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	1	0.89
1	6	∞	2	8	9	3.238	3.404	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	0	0.89
1	6	∞	1	8	9	845	915	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-1	0.89
Evolution of the window size (with faster source)															
1	6	∞	5	8	8	5.662	6.788	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	3	1
1	6	∞	4	8	8	7.698	9.310	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	2	1
1	6	∞	3	8	8	8.671	10.535	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	1	1
1	6	∞	2	8	8	5.194	6.054	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	0	1
1	6	∞	1	8	8	845	915	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-1	1
Evolution of the window size (with more losses)															
2	6	∞	6	8	20	6.375	7.267	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	3	0.4
2	6	∞	5	8	20	8.153	9.281	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	2	0.4
2	6	∞	4	8	20	9.577	10.932	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	1	0.4
2	6	∞	3	8	20	8.796	9.687	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	0	0.4
2	6	∞	2	8	20	9.678	10.222	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-1	0.4
2	6	∞	1	8	20	1.989	2.169	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-2	0.4
Evolution of the number of losses															
0	6	∞	4	8	20	344	344	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	3	0.4
1	6	∞	4	8	20	2.397	2.546	**tt**	**tt**	**tt**	**tt**	**tt**	**ff**	2	0.4
2	6	∞	4	8	20	9.577	10.932	**tt**	**tt**	**tt**	**tt**	**ff**	**ff**	1	0.4
3	6	∞	4	8	20	22.924	27.228	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	0	0.4
4	6	∞	4	8	20	33.014	39.636	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-1	0.4
5	6	∞	4	8	20	35.800	43.088	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-2	0.4

Table 4: Are all losses recovered? Effects of the number of losses and window size

# of losses	# of mess.	AMB rate	Window Size	RTT	Prod. rate	States	Trans.	Mess. always delivered											$\lvert w\rvert -$ $1-l$	$\frac{RTT}{Prod}$
								0	1	2	3	4	5	6	7	8	9	10		
Evolution of the rate of ambiant SPM																				
3	7	∞	4	8	6	234,090	308,894	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	-	-	-7	1.33
3	7	2	4	8	6	753,936	989,649	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	1.33
3	10	2	4	8	6	753,936	989,649	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	**tt**	**ff**	**tt**	**ff**	1	1.33
3	7	1	4	8	6	753,936	989,649	**ff**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	-	-	2	1.33
Evolution of the production rate																				
3	7	2	4	8	10	202,502	253,482	**ff**	**tt**	**tt**	**tt**	**tt**	**tt**	**tt**	-	-	-	-	1	0.80
3	7	2	4	8	9	116,142	136,320	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	0.89
3	7	2	4	8	8	855,154	1,125,118	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	1.00
3	7	2	4	8	7	117,731	138,085	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	1.14
3	7	2	4	8	6	753,936	989,649	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	1.33
3	7	2	4	8	5	347,583	437,608	**ff**	**ff**	**ff**	**tt**	**ff**	**tt**	**ff**	-	-	-	-	1	1.60
3	7	2	4	8	4	10,538,131	14,372,868	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	-	-	1	2.00
3	7	2	4	8	3	2,202,437	2,879,437	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	**ff**	-	-	-	-	1	2.67

Table 5: Are all losses recovered? Effects of the ambiant rate of `SPM`

An interesting point is that, even if the Equation 1 is not satisfied (with `AMB_RATE` = 2 on the second line for example), some messages are always recovered (one out of two). In fact, the message sent just before the `SPM` can always be recovered, because the sum $\Delta_{SMP} + RTT$ is small enough for this message.

The second series increases the production rate. We have the same kind of behavior that for the first series: when the relation is satisfied, all messages are received, and when increasing the data rate, more and more messages can be definitively lost.

Considerations on time and space. It can be noticed that, in some series, the size of the state space increases globally, but not locally. It is particularly visible on second, third and fourth series of Table 3.

This global increase seems "normal": there are more and more messages in the system at a time as the production rate increases, which implies that the system is more "complex".

Nevertheless, there are some local decreases: in the second series, when `PROD_RATE` comes from 10 to 9, from 8 to 7, in the second series when `PROD_RATE` comes from 10 to 9, from 8 to 7. Each time, from an even to an odd value.

Our explanation is that this is linked to the fact that the RTT is even (in fact, the delay of each queue is 2): if the production delay (`PROD_RATE`) is also even, there are more events which can interleave at even dates than if the production delay is odd.

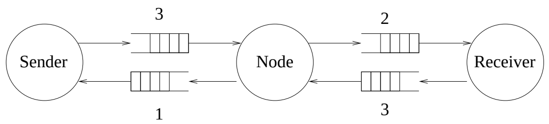


Figure 9: The delays of the experiments given in Table 6

To confirm this hypothesis, the same series of experiments as in Table 3 have been made, changing just the transit delays. They are given in Figure 9[8]. The total gives a RTT value of 9. The results are in Table 6.

It can be clearly seen that the divisibility of the RTT by the production rate has a great influence on the size of the state space.

In the first series of experiments (in which the size of the transmission is increased), with RTT=`PROD_RATE` =9 the state space is about twice bigger than with RTT=8 and `PROD_RATE` =9.

In the second series of experiment, we clearly have a smaller state space with RTT=9 (whereas increasing the delay should increase the complexity of the system), except for production rates 9, 7, 5 and 3. It can be explained by the fact that, for RTT=8, 8 and 7 are coprime, and also for 8 and 5: the non-interleaving gain is the same for both systems, and a system with a greater RTT is more complex. On the opposite, when the RTT and the production rate are not coprime (RTT=9 and `PROD_RATE` =9 or 3), the situation is far worse.

