Electronic Notes in Theoretical Computer Science 9 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume9.html 9 pages

The Bounded Retransmission Protocol Revisited

(Extended Abstract)

Revised July 1, 1999 [1]

Dennis Dams and Rob Gerth

Dept. of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven, The Netherlands

Abstract

We demonstrate the use of abstract interpretation techniques in model checking by verifying Philips' Bounded Retransmission Protocol for all parameter instances.

1 Introduction

Philips' Bounded Retransmission Protocol, developed for the 4th generation infra-red remote control systems [10], has been the focus of various verification efforts [5–7,9,2,4]. It is a non-trivial protocol to transfer files across a lossy medium as a sequence of chunks. It allows for only a bounded number of retransmissions of chunks and it uses time outs to detect chunk loss. As such, the protocol is parameterised by the retransmission bound and the various time-out values. Correctness entails verification for any file and any number of chunks.

We use this protocol to illustrate the use of abstract interpretation techniques in model checking. Specifically, we verify the protocol for all parameter values, files and number of chunks using a finite-state model checker without inductive arguments. Instead, we rely on a number of *canonical* abstractions of the files, timers etc. to render the parameterised system finite state.

The techniques that we use here are from [1].

2 Protocol description

The core service accepts files, each as a sequence of chunks, and delivers each file as a sequence of pairs; each pair comprises a chunk together with

[1] A page containing figures has been added.

an indication: I_FIRST if the chunk is the first (and not also the last) of a file, I_INCOMPLETE for every following chunk, excepting the last one, which carries an I_OK tag; an error is indicated by an I_NOTOK tag after which no more chunks of the current file will be delivered. Hence, the service guarantees for each file delivery of a prefix (of chunks) only. The file provider receives confirmation after each file transfer: C_OK on success, C_NOTOK if chunks were lost, and C_DONTKNOW if neither success nor failure can be established (see below).

The protocol consists of a sender that transmits chunks across a lossy channel to a receiver that acknowledges each transmission along a second lossy channel to the sender. The two channels are assumed to either loose a message or deliver it correctly. After transmitting a chunk the sender waits `Vresend` seconds for an acknowledgement. If none arrives, the sender times out and resends the chunk. Hence, `Vresend` must exceed the maximal time it takes for a chunk to reach the receiver and the corresponding acknowledgement to arrive back at the sender, assuming no message loss. The sender will attempt to send each chunk at most `MAX` times. If for some chunk even the `MAX`th attempt failed, the rest of the file is skipped (and the file provider receives an C_NOTOK indication). Note that the resend limit can be reached because acknowledgements are being lost. If this happens for the last chunk of a file, then the whole file might have been transmitted successfully; the sender just cannot be sure. It is in this case that a C_DONTKNOW indication is returned.

Because the acknowledgement channel is lossy as well, the protocol must guard against chunk duplication. Accordingly, chunks are 'piggy-backed' with an alternating bit and the receiver discards (but does acknowledge) chunks with the 'wrong' bit. The receiver must also detect the situation that the sender has given up transferring a file. For this, the receiver is prepared to wait for at most `Vgiveup` seconds for the next chunk in a file after it has sent an acknowledgement. Hence, `Vgiveup` must exceed (roughly) `MAX` $*$ `Vresend`, being the maximal time it takes for the sender to give up. In its turn, after the sender has given up it cannot start transmitting a new file before it is sure the receiver knows it has done so; otherwise the first chunk of the new file might be taken as part of the preceding file. The receiver is prepared to wait `Vgiveup` seconds, so the sender must wait at least as long (`Vrecover`) after it has given up (cf. [2]). Note that after the sender has given a C_NOTOK confirmation and sends the first packet of the next list, the receiver should only accept this packet after having indicated an I_NOTOK.

2.1 The model

We implement this protocol by the synchronous discrete-time automata shown in Figure 1 at the end of this article — each automaton representing a process. A step of the system consists of a parallel move of the automata, each of them following an outgoing edge whose condition evaluates to *true* and updating the global state accordingly (it is required that such a move can always be made, possibly leading to the same location — the condition "otherwise" associated with a self-loop on a location abbreviates the conjunction of the negations of the conditions along all other outgoing edges). Communication is through shared variables. Reading a variable yields its value at the beginning of the step, so that the effect of writes only becomes visible in the next step. System steps are alternated with *time steps* in which the value of any timer that is not in its reset state is decreased by 1, or put to reset if it was 0. The initial locations of the automata are indicated by arrows.

