Assertional Verification of a Timer-based Protocol

Gerard Tel

RUU-CS-87-15
September 1987

Rijksuniversiteit Utrecht

Vakgroep Informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

Assertional Verification of a Timer-based Protocol

Gerard Tel

Technical Report RUU-CS-87-15
September 1987

Department of Computer Science
University of Utrecht
P.O. Box 80.012, 3508 TA Utrecht
The Netherlands

Assertional Verification of a Timer Based Protocol

[illegible]

[illegible]
[illegible]

Abstract: We introduce a timer-based protocol skeleton for end-to-end data transport and connection management that includes the well-known Δt-protocol [FW78]. The verification of the skeleton is done using a technique of system-wide invariants. To our knowledge this is the first time this technique is applied to timer-based algorithms. The approach is extended to handle the case of inaccurate timers. Thus the contribution of this paper is not only a rigid correctness proof of a timer-based communication protocol, but also the extension of the proof method of system-wide invariants to a wider class of distributed algorithms. The proof technique adds to a better understanding of how time helps us in designing distributed algorithms.

Key Words: End-to-end protocols, assertional proofs, Time in distributed systems, Connection management

1 Introduction

Assertional correctness proofs have been recognized as an important means for the [illegible] of distributed programs and communication protocols. [illegible] been used to prove the correctness of [illegible] communication protocols, see Knuth [illegible], or Schneider [illegible]. In the protocols described there, we find no notion of timers or time-out conditions. This reflects that these mechanisms have no influence on the partial correctness of the data transmission parts of communication protocols.

However, timers may be of use for the connection management part of the protocol, that is the part that governs the opening and closing of the connection. [illegible] Fletcher and Watson [illegible] the use of timers [illegible]

This work was supported by the Foundation for Computer Science Research (SION) of the Netherlands Organization for Scientific Research (NWO). The author [illegible] by an external observer).

The task of a transport protocol is to let A and B exchange information in a *reliable* way. That is, no information may be lost, duplicated or delivered out of order. Starting with no status information in A or B, a connection must be *opened* when either station has some information to send. When there is no information left to be sent, the connection must be *closed*, that is, status information must be discarded on both sides. Subsequent arrival of messages (from an earlier connection) may not cause duplication, i.e., information may not be accepted twice. This opening and closing (which is normally referred to as connection management) and the subsequent exchange of information (data transfer) must be carried out under the control of a transport protocol.

We consider information traffic from A to B only and, for the time being, assume that timer drift is 0. A duplex communication can be set up by using two simplex connections, whereas the extension to handle timer drift is discussed in section 3. Opening a connection is implicit in the sending or receipt of a packet. At the same time a timer is set. This timer is refreshed when subsequent messages are sent or received. The connection is closed when the timer runs out. An acknowledgement and retransmission mechanism is employed to avoid loss of information. The sending protocol will ensure that each unit of information is sent only during a time interval of length U. Because the lifetime of a packet is bounded by MPL, each unit of information can be in the network only during an interval $U+MPL$. So, more than $R \geq U+MPL$ time after the acceptance of a packet there is no risk of accepting the same packet again, and the receiver can discard status information safely. Only within an interval R after the acceptance of an element B can send an acknowledgement for this element. The sender keeps timers also. No acknowledgements for packets can be received more than $2MPL+R$ after the sending of these packets, so after this time the packets can be reported as (possibly) lost. When no packets have been sent during an $S \geq 2MPL+R$ interval, the sender times out and closes the current connection. The fact that opening and closing a connection costs no extra messages makes timer-based protocols particularly efficient for small bursts of communication.

1.2 Details of the protocol

We describe a protocol skeleton rather than a complete protocol. That is, we give a list of atomic actions, which are allowed to be executed in any arbitrary sequencing. Still, the invariants asserting the correctness can be shown to remain true. It follows that any actual protocol implementation, based on these atomic actions, is partially correct. Thus, in fact a class of protocols is validated. The protocol skeleton is designed so as to capture the essence of the Δ-t protocol due to Fletcher and Watson [FW78].