We can do the same kind of comparisons with the third and forth series, and we can also compare the timed and untimed versions.

We can compare the SNR q configuration of the "removing node" experiment (Section 4.4, with only 1 loss) and this timed version.

We can see that, except for a high rate of `ODATA`s, the state space is bigger in untimed systems that in the timed one. It can be explained by the fact that the timing constraints

[8]Downstream 3 from sender to node, 2 from node to receiver; upstream: 3 from receiver to node and 1 from node to sender.

# of losses	# of mess.	AMB rate	Window Size	RTT	Prod. rate	States	Trans.
Increase the size of the transmission							
1	3	∞	3	9	9	1,199	1,291
1	4	∞	3	9	9	2,677	2,929
1	5	∞	3	9	9	4,329	4,753
Evolution of the production rate							
1	6	∞	3	9	15	2,473	2,581
1	6	∞	3	9	10	2,957	3,202
1	6	∞	3	9	9	6,149	6,753
1	6	∞	3	9	8	3,372	3,854
1	6	∞	3	9	7	3,356	3,646
1	6	∞	3	9	5	10,793	13,657
1	6	∞	3	9	3	34,968	44,658
Evolution of the production rate (with wider window but same diff,)							
2	6	∞	4	9	15	6,937	7,411
2	6	∞	4	9	10	9,008	10,061
2	6	∞	4	9	9	30,659	35,122
2	6	∞	4	9	8	12,197	14,722
2	6	∞	4	9	7	14,918	17,542
2	6	∞	4	9	5	29,751	36,675
2	6	∞	4	9	3	128,329	166,095
Evolution of the production rate (with wider window and bigger diff,)							
2	6	∞	5	9	15	5,867	6,233
2	6	∞	5	9	10	7,331	8,111
2	6	∞	5	9	5	22,162	26,936
2	6	∞	5	9	4	17,078	20,725
2	6	∞	5	9	3	125,125	159,150
2	6	∞	5	9	2	37,777	45,744

Table 6: Same experiments that in Table 3 with RTT=9

# of losses	# of mess.	AMB rate	Window Size	Prod. rate	RTT=8		RTT=9	
					States	Trans.	States	Trans.
1	5	2	3	9	4,737	5,119	16,666	20,719
1	5	2	3	8	16,666	20,719	4,611	5,008
1	5	2	3	5	5,514	6,182	11,785	14,781
1	5	2	3	4	85,391	116,517	22,207	26,61
1	5	2	3	3	18,383	22,568	150,571	205,007
1	5	2	3	2	1,406,428	1,935,306	218,920	276,457
1	5	2	3	untimed	1,345,093 / 2,594,217			
1	7	∞	2	9	4,394	4,619	6,800	7,880
1	7	∞	2	9	6,800	7,880	4,569	4,824
1	7	∞	2	9	5,133	5,561	12,427	15,497
1	7	∞	2	9	43,420	58,878	14,158	16,451
1	7	∞	2	9	13,311	15,621	85,807	116,624
1	7	∞	2	9	1,094,280	1,531,442	85,807	116,624
1	7	∞	2	untimed	689,024 / 1,287,932			

Table 7: Comparing timed and untimed systems

reduce the interleaving of actions between processes: for example, the production rate with the upper bound of queues limits the number of data in the queues, and therefore the reactions of the node and receiver.

5.5 Conclusion and future work

Before the general conclusions about the whole case study, it is interesting to conclude about the results obtaining while using the existing tools for finite and timed systems. The first point is that, even with tools which don't take into account infinite behaviors, it is possible to verify interesting properties, as we just showed. More specifically, we studied the following points:

- First, we showed that the minimal property that losses get signalled at the end of the session is not necessarily respected. It will be necessary for this property to be respected that a "closing SPM" mechanism is implemented.
- Then, we looked at the role of the nodes elements in the protocol. If this role is obvious as relay for the messages, it wasn't clear if they could be suppressed in the study of the protocol: do they have more effect than simple relays? In fact, we showed that nodes have to be taken into account when studying the protocol: they introduce a decoupling between the `NCF` answer and the `RDATA` messages that follow them which implies a more complex behavior than if the nodes are suppressed.
- Last, we tried to guess what should be the relations between the various parameters of the protocol for ensuring the recovery of all losses. The exhaustive simulation tool for timed systems enabled to verify this relation on a whole set of different configurations.

But this study also reveals the well-known limitations of existing tools. That is, the studies were done with finite-stated models, whereas the nature of the protocol implies that infinite behavior should be studied. This would be the goal of the following studies.

First, in section 6, we will study the protocol with the number of transmitted packets being a parameter, whereas the previous study uses at most something like 10 messages. Second, in section 7, we will study the protocol with any kind of topology, whereas the previous study uses a fixed topology of one sender, one node, one receiver. Last, it should be interesting to derive automatically a relation for full reliability. The "manual" proof shows that such a relation may be complex, since non linear on integer parameters (undecidable theory). So we have to use new abstractions and tools that are able to deal with such relations, *e.g.* TREX.

Another point that should be discussed is the usability of the tools in the telecommunication industry.

As we have seen in 3, the version of the IF language designed in the Advance project is powerful and expressive enough to have simple and complete descriptions of the PGM protocol. It can be argued that those descriptions are better than those done in the SDL language, in part because of the cleaner syntax of the IF language. A graphical syntax could be a nice add-on to this language. Also, the lack of a "loop" construct in the automaton framework has been resented as limiting. Nevertheless, it is possible to call external C code, which alleviates this limitation.