The data and acknowledgement channels (datachan, ackchan) between the sender and receiver as well as the communication links with the clients (req, sresult, rresult) are implemented by shared variables. Sending a message is achieved by writing a value into the variable after having checked that it is empty. Receiving a message consists in reading a non-empty value and then writing the value empty in the variable. Thus, such channels can be seen as one-place buffers. All these variables initially contain the value empty. The channel datachan can contain a quadruple representing three headers (in the fields first, last, and altbit) and a packet (in the field cont), or else the value empty. Its lossy behaviour is captured in the procedure SendPacket. Similarly, the acknowledgement channel is modelled by ackchan, that contains either the message ack or empty. The list of packets (= chunks) to be sent is represented by the variable list with operations head and tail as usual and a test one_el that inspects whether list contains precisely one element. Besides list (and the two timers), the following variables are only used by the sender: rn (the counter that keeps track of the number of packet-sending attempts); sfirst (denoting that the packet being sent is the first of a new list); and stoggle (containing the alternating bit to be sent along with the packet). The variable rtoggle (the expected alternating bit in the next packet to be received) is local to the receiver. The timers tresend and trecover are written (Set, Reset) and read (TimeOut) by the sender; tgiveup by the receiver.

The variables sresult and rresult are used to inform respectively the sending and receiving clients, which are not shown here. rresult may be empty, I_NOTOK, or a value with two fields: ind containing the receiver's indication and pkt for the lastly received packet.

3 Verification by Abstraction

A number of problems stand in the way of a finite state verification. First, there are an infinite number of protocol instantiations due to the parametrisation with `MAX`, `Vresend`, `Vgiveup`, and `Vrecover`. Second, every one such protocol still has an infinite state space because packets may have arbitrary contents and lists arbitrary lengths. We will illustrate how abstraction techniques may be used to bring down the state space to a manageable finite size, while properties shown to be true for the abstraction are guaranteed to hold for the original system. We do not give the technical background; the reader is referred to [1] for details and further references. We focus on the following properties: *For any instantiation of the parameters such that* `Vresend` *is at least 4,* `Vgiveup` *at least* $(\mathtt{MAX}-1)\cdot(\mathtt{Vresend}+1)+3$ *and* `Vrecover` *at least*

$\texttt{Vgiveup} - 3$*:*

prefix property *If the sender is transmitting a list ℓ and the first packet of ℓ arrives at least once at the receiver, then the sequence of packets that the receiving client receives (from the last* I_OK/I_NOTOK *to the next) forms a prefix of ℓ.*

timeliness *For every packet that arrives at the receiver, the receiving client gets an indication within time* $\texttt{Vgiveup} + 3$*.*

For the prefix property, it can be shown that it suffices to show the following properties (the case that the input or the output list has length one is excepted and needs to be treated separately). (**1**) ("no swaps or losses":) For any two values e_1 and e_2 on positions i and j resp. in the input list, with $i < j$, one of the following two properties holds for the output list: either (a) e_2 does not occur, or (b) e_1 and e_2 occur on positions i' and j' resp., with $i' < j'$. (**2**) ("no duplication or junk":) For any two values e_1 and e_2 on positions i' and j' resp. in the output list, with $i' < j'$, e_1 and e_2 occur on positions i and j resp. in the input list, with $i < j$. The reason that we reduce the correctness criterion to these order-preservation properties is that there exists a canonical abstraction (see [3]) on sequences that is appropriate for proving such properties. Given values $e_1 \neq e_2$ and a sequence $\bar{s}$ which contains no value more than once, this abstraction abstracts away from all information contained in $\bar{s}$ except the fact which of e_1 and e_2 occur, and, if both occur, in which order. Representing e_1 by $\mathbf{e}$, e_2 by $\mathbf{e}'$, and the order by $\bullet$, we abstract, e.g., the sequence $e_6, e_3, e_2, e_4, e_1, e_9$ to $\mathbf{e}' \bullet \mathbf{e}$, the sequence e_6, e_1 to $\mathbf{e}$, and the sequence e_3, e_6 to ϵ (the empty abstract sequence). This can be formalised by a *concretisation function* $\gamma_{(e_1,e_2)}$ that maps an abstract sequence to the set of all concrete sequences it represents. The idea is now to replace all lists that occur in the BRP by their abstractions, and to "lift" all list operations that take place to the abstract values. It is essential that the operations are lifted in such a way that the set of all behaviours that may thus occur represents a superset of all possible concrete behaviours (*safety*). For example, the abstract operation head, when applied to the list $\mathbf{e}' \bullet \mathbf{e}$, nondeterministically yields a value in $\{\mathbf{e}', \mathbf{ne}\}$, thus reflecting the fact that any concrete list described by $\mathbf{e}' \bullet \mathbf{e}$ may start with e_2 or with an element that is not equal to e_1 nor to e_2. This ensures that any universal[2] property that is true of the abstracted model also holds for the concrete system. Proving the safety of abstract operators can in general not be performed automatically as it involves elimination of quantifiers over infinite (concrete) data domains. However, such proofs are typically readily performed by hand and need to be done only once.