We call the unit of information an *element*, and it is not important whether an element is a bit, a byte, a file, or whatever. Although A and B actually discard status information between connections, in our skeleton A keeps some. This is not necessary for the correctness of the skeleton, but we need this for our analysis. The variables of A are: an infinite array $M[1], M[2], \cdots$ of elements, an infinite array $Ut[1], Ut[2], \cdots$ of associated timers, integers *Low* and *High* to indicate A's sending window, an integer *Base* used for sequence numbers, and a timer St. It is not essential that A keeps elements in an array: a queue could do as well. The assumption that A keeps elements in an array only serves to simplify the correctness proof. The variables of B are: an expected sequence number *Next*, and a timer Rt. Timers are a special kind of variable. The value of a timer decreases constantly in time, even when it is not assigned to. We assume that the speed at which timers decrease is the same for all timers in the system. Constants are: U, the length of the send interval for packets at A (the value of U is discussed in section 4), $R \geq U + MPL$, the receiver's time-out value, and $S \geq 2MPL + R$, the sender's time-out value. The value of MPL is a constant, depending on the network.

We now discuss the actions A and B can execute. A accepts a next element for transmission and increases its sending window by executing the following operation:

A1: **begin** $Ut[High] := U$;
$M[High] :=$ "new element" ;
$High := High + 1$
end

When A sends a packet, a connection is implicitly opened (if none exists). The St timer is set, and when it reaches the value 0 (and $High = Low$) the connection is closed again. The format of a packet sent by A is $<SoS, SN, M>$, where SoS is a boolean value (Start of Sequence), SN is a sequence number, and M an element. A sets SoS to *true* iff the packet contains the first element in A's sending window, i.e., the packet contains the element with number *Low*. For B this means that the element is to be accepted, even if no open connection exists. A uses consecutive sequence numbers within each connection, but is free to choose new sequence numbers for each new connection. So, the sequence number of a packet containing $M[i]$ is $i + Base$, where *Base* is constant within a connection. The packet must be within the window and have a positive timer:

A2: $\{ Low \leq i < High \wedge Ut[i] > 0 \}$
begin send $<(i = Low), i + Base, M[i]>$;
$St := S$
end

The format of acknowledgements A receives from B is just an expected sequence number. The acknowledgement serves to acknowledge the receipt of all elements with a smaller sequence number within the connection. This is of course the absolute element number plus *Base*. Its receipt triggers:

A3: { Receive <*ESN*> }
begin $Low := \max(Low, ESN - Base)$ **end**

When an element is not acknowledged for a long time, is is reported as possibly lost. It is also possible, that the element is accepted by B, but only the acknowledgement was not received. A cannot distinguish between these two cases [Be76]. The moment an element is reported can be chosen in different ways. In [FW78] all outstanding elements are reported when *St* runs out. We chose to report an element $2MPL + R$ after the end of its transmission interval. Both possibilities result in a correct protocol skeleton.

A4: $\{ Ut[Low] < -2MPL - R \}$
begin report $M[Low]$ as possibly lost ;
$Low := Low + 1$
end

We allow A to choose new sequence numbers in each connection by adding an operation that changes *Base* when no connection exists:

A5: $\{ St < 0 \}$
begin $Base :=$ random($\mathbb{Z}$) **end**

For the receiver B there are only two actions: the receipt of a packet and the sending of an acknowledgement.

B1: **upon** receipt of <*SoS*, *SN*, *M*> **do**
if $(Rt \le 0 \wedge SoS) \vee (Rt > 0 \wedge SN = Next)$ **then**
begin accept M ;
$Next := SN + 1$;
$Rt := R$
end

B2: $\{ Rt > 0 \}$
begin send <*Next*> **end**

The actions A1 through A5, B1, and B2 together form the protocol skeleton. The actions can be executed by A and B (respectively) in any desired order and with any desired frequency. A closes a connection when *St* runs out, B closes a connection when *Rt* runs out. B discards all status information when *Rt* reaches the value 0. In action B1, B does not need the "old" values of variables in case $Rt \le 0$. Even *Rt* can be discarded: if B has no "connection record" of an incoming connection from A, this is interpreted as $Rt \le 0$.