The associated tools enable to realize precise studies and verifications of the behavior of specifications. Some limitations have been encountered during this first study. The more stringent are the following ones:

- Some properties to be verified had been described using the μ-calculus in the `evaluator` tool. A simpler language for expressing properties could be investigated.
- The outputs of the verification tools are difficult to handle. A "debugging" mode would be interesting, with step-by-step reachability and labelling of the transitions.

6 Verification with infinitely many packets

6.1 Introduction

The *Liège Automata-based Symbolic Handler (LASH)* is a tool for building and manipulating finite-state representations of sets, and for exploring the state-space of infinite-state systems. This document describes analysis experiments of a simple IF model of the PGM protocol using LASH.

6.2 Protocol Model

6.2.1 Restrictions

The state-space exploration technique implemented in LASH is not able to handle the general IF specification of PGM as provided by VERIMAG and France Telecom. The following restrictions have been applied in our model:

- All processes are instantiated statically. Indeed, LASH does not handle dynamic features, because it relies on *meta-transitions* which can currently only be computed on static structures.
- The integer data type is the only one to be handled. This is due to the symbolic representation system used in the current version of the state-space exploration module of LASH. As a consequence, clocks, timers and signals have been abstracted out of our model.

6.2.2 Structure

The model described in this section is written in IF 1.0. There are four static processes: *Init*, *Source*, *Network Element*[9] and *Receiver*, communicating with each other by means of synchronizations. The processes *Source*, *Network Element* and *Receiver* represent respectively a transport-layer originator of PGM data packets, a network-layer entity and a transport-layer consumer of PGM data packets. The process *Init* is used to initialize the system. No additional source, network element or receiver is instantiated during execution.

Because of this static structure, the `SPM` messages have been abstracted out. These messages carry the source-assigned information regarding the edges of the source's transmit window (`TXW_TRAIL` and `TXW_LEAD`), and the information regarding the path followed by the messages from the source to the receiver. The former element is merely a duplication of the same information embodied in the `ODATA` and `RDATA` messages, and the latter one is constant in our model.

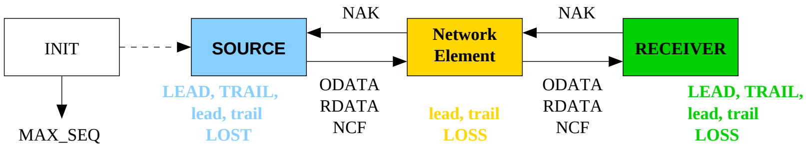


Figure 10: Abstracted PGM model.

6.2.3 Behavior

The Source sends `ODATA` messages to the Network Element, which forwards them directly to the Receiver. If the Receiver identifies a missing message with a sequence number still belonging to the transmit window, it sends a `NAK` to the Network Element, until it gets back a `NCF`. Upon reception of a `NAK`, the Network Element sends a `NCF` to the Receiver, establishes a repair state for the `NAK` if it does not already exists, and sends `NAK` messages to the Source until it eventually gets back a `NCF`. Finally, if the missing message sequence number is still within the transmit window, the Source sends a `RDATA` to the Network Element, which forwards it to the Receiver.

6.2.4 Transmit and Receive Windows

Any message is associated with a sequence number ranging from 0 to a bound called `TXW_MAX_SEQ`. This bound can either be a specific value, or a parameter.

The *Transmit Window* maintained by the Source is delimited by two sequence numbers, `TXW_TRAIL` and `TXW_LEAD`. Any message with a sequence number that belongs to this interval can be "repaired", i.e., sent as an `RDATA` in the case of a message loss. `TXW_LEAD` is incremented each time a `ODATA` is sent, and `TXW_TRAIL` is incremented non-deterministically. The maximum window size is fixed by the variable `TXW_MAX_RTE` (in our model). When `TXW_LEAD` or `TXW_TRAIL` reaches `TXW_MAX_SEQ`, it is reset to 0.

The Network Element does not maintain any window (any message `ODATA` received is thus forwarded directly), and only `RDATA` corresponding to the repair state (if any) is forwarded.

[9]That is, a node of the network.

Finally, the Receiver maintains a *Receive Window*, delimited by the variables RXW_TRAIL and RXW_LEAD. When an ODATA is received, RXW_LEAD is updated according to the value of the *lead* field of the message. RXW_TRAIL is incremented non-deterministically but, if its value corresponds to an older message than the oldest message available for repair from the source (as indicated by the *trail* field of the RDATA and ODATA messages), RXW_TRAIL is set equal to *trail*. As in the case of the Transmit Window, when RXW_LEAD or RXW_TRAIL reaches TXW_MAX_SEQ, it is reset to 0.

6.2.5 Message Transmission

The messages are transmitted from one process to another by means of *synchronizations*. One gate is used per message (ODATA, RDATA, NAK, NCF) and per direction, hence a total of eight gates are used for the communication. In addition, one gate is used internally in order to generate an initial non-deterministic value for the parameter TXW_MAX_SEQ. The synchronization mechanism acts like a queue of zero size (i.e., a rendez-vous), preserving the order of messages.

6.2.6 Loss Modeling

In our model, at most one ODATA is lost non-deterministically between the Source and the Network Element. The sequence number of the lost message corresponds to the value of the LOSS_SEQ_NB variable.

The Receiver detects a loss as soon as the sequence number of a received ODATA is not contiguous to the sequence number of the last accepted ODATA. This is done by comparing the *lead* field of the message with RXW_LEAD. Once a loss is detected, the sequence number is stored in a local variable (LOSS). This variable is set to -1 as long as no loss is pending. If the trailing edge of the Receive Window moves beyond the sequence number of the loss, or if an RDATA corresponding to the loss is received, data recovery for that sequence number terminates by resetting LOSS to -1.