In order to apply the order abstraction to the lists in the BRP, two additional observations are needed. First, it seems that an abstraction as induced by $\gamma_{(e_1,e_2)}$ can only be used to show order preservation for the fixed values e_1 and e_2. However, it is easily seen (by inspecting the safety proofs) that the lifted operations are *independent* of the specific values that $\mathbf{e}$ and $\mathbf{e}'$ represent. More formally, $\gamma_{(\bar{e},\tilde{e})}$ induces the same abstraction for any two values $\bar{e}$ and $\tilde{e}$.

[2] A universal property says something about all executions of a system.

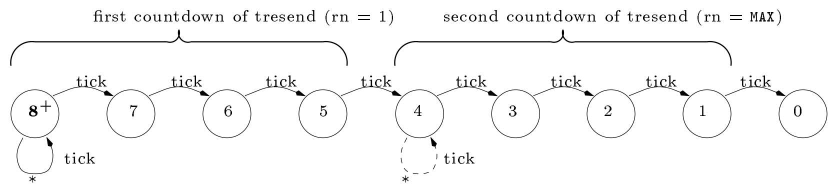

Figure 3

This means that we need to give a single verification only. Second, we have assumed so far that every value occurs at most once in a list that is to be abstracted. In order to overcome this restriction, it suffices to note that the protocol is data independent, so that the elements in lists may be assumed to be unique.

Next, we illustrate how the timers are abstracted (for the time being we assume that MAX is still a concrete number). The protocol only works correctly for certain minimal values of the parameters Vresend, Vgiveup and Vrecover, as explained before. Vresend is bounded from below by the maximum round-trip time, 4, between sender and receiver. Hence, the values of timer tresend cannot be abstracted completely, because that would introduce behaviours (resending a packet too soon) violating the correctness properties we want to check. In order to express the lower bound on Vresend we leave all values below 4 unabstracted[3], while all values greater than or equal to 4 are "collapsed" by representing them by single abstract value, say $\mathbf{4}^+$. An abstract tick (= decrease by 1, time step) action of tresend, when its value equals $\mathbf{4}^+$, now nondeterministically either leaves its value unchanged or changes it to 3; it affects it as usual in other values. The abstract operation Set(tresend, Vresend) has to set tresend to $\mathbf{4}^+$; this is a safe abstraction for any concrete value of Vresend that is ≥ 4.

An abstraction of tgiveup, the receiver's timer, needs to capture the relation $\mathtt{Vgiveup} \geq (\mathtt{MAX}-1)\cdot(\text{Vresend}+1)+3$. The problem is that the value of Vresend is not known — except that it is at least 4. An abstraction of tgiveup that collapses all values $\geq (\mathtt{MAX}-1)\cdot(4+1)+3$ and explicitly distinguishes all lower values is too weak in order to retain the properties of interest that we want to prove. This is illustrated by the following example in which we assume $\mathtt{MAX} = 2$ and where we have replaced, for simplicity, the relation by $\mathtt{Vgiveup} \geq \mathtt{MAX} \cdot \text{Vresend}$ (=8). Suppose that tresend (see Figure 2), after being set for the first transmission attempt, "stutters" 3 times in $\mathbf{4}^+$ before walking down the sequence $3, 2, 1, 0$, then times out, is set to $\mathbf{4}^+$ again (because $\mathtt{MAX} = 2$), stutters for 2 times, and then counts down to 0 after which further transmission attempts are given up. On the other hand, suppose that tgiveup (see Figure3 — ignore the dashed transition for the moment) does not stutter and immediately starts following $7, 6, \ldots, 0$, concluding on time out that the sender must have given up. Then the sender has abandoned transmission after 15 time units while the receiver concluded so within 8 units only. However, the receiver should at least have taken 15 units as well in order for the protocol to always work correctly. The problem is this: if tresend has some value in $\gamma(\mathbf{4}^+)$

[3] Formally, we would have to introduce separate abstract values for each of them.