2 Correctness proof of the skeleton

In section 2.1 we give the protocol skeleton in a slightly modified form, which we need for the correctness proof. In section 2.2 we prove that no undetected loss of elements occurs, and in section 2.3 we prove that no duplicates are accepted, and sequencing is done correctly.

2.1 Modified protocol skeleton

In the correctness proof we need assertions involving not only the variables of A and B, but also aspects of the system state that are not observable to A and B. Hence we add these as "auxiliary variables". We keep boolean arrays *accepted*[..] and *error*[..] to indicate what elements have been accepted by B or reported as error by A. To the packets in the network we add a field *RPL*, Remaining Packet Lifetime, to indicate how long the packet can still be in the network. To packets traveling from A to B we also add the absolute number of the element included. The format of packets sent by A is now $<SoS, SN, M, i, RPL>$, where *SoS*, *SN*, and *M* are as before, *i* is the absolute element number of *M*, and *RPL* is as explained. The format of packets sent by B is now $<ESN, RPL>$. The reader should realize that the added variables are "invisible" to A and B, and should not be there at all in a real implementation. We refer to the multiset of packets, traveling from A to B as the *AB-pool* (and vice versa). Initially, the value of *Low, High, Next, Last, St,* and *Rt* is 0, *accepted*[*i*] and *error*[*i*] are *false* for all *i*, and the pools contain no messages. The actions of the skeleton are modified to:

A1: **begin** $Ut[High] := U$;
$M[High] :=$ "new element" ;
$High := High + 1$
end

A2: $\{ Low \le i < High \wedge Ut[i] > 0 \}$
begin send $<(i = Low), i + Base, M[i], i, MPL>$;
$St := S$
end

A3: { Receive $<ESN, RPL>$ }
begin $Low := \max(Low, ESN - Base)$ **end**

A4: $\{ Ut[Low] < -2MPL - R \}$
begin $error[Low] := true$;
$Low := Low + 1$
end

```
A5:  { St < 0 }
     begin Base := random(ℤ) end

B1:  upon receipt of <SoS, SN, M, i, RPL> do
         if (Rt ≤ 0 ∧ SoS) ∨ (Rt > 0 ∧ SN = Next) then
             begin accepted[i] := true ; Last := i ;
                 Next := SN + 1 ; Rt := R
             end

B2:  { Rt > 0 }
     begin send <Next, MPL> end
```

To model the behavior of time we introduce a new atomic action. This action represents what happens if no other action takes place during a certain time δ. It decreases all timers and *RPL*'s of packets by δ, and discards packets whose *RPL* becomes zero or less. Although this action involves variable changes all over the system, it is realistic to consider it atomic.

```
TIME:
     begin δ := random(ℝ⁺) ;
         forall i do Ut[i] := Ut[i] − δ ;
         St := St − δ ;
         Rt := Rt − δ ;
         forall <..,RPL> in pools do
             begin RPL := RPL − δ ;
                 if RPL ≤ 0 then discard message
             end
     end
```

We want to prove that the protocol skeleton is resilient against loss, duplication, and resequencing of packets. We handle resequencing by modeling pools as sets rather than as queues. The loss or the duplication of a packet in the network can also be modeled as actions.

```
LOSS:
     { M ∈ pool }
     begin discard M from pool end

DUP: { M ∈ pool }
     begin insert M to pool end
```

Here for *pool* one can read the AB- as well as the BA-pool, and M is any message in this pool. The formulation of all of our assertions is such that these actions preserve them:

Observation 1: An invariant of the form "*timer* 1 $\geq$ *timer* 2 + *constant*" is preserved by TIME.

Observation 2: An invariant of the form "For all M in pool: $P(M)$" is preserved by LOSS (or DUP) if removal (duplication) of M preserves $P(M)$.