The Network Element creates a repair state as soon as it receives a NAK from the receiver. This repair state is modelized by a variable that holds the value of the loss sequence number. As soon as a RDATA corresponding to the repair state is received by the Network Element, the repair state is removed by resetting LOSS to -1.

6.3 Analysis

6.3.1 Goals

Our verification experiment pursues two goals. The first one is merely technical and can be viewed as a debugging tool. The goal is to get knowledge of the model behavior by analyzing the reachable values of the auxiliary variables (mostly required to ensure a correct resetting of the windows boundaries), and to correct undesired behaviors. This analysis also encompasses the performance impact of the meta-transitions, which are required in order to explore infinite state space with finite resources. Since the meta-transition computation algorithms are costly both in time and memory, a trade-off is often required for the number of meta-transitions that must belong to the model.

The second verification goal concerns the analysis of the state space and, specifically, the relationships between the variables and the global and local properties of the system. Specifically, our main goal was to study closely the relationships between the LEAD and TRAIL values of the windows.

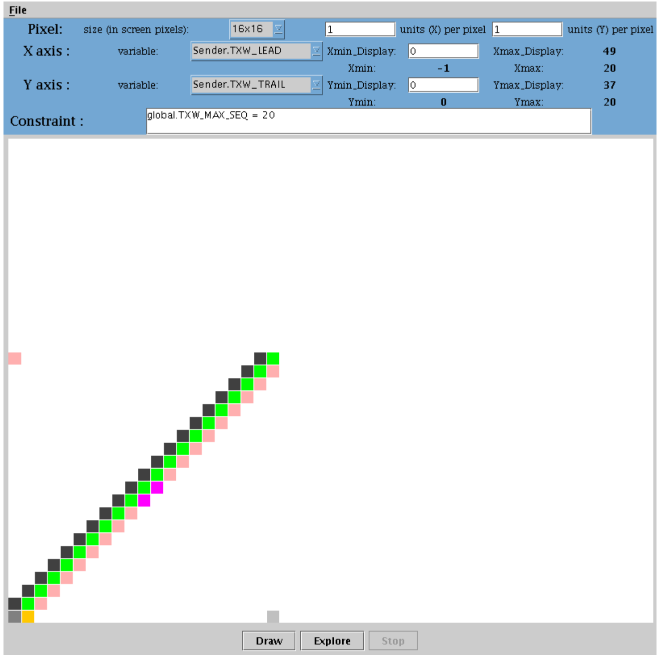


Figure 11: Transmit Window.

6.3.2 Tools

The LASH toolset provides two means for analyzing the state space of the system. The first method consists in inserting some assertion within the IF code of the model, which is then either validated or disproved by the state-space exploration. A second method, more interactive, relies on a prototype tool for visualizing the reachable values of a given pair of variables. In this document, we will mainly use the latter.

6.3.3 Experiments

The PGM model described in 6.2 has been fitted with a Transmit Window size (`TXW_MAX_RTE`) of 2, and with a loss sequence number set to 10. Two sets of experiments were conducted on the resulting model. We started by setting explicitly a maximum sequence number `TXW_MAX_SEQ` for messages. In a second phase, we let `TXW_MAX_SEQ` be a free parameter of the model (which, in this case, is infinite-state).

Specific `TXW_MAX_SEQ`. We chose the value 20 for `TXW_MAX_SEQ` because it is large compared to the Transmit Window size (in our model, 2), but still manageable.

Figure 11 displays the reachable values of the variables `TXW_LEAD` (x-axis) and `TXW_TRAIL` (y-axis) of the Source, each color corresponding to a different exploration step. The lower left corner of the graph corresponds to the pair (0,0), and the top left to the pair (20, 20). Figure 12 displays the evolution of this graph during the state space exploration.

Three interesting observations can be inferred from this picture. First, the resetting of the variables has been successfully implemented and amounts to a modulo operation. Second, although the transmit window size is 2, each value of one variable corresponds to three possible values for the other one. Finally, the effect of a meta-transition is noticeable. Indeed, all the pairs of identical values are reached after a single exploration step. The cyclic path represented by this meta-transition corresponds to the following behavior of

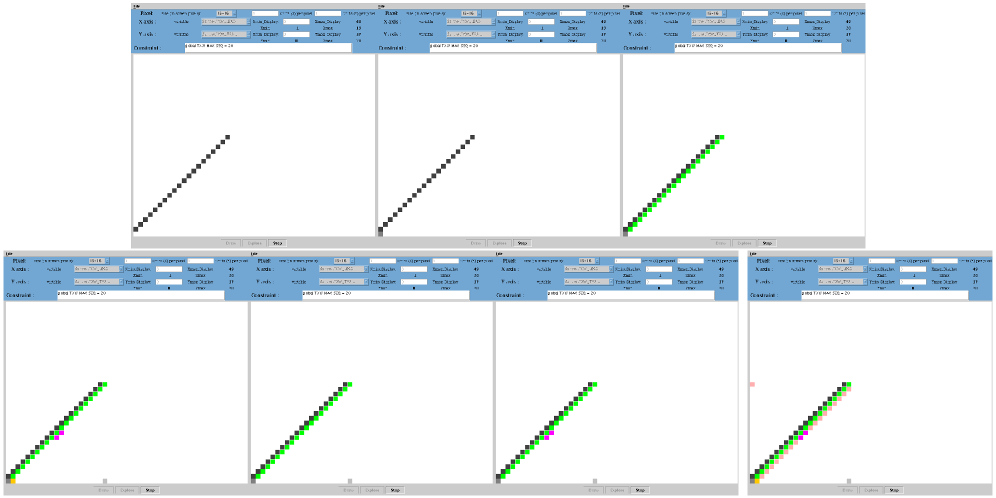


Figure 12: Transmit Window: Sequence during state-space exploration.

the system : The source sends an `ODATA`, which is forwarded by the Network Element to Receiver, and the `TXW_TRAIL` is then incremented.