	Property	Model	# states	Mem usage in % w.r.t. 4
1	Timeliness	files, timers and # tries abstracted	5.64864e+06	107%
2	Prefix	files, timers and # tries abstracted	10.96880e+06	237%
3	Prefix	files, timers and # tries abstracted; lowerbound for `Vgiveup`, and `Vrecover`	2.92721e+06	71%
4	Prefix	concrete model; 2 different files of 5 chunks each; # tries=5	3.63991e+06	100%

Table 1
Verification Results

and tgiveup some value in $\gamma(\mathbf{8}^+)$, how can we still express the fact that tgiveup $\geq$ `MAX`$\cdot$ tresend, without refining the abstract domains (because that would lead us to distinguish all concrete values, which is obviously of no help)? The solution is suggested by the following observation: *Every stutter step that the abstracted tresend takes while its value is* $\mathbf{4}^+$ *must be matched by a stutter step of tgiveup.* Hence, we need to introduce a dependency between these actions. In Figures 2 and 3, this dependency has been indicated by marking with a * the transitions that have to be synchronised. Note that tgiveup must now be allowed to stutter in state 4 as well (the dashed transition). The basic idea is again to introduce a parameter Δ that is equal to the actual number of times that tresend stutters in $\mathbf{4}^+$, i.e. $\Delta = \text{Vresend} - 4$. The concretisation function then depends on Δ, but the induced abstract operations are independent from it—hence allowing a single verification.

Using a similar solution, we can now also abstract the values of rn. Figure 3, which illustrates the case that `MAX` $= 2$ and rn is unabstracted, can be "curled up", giving Figure 4 reflecting the situation where the precise value of rn is unknown. The restriction is that whenever timer tresend is set to $\mathbf{4}^+$ — indicating that rn has not yet reached `MAX`, timer trecover may not leave its cycle. This dependence is indicated by a ** on the corresponding transitions.

The formalisation of these types of *affine* relations between timers in the abstraction framework will have to wait for the full paper.

Experimental results

We have implemented the protocol in Promela, the modelling language of the the SPIN model checker ([8]). Results are indicated in Table 1.

Both the timeliness property (1) and the prefix property (2) can be verified for all files and parameter values. For comparison, (4) list a verification result for a concrete model of the protocol, where the number of tries is set to 5 and the sending client cycles through a sequence of 2 different files each with 5 chunks.

4 In conclusion

In this article we verified a parameterised system—the Bounded Retransmission Protocol—using a finite state model checker. We succeeded in this by an (at least to us) surprising application of abstract interpretation techniques. The key factor in this success is the fact that although the abstracted operations depend on the parameters, this dependence only surfaces in the concretisation function but not in the abstracted operations on datatypes and timers.

The obvious question is how common such parameter independence is in parameterised systems. For this, more experience in system verification using abstraction is needed and it may be possible to characterise classes of parameterised systems that are amenable to this techique.

Acknowledgements

We thank Orna Grumberg for comments on an earlier version of this paper. Dragan Boshnachki and Pedro D'Argenio are acknowledged for sharing with us their experiences in verifying the protocol.

References

[1] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[2] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in LNCS, pages 416–431, Berlin, 1997. Springer. Full version available as CTIT report 97-03, University of Twente, The Netherlands, 1997.

[3] Susanne Graf. Verification of a distributed cache memory by using abstractions. In David L. Dill, editor, *Computer Aided Verification*, number 818 in LNCS, pages 207–219, Berlin, 1994. Springer-Verlag. To appear in *Distributed Computing*.

[4] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification*, number 1254 in LNCS, pages 72–83, Berlin, 1997. Springer.

[5] Jan Friso Groote and Jaco van de Pol. A bounded retransmission protocol for large data packets. In Martin Wirsing and Maurice Nivat, editors, *Algebraic Methodology and Software Technology*, number 1101 in LNCS, pages 536–550, Berlin, 1996. Springer-Verlag.

[6] Klaus Havelund and Natarajan Shankar. Experiments in theorem proving and model checking for protocol verification. In Marie-Claude Glaudel and James Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, number 1051 in LNCS, pages 662–681, Berlin, 1996. Springer.

[7] L. Helmink, M.P.A. Sellink, and F.W. Vaandrager. Proof-checking a data link protocol. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs*, number 806 in LNCS, pages 127–165, Berlin, 1994. Springer-Verlag.

[8] Gerard Holzmann. Spin package. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[9] Radu Mateescu. Formal description and analysis of a bounded retransmission protocol. In Z. Brezočnik and T. Kapus, editors, *Proceedings of COST 247, International Workshop on Applied Formal Methods in System Design (Maribor, Slovenia)*. University of Maribor, Slovenia (technical report), 1996. Also available as technical report no. 2965, Institut National de Recherche en Informatique et en Automatique, 1996.