Observation 1 holds because TIME decreases all timers by the same amount, observation 2 holds because the conclusion of the invariant remains true. Most of our invariants involving timers or pools are of these forms. Note that a programmer has control over the execution of actions A1 to A5, B1, and B2, but not over TIME, LOSS, or DUP.

2.2 Loss of elements

In this section we prove a series of invariants, preserved by all actions A1 through A5, B1, B2, TIME, LOSS and DUP. The last one will be $\forall i < Low: ok(i)$, stating that no undetected loss occurs.

Lemma 2.1: $St \leq S$, $Rt \leq R$, $\forall i\ Ut[i] \leq U$, $\forall$<.., RPL> in pools: $0 < RPL \leq MPL$.

Proof: Initially all timers are 0 and there are no packets in the pools. St is assigned to only in A2 and TIME, A2 sets St to S and TIME decreases St. So $St \leq S$ invariantly holds. $Rt \leq R$ and $Ut[i] \leq U$ are proven similarly. Packets are sent with $RPL = MPL$, and TIME decreases RPL but discards packets when it reaches 0. □

Lemma 2.2: For all <.., RPL> in the AB-pool, $St \geq RPL + MPL + R$.

Proof: Initially the pool is empty so the lemma holds trivially. Upon sending <.., MPL>, St is set to $S \geq 2MPL + R = MPL + MPL + R$. The increase of St leaves the relation invariant for already existing packets. TIME preserves this invariant by observation 1. LOSS and DUP preserve this invariant by observation 2. Other actions do not involve the variables involved in the lemma. □

Lemma 2.3: If $Rt > 0$ then $St > Rt + MPL$.

Proof: Initially $St = Rt = 0$ so the relation holds. A2 increases St so A2 preserves this inequality. Upon receipt of <.., RPL>, B sets Rt to R (action B1). By 2.1, $RPL > 0$, and by lemma 2.2, $St \geq RPL + MPL + R$. So after action B1 $St > Rt + MPL$ holds. TIME preserves this invariant by observation 1. □

Lemma 2.4: For all <ESN, RPL> in the BA-pool, $St > RPL$.

Proof: Initially the pool is empty so the statement holds trivially. A2 increases St, so A2 preserves this assertion. B sends an acknowledgement <ESN, RPL> with $RPL = MPL$ only when $Rt > 0$ (action B2) and, by the previous lemma, we then have $St > MPL$. TIME preserves this invariant by observation 1. LOSS and DUP preserve this invariant by observation 2. □

Lemma 2.5: For all <SoS, SN, M, i, RPL> in the AB-pool, $SN = i + Base$.

Proof: A packet satisfies this relation when it is sent (action A2). For a packet in transit, i and SN are never changed. When a packet is in transit, $St > 0$ by lemma 2.2, and hence a change of *Base* (action A5) does not occur. LOSS and DUP preserve this relation by observation 2. □

We want to prove, that every element is received or reported as possibly lost. Therefore, define the predicate $ok(i) :\Leftrightarrow (error[i] \vee accepted[i])$.

Lemma 2.6:

(i) $\forall i < Low: ok(i)$,

(ii) For all <$true, SN, M, j, RPL$> in the AB-pool: $\forall i < SN - Base: ok(i)$,

(iii) If $Rt > 0$ then $\forall i < Next - Base: ok(i)$, and

(iv) For all <ESN, RPL> in the BA-pool: $\forall i < ESN - Base: ok(i)$.

Proof: The proof goes by simultaneous induction. Observe that nowhere (after initialization) $accepted[i]$ or $error[i]$ is set to *false* and hence, once $ok(i)$ is *true* for some i, it remains *true* forever. This assertion holds initially, for (i) $Low = 0$, (ii) the AB-pool is empty, (iii) $Rt = 0$, and (iv) the BA-pool is empty.

(i) Because $ok(i)$ is stable, we only have to consider actions that increase *Low*. A3 and A4 do so. In A4 $error[Low]$ is set to maintain the invariant. If *Low* is increased to $ESN - Base$ in A3, we have $\forall i < Low: ok(i)$ by (iv).