Let focus now on the Receive Window. Figure 13 displays the variables `RXW_LEAD` (x-axis) and `RXW_TRAIL` (y-axis) of the Receiver.

An interesting observation is the breadth of the diagonal strip, which indicates that for each value of one variable the other one may take four different values. An exception to this rule is the extra value noticeable at (11,8), corresponding to the possible message loss. Indeed, when the receiver gets an `ODATA`, it updates first its variable `RXW_LEAD` and then updates `RXW_TRAIL`. So, when a loss appears, the gap between the updated value of `RXW_LEAD` and the "old" `RXW_TRAIL` is maximum.

As an example of the possibilities offered by LASH, we display in Figure 15 the values of the reachable bounds of the Transmit Window in the particular case of the extreme value (11,8) reached by the Receive Window. (This displayed set of values is simply obtained by applying elementary set operations on the sets previously generated during the state space exploration.) The advantage of this approach is that the computation of the reachable state space of the model has only to be performed once. Visualizing interesting features of this set, such as the one discussed here, is then fast (a few seconds) and easy.

`TXW_MAX_SEQ` as a parameter. In those series of tests, we do not fix the value of `TXW_MAX_SEQ`, which is then chosen non-deterministically among all the positive integers prior to running the model. Obviously, since the model becomes infinite-state, its exploration with a finite amount of resources is only possible if it contains suitable meta-transitions.

By taking the same value (10) for `LOSS_SEQ_NB` and `TXW_MAX_RTE`, we obtain the Figures 16 and 17, corresponding respectively to Figures 11 and 13.

This parametrized model generalizes the models in which a specific value of `TXW_MAX_SEQ` is specified. Notice that any pair of values appearing in the Figures 11 and 13 appears also in the Figures 16 and 17. This provides an efficient mean to test for several values of a parameter. It is sufficient to explore the parametrized system and construct its set of reachable states, and in a second phase, constrain the parameter to be equal to the desired values.

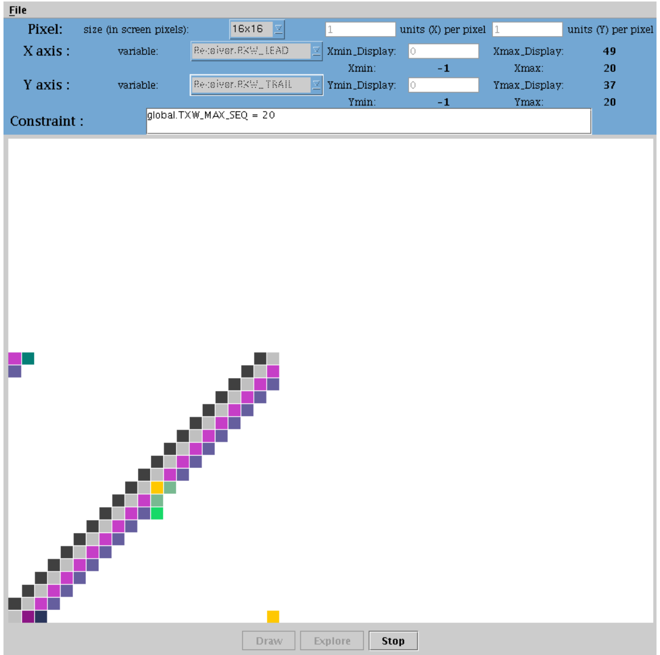


Figure 13: Receive Window.

Figure 14: Receive Window: Sequence during state-space exploration.

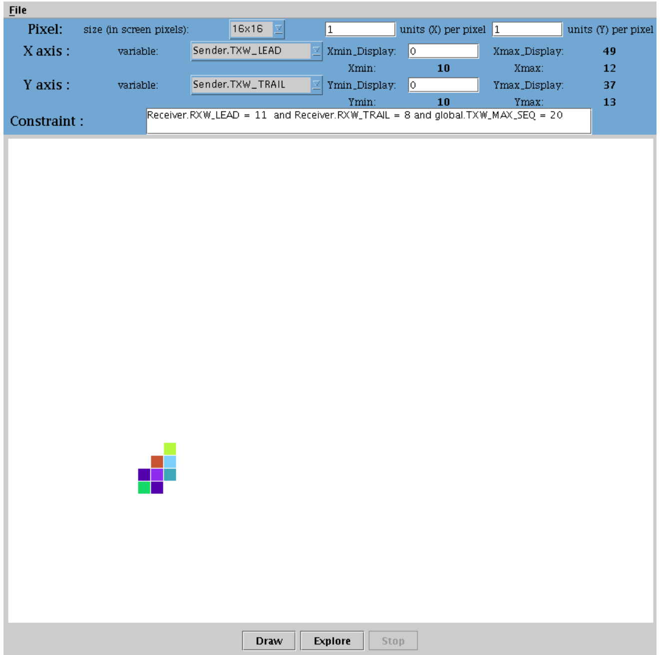


Figure 15: Constrained Transmit Window.