[10] Infra red remote control system RC6, April 1997. Philips Consumer Electronics B.V.

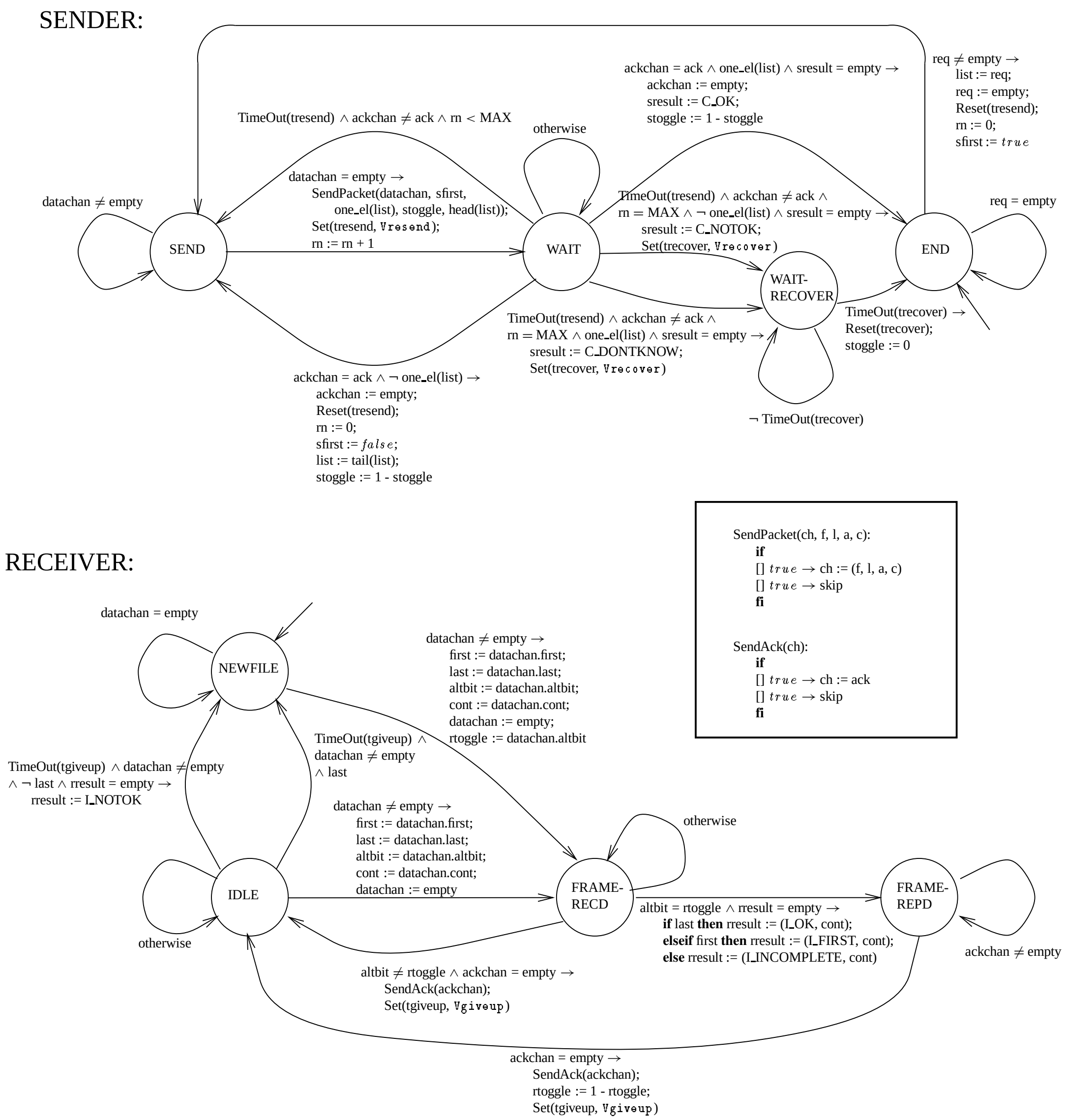

Figure 1

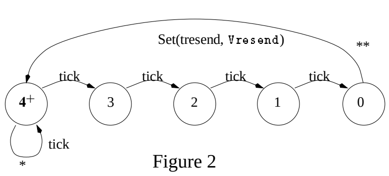

Figure 2

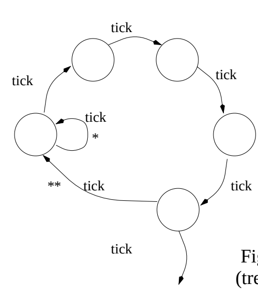

Figure 4
(trecover)