(ii) A packet is sent with $SoS = true$ only if $Low = j$ and with $SN = Base + j$, hence (use i), $\forall i < SN - Base: ok(i)$. *Base* is not changed while a message is under way. LOSS and DUP preserve this relation by observation 2.

(iii) If $Rt \le 0$, action B1 can set Rt to R and *Next* to $SN + 1$ upon receipt of <$true, SN, M, j, RPL$>. From (ii) $\forall i < SN - Base: ok(i)$, (lemma 2.5) $j = SN - Base$, and the fact that $accepted[j]$ is set to *true* it follows that now $\forall i < Next - Base: ok(i)$.
If $Rt > 0$, action B1 can set *Next* to $SN + 1$ upon receipt of <$SoS, Next, M, j, RPL$>. From (iii) $\forall i < Next - Base: ok(i)$, (lemma 2.5) $j = Next - Base$, and the fact that $accepted[j]$ is set to *true* it follows that now $\forall i < Next - Base: ok(i)$.

(iv) If <$Next, MPL$> is sent (in action B2) we have $\forall i < Next - Base: ok(i)$ by (iii). *Base* is not changed while a message is in the BA-pool, because $St > 0$ (lemma 2.5). LOSS and DUP preserve this relation by observation 2. □

We now make the assumption that action A4 is always executed as soon as it is enabled. The main result of this section is:

Theorem 2.1: No element is lost undetectedly.

Proof: If, $U+2MPL+R$ after the delivery of element i to A (action A1), $Low > i$, the element is received by B or already reported by lemma 2.6i. If not, the element will now be timed out (action A4) and reported. □

2.3 Sequencing and duplicates

In this section we will prove that elements are always accepted in correct order, that is, with strictly increasing element numbers. Thus, no element is accepted twice or followed by an element with lower sequence number.

Lemma 2.7: For all $<SoS,EN,M,i,RPL>$ in the AB-pool: $Ut[i] > RPL - MPL$.

Proof: The packet is sent with $RPL = MPL$ and $Ut[i] > 0$, hence $Ut[i] > RPL - MPL$ holds. TIME preserves this relation by observation 1. LOSS and DUP preserve this relation by observation 2. □

Lemma 2.8: $accepted[i] \Rightarrow Rt \geq Ut[i]+MPL$.

Proof: An increase of Rt (in action B1) leaves this relation invariant for earlier accepted elements. For the newly accepted element i we have $Ut[i] \leq U$, Rt is set to R, and $R \geq U+MPL$, hence $Rt \geq Ut[i]+MPL$. TIME preserves this relation by observation 1. □

Note in particular that $Rt \geq Ut[Last]+MPL$.

Lemma 2.9: For all $i_1 \leq i_2 < High : Ut[i_1] \leq Ut[i_2]$.

Proof: Initially $High = 0$ so this holds. A1 increments $High$ from, say, h to $h+1$. $Ut[h]$ is set to U, and for all $i_1 \leq h$ we now have $Ut[i_1] \leq Ut[h]$ by lemma 2.1. For smaller i_2 the increase of $High$ of course preserves the relation. TIME preserves this relation by observation 1. □

Lemma 2.10: If $accepted[i_2]$ and $<SoS,SN,M,i_1,RPL>$ is in the AB-pool for some $i_1 \leq i_2$, then $Rt > 0$.

Proof: $accepted[i_2]$ implies $Rt \geq Ut[i_2]+MPL$ by 2.8. $i_1 \leq i_2$ implies $Ut[i_1] \leq Ut[i_2]$ by 2.9. $<SoS,SN,M,i_1,RPL>$ in the AB-pool implies $Ut[i_1] > -MPL$ by 2.7 and 2.1. $Rt > 0$ follows. □

Lemma 2.11: $Rt > 0 \Rightarrow Last = Next - Base - 1$.

Proof: Each time Rt is set (upon receipt of a packet $<SoS,SN,M,i,RPL>$), $Last$ is set to i and $Next$ to $SN+1$, and $SN = i+Base$ (lemma 2.5), so $Last = Next - Base - 1$ follows. $Rt > 0$ implies $St > 0$ (lemma 2.3), so A5 is disabled. □

We are now ready to prove the following important result:

Theorem 2.2: B accepts elements with strictly increasing element numbers.