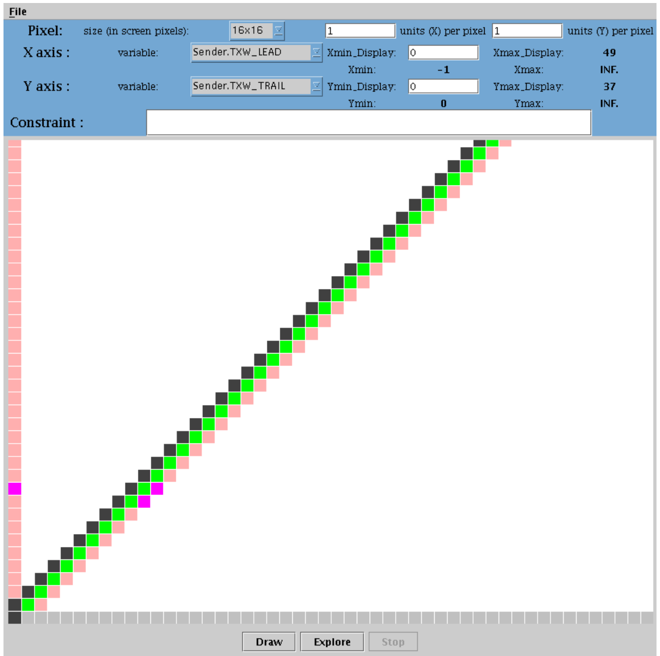


Figure 16: Transmit Window in parametrized case.

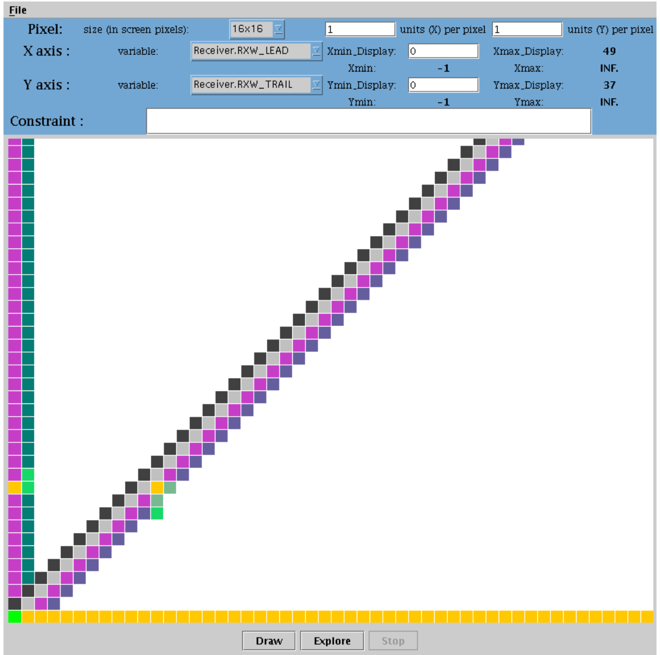


Figure 17: Receive Window in parametrized case.

TXW_MAX_SEQ	Nb_steps	Nb. NDD States	Nb. States	Max. Mem.	Time
20	62	100 K	26 K	125 Mb	7h42
200	62	160 K	249 K	283 Mb	10h19
2000	62	219 K	2,479 K	467 Mb	17h09
20000	62	294 K	24,781 K	719 Mb	35h59
ND	76	225 K	INF.	279 Mb	23h52

Table 8: Analysis costs

6.4 Analysis Costs

Table 8 summarizes the analysis costs for the model described in this document. (In the first part of the table, the value of `TXW_MAX_SEQ` is explicitly specified in the analyzed model; the last line corresponds to the parametrized case.) For each value of `TXW_MAX_SEQ`, the table gives the number of exploration steps before reaching a fixpoint, the number of internal states of the finite-state representation of the set of reachable states, and the number of elements in this set. The time and memory costs are also given.

On a logarithmic scale, the number of reachable states follow linearly the value of `TXW_MAX_SEQ`. This behavior is explained by noticing that the windows bounds remain close to the current sequence number of messages, which ranges from 0 to `TXW_MAX_SEQ`. The values of the other auxiliary variables do not depend on `TXW_MAX_SEQ`. An interesting feature is that the number of exploration steps remain constant when `TXW_MAX_SEQ` grows. This indicates that the meta-transitions manage to capture the repeated incrementation of ODATA sequence numbers. The size of the internal representation is logarithmic in the value of TXW_MAX_SEQ. This indicates that the internal representation handles adequately this type of system behavior captured by the meta-transitions.

The behavior of the parametrized system is worth noticing. Although the result of its state-space exploration contains sufficient information for building exactly the set of reachable states of every model in which the value of TXW_MAX_SEQ is fixed, the resources needed by the computation are similar to those required by a fixed value of 2000.

6.5 Property Validation

After this global analysis of the system behavior, we can now write down a property which sets precisely the range of values that the Receive Window size is thought to take, and check that this property holds in all model executions. The property that has been chosen is given below. Intuitively, there are two possible situations, whether a loss is pending or not. For each of these situations, there are three cases: a nominal one (the diagonal strip on the figures), a case in which only RXW_LEAD has been reset, and a case in which only RXW_TRAIL has been reset.

```
(Sender.LOST = 0 and
  ( ( Receiver.loop_LEAD = 0 and Receiver.loop_TRAIL = 0 and
      Receiver.RXW_LEAD - Receiver.RXW_TRAIL <= 2 and
      Receiver.RXW_LEAD - Receiver.RXW_TRAIL >= -1)
    or
    ( Receiver.loop_LEAD = 1 and
      Receiver.RXW_LEAD + global.TXW_MAX_SEQ - Receiver.RXW_TRAIL <= 2 and
      Receiver.RXW_LEAD + global.TXW_MAX_SEQ - Receiver.RXW_TRAIL >= -1)
    or
    ( Receiver.loop_TRAIL = 1 and
      Receiver.RXW_LEAD - global.TXW_MAX_SEQ - Receiver.RXW_TRAIL <= 2 and
      Receiver.RXW_LEAD - global.TXW_MAX_SEQ - Receiver.RXW_TRAIL >= -1)))
or
(Sender.LOST = 1 and
  ( ( Receiver.loop_LEAD = 0 and Receiver.loop_TRAIL = 0 and
      Receiver.RXW_LEAD - Receiver.RXW_TRAIL <= 3 and
      Receiver.RXW_LEAD - Receiver.RXW_TRAIL >= -1)
    or
    ( Receiver.loop_LEAD = 1 and
      Receiver.RXW_LEAD + global.TXW_MAX_SEQ - Receiver.RXW_TRAIL <= 3 and
      Receiver.RXW_LEAD + global.TXW_MAX_SEQ - Receiver.RXW_TRAIL >= -1)
    or
    ( Receiver.loop_TRAIL = 1 and
      Receiver.RXW_LEAD - global.TXW_MAX_SEQ - Receiver.RXW_TRAIL <= 3 and
      Receiver.RXW_LEAD - global.TXW_MAX_SEQ - Receiver.RXW_TRAIL >= -1)))))
```

This property is validated by generating the set of values satisfying its negation, and then checking whether the intersection of this set by the projection of the computed set of reachable values over the relevant variables is empty. Once the reachable part of the state space is computed, the projection, intersection and test emptiness operations take only a few seconds.

6.6 Conclusions

We have conducted successfully experiments in which abstracted versions of a PGM model have been analyzed, both in the case of a fixed bound on the message sequence numbers, and with a parametrized model. The methodology described in this document indicates the essential feedback provided by our prototype visualization tool.

Finally, the performance analysis indicates that running LASH is still very time consuming. The positive aspect is that the state-space exploration needs to be achieved only once, since checking a property on a computed representation of the reachable state-space can be done with little additional cost.

7 Verification with infinitely many topologies

We considered an abstract version of the PGM protocol without time. The PGM protocol consists of a sender, a receiver and a network element process. It is parameterized and can be instantiated in form of any tree that has the sender at the root, receivers at leaves and network elements at all internal nodes. So there is an infinite set $\mathcal{N}(PGM)$ of instantiations of the PGM protocol. Verifying a property for a parameterized protocol means showing that any instantiation satisfies the property.

We addressed the problem of dealing with instantiations of arbitrary shape by reducing the instantiations that have to be considered. More precisely, we identified subsets $\mathcal{N}'(PGM) \subseteq \mathcal{N}(PGM)$ such that all instantiation in $\mathcal{N}'(PGM)$ satisfies the property if and only if all instantiations in $\mathcal{N}(PGM)$ satisfy the property. Thereby, we restricted ourselves on properties that speak only of the sender and the receivers.

7.1 Reduction to simple instantiations for the PGM protocol

We started by proving that it suffices to consider the set of instantiations $\mathcal{N}^1(PGM)$ that contains all trees of depth at most 3, i.e. all instantiations with only one layer of network elements. The proof idea is to show that any $T \in \mathcal{N}(PGM)$ can be simulated (with respect to the sender and the receivers) by some $T' \in \mathcal{N}^1(PGM)$.

Theorem 1 ([9]) *Let ϕ be a property speaking only about the sender and receivers. $\mathcal{N}(PGM) \models \phi$ if and only if $\mathcal{N}^1(PGM) \models \phi$.*

While instantiations in $\mathcal{N}^1$ are easier to handle than general instantiations, $\mathcal{N}^1$ is still infinite because there can be any number of receivers. In a second step, we observed that if one is only interested in the behavior of a bounded number of receivers, and if the number of messages of the protocol is finite, it is possible to restrict the number of receivers that have to be considered. We showed that if the number of data packets sent in the PGM protocol is bounded, then there is a particular element $T \in \mathcal{N}^1$ for which it suffices to check the property. This works because the network element and the receiver of the PGM protocol can iterate to send a particular message any number of times. (Such behavior can occur if the agent does not receive messages any more, i.e. all messages sent to it are lost.)

Theorem 2 ([9]) *Let ϕ be a property speaking only about the sender and k receivers, and let the PGM protocol be restricted to a finite set of data packets. Then there is an instantiation $T \in \mathcal{N}^1$ such that $T \models \phi$ if and only if $\mathcal{N}^1(PGM) \models \phi$.*

7.2 Generalizing the approach

In the next step, we wanted to formulate the approach we took for verification of the PGM protocol in an abstract way. Thereby be made clear under which conditions such reductions are possible.

In [6], we defined an abstract framework for multicast protocols with instantiations in form of trees.

For a multicast protocol P, let $\mathcal{N}(P)$ be the set of all instantiations, and let $\mathcal{N}^1(P)$ be the instantiations with only one layer of internal nodes. We say that a protocol P is collapsible if it suffices to consider instantiations in $\mathcal{N}^1(P)$ in order to prove properties about $\mathcal{N}(P)$ that only speak about the sender and receivers.

The main part of [6] consists of defining conditions under which a multicast protocol is collapsible. The main restriction concerns the network elements. In the PGM example, network elements can forward incoming messages, i.e. multicast messages from above to successors, and send messages from below to the predecessor. Thereby, downward messages can be filtered, i.e. during a multicast, it depends on the history of a channel whether a message is sent down to it. Furthermore, a network element can 'generate' a message, i.e. send a message m' when receiving another message m (or no message), depending on its internal state. We defined this type of functionality of network elements in the abstract setting and gave restriction on the filtering and generating of messages that suffice to make a protocol collapsible.

Theorem 3 ([6]) *For a collapsible multicast protocol and a property speaking only about the sender and receivers, the following holds:* $\mathcal{N}(P) \models \phi$ *if and only if* $\mathcal{N}^1(P) \models \phi$.

Our aim was to obtain clear, simple conditions under which reduction is possible. This lead us to a formulation of collapsible multicast protocol that is only satisfied by a slight variant of the PGM protocol. But we felt that it is interesting to consider a variant of the PGM protocol for which collapsing is simple to prove.

We also formulated the reduction to only one instance of the protocol in the abstract setting, and obtained the following:

Theorem 4 ([6]) *Let* ϕ *be a property speaking only about the sender and* k *receivers, and let* P *be a protocol with a finite set of messages. Then there is an instantiation* $T \in \mathcal{N}^1(P)$ *such that* $T \models \phi$ *implies that* $\mathcal{N}^1(P) \models \phi$.

If the receiver and the network element agent can iterate sending messages, then the other direction holds as well.

7.3 Conclusion and future work

This section presents an important result about multicast protocols in general. This result has not been obtained by using an automatic tool, but by a mathematical reasoning. We have studied conditions which enables to derive properties for all kinds of instantiations of multicast protocols from the simplest one of the instantiations: the protocol reduced to one sender, one node, and one receiver. Unfortunately, those conditions are verified only for a restricted version of PGM. Nevertheless, this first study shows the way for generalizing the results we obtain previously to all sets of multicast topologies.

At the moment, the abstract formulation of collapsible multicast protocols is motivated by the PGM protocol. We plan to investigate in a more systematic manner under which reductions described here are possible for multicast protocols.

8 Conclusion

At the end of this report, we draw some conclusions on the work done until now on the PGM protocol in the framework of the Advance project.

An important effort has been done on the modelization of the protocol. Indeed, the initial description in SDL contains a lot of details that are not useful for the verification of the PGM properties and make the verification task difficult. A first result in this direction was to identify all the sources of complexity in the protocol and to find which are the possible abstractions that preserve the properties to be verified. Then, several models have been designed by applying some of the abstractions found. In this way, we were able to apply the different tools of the project on common models. This process allowed to test the new version of the IF language. The conclusion is that it is able to model accurately complex telecommunication protocols. The PGM example has been completely modeled, from an initial SDL version. The main advantage upon the previous version is an easier description of parametrized architectures. The main interest that comes from the existence of IF is that a whole set of tools are being developed, which will take as entry points specifications described in the IF language. To have a common language implies that it is not necessary to have several developments for using different tools, using languages specific to each tool.

The first series of experiments addressed the verification of the PGM protocol using finite-state untimed models. This study enabled us to detect an erroneous behavior in the specification of the protocol (for the end of the transmitting session) and to propose a mechanism to avoid it. Also, we have learned a lot about the behavior of the protocol thanks to these first experiments in the finite-state framework. In particular, we could show that the network elements cannot be fully abstracted in the presence of `NCF` messages.

The breakthrough of our project is to address parametric verification problems of the PGM, and to be able to solve these problems for the first time, by considering several different parameterization aspects and infinity sources: topology and unbounded number of components, infinity of the number of sent data, and quantitative constraints on timing and window size parameters.

The first important and new result was the derivation and the verification using model-checking tools (IF-tools) of a constraint ensuring the reliability of the protocol. The derived constraint is nonlinear and relates five integer parameters of the protocol.

The second important result was to establish automatically using LASH that the window policy is respected whatever is the number of data packets.

Finally, the third major result was the definition of a mathematical framework allowing to obtain the conditions under which the properties proved on a simple architecture are also true for all possible tree-like architectures. This allows to prove that the PGM is correct for all possible architectures provided it is correct for some few simple instances.

Future work on the PGM includes:

- Automatic synthesis of the parametric constraints for the reliability property using TReX and LASH.
- Quantitative analysis of the protocol, e.g., (1) verification of the reliability up to some percentage assuming some loss rates, and (2) synthesis of parameter values (timing, loss rates, etc) in order to minimize the number of control messages per data messages.

The achievement of these goals requires qualitative and quantitative extensions and improvements of our techniques and tools.

References

[1] Advance project home page. On the WWW at `liafa.jussieu.fr/haberm/ADVANCE/main.html`.

[2] Ahmed Bouajjani Aurore Annichini and Mihaela Sighireanu. Trex: a tool for reachability analysis of complex systems. In *Proceedings of CAV'2001*, volume 2102 of *LNCS*. Springer-Verlag.

[3] Gérard Babonneau. Pgm reliable transport protocol specification. Technical Report BD/CNET/DSM/RDC//98/NLM, CNET, 1999.

[4] M. Bozga, J.-Cl. Fernandez, S. Ghirvu, S. Graf, J.-P. Krimm, and L. Mounier. If: a validation environment for timed asynchronous systems. In *Proceedings of CAV'2000*, LNCS. Springer-Verlag.

[5] Cadp home page. On the WWW at `www.inrialpes.fr/vasy/cadp`.

[6] Javier Esparza and Monika Maidl. Simple representative instantiations for multicast protocols. To be submitted in september 2002.

[7] Jean-Claude Fernandez and Laurent Mounier. Verifying bisimulation on the fly. Technical report, LGI-IMAG, 1990. Distributed with the CADP package.

[8] If home page. On the WWW at `www-verimag.imag.fr/async/IF/index.html`.

[9] Monika Maidl. Simple representative instantiations for the pgm protocol. Unpublished.

[10] Tony Speakman, Dino Farinacci, Jon Crowcroft, Jim Gemmell, Steven Lin, Dan Leschiner, Michael Luby, Alex Tweedly, Nidhi Bhaskar, Richard Edmonstone, Todd Montgomery, Luigi Rizzo, Rajitha Sumanasekera, and Lorenzo Vicisano. Pgm reliable transport protocol specification. RFC 3208, IETF, 2001.