Proof: Assume <*SoS*, *SN*, *M*, *i*, *RPL*> arrives and $i \le Last$. Because *accepted*[*Last*] is *true*, from 2.10 follows $Rt > 0$. By 2.5, $SN = i + Base$. By 2.11, $Next = Last + Base + 1$, and so, for $i \le Last$, $Next > SN$. Hence B does not accept M. □

Theorems 2.1 and 2.2 together state that the protocol skeleton guarantees reliable information exchange and connection management.

3 Timer drift

Until now we have assumed that all timers in the network run at equal speed, but in practice this will not be the case. Mechanical or electronic clocks tend to suffer a "drift". This drift is very small: quartz clocks that one can buy for a dollar everywhere show an inaccuracy of about one part in a million. We will now see how the program is modified to handle any clock drift, assuming that the drift is ρ-*bounded*. That is, in real time δ a clock is decreased by an amount δ′, where $\frac{\delta}{1+\rho} \le \delta' \le \delta \times (1+\rho)$. The TIME action can easily be modified to model this changed behavior of real time. In the following formulation we assumed that all timers in computer A run at the same speed. In practice, these timers are not implemented by using a large number of physically independent clocks, but by one hardware clock and additional software [Ta81, p.157].

```
TIME-ρ:
      begin δ := random(ℝ⁺) ;
            δ′ := .. ; (* δ/(1+ρ) ≤ δ′ ≤ δ×(1+ρ) *)
            forall i do Ut[i] := Ut[i] − δ′ ;
            St := St − δ′ ;
            δ″ := .. ; (* δ/(1+ρ) ≤ δ″ ≤ δ×(1+ρ) *)
            Rt := Rt − δ″ ;
            forall <..,RPL> in pools do
                  begin RPL := RPL − δ ;
                        if RPL ≤ 0 then discard message
                  end
      end
```

Of course, this is not the only possible way to modify the TIME action. It is possible to assume that timers within A drift independently, and model TIME-ρ accordingly. If the network uses time stamps and clocks for discarding messages after *MPL*, the network clocks may suffer drift also. One can take this drift into account in TIME-ρ but, on the other hand, it is easily seen that now $MPL' = (1+\rho)MPL$ is an exact bound on the (real time) life-time of a packet. It is possible to model a different drift in A and B, etc.

The protocol skeleton remains unchanged, except that the constants have a different value. Take $R \geq (1+\rho)((1+\rho)U + (1+\rho)^2 MPL)$, and $S \geq (1+\rho)(2MPL + (1+\rho)R)$. We will now formulate weaker invariants, and show that these weaker invariants are maintained by the modified actions. The modification of the correctness proof is done in an almost mechanical manner. Recall observation 1. We now consider invariants of the form $timer1 \geq (1+\rho)^2 timer2 + constant'$. Because in TIME-ρ *timer*1 is decreased by **at least** $\frac{\delta}{(1+\rho)}$, and *timer*2 by **at most** $(1+\rho)\delta$, TIME-ρ preserves this new invariant. We use the fact that, for $t1$, $t2$, d, $d1$, $d2$, r, and c in $\mathbb{R}$, $d > 0$, $r > 1$, from $t1 \geq r^2 t2 + c$, $\frac{d}{r} \leq d1 \leq d\,r$, $\frac{d}{r} \leq d2 \leq d\,r$, follows $(t1 - d1) \geq r^2(t2 - d2) + c$.

Observation 3: TIME-ρ preserves invariants of the form $timer1 \geq (1+\rho)^2 timer2 + constant'$.

The *constant* in the invariant (and thus, the constants in the protocol skeleton) must be changed so that the new invariant is also maintained by the actions that set the timers. Because in TIME-ρ the *RPL*-"timers" run accurately, one factor (1+ρ) suffices if *timer*1 or *timer*2 is the *RPL*-field of some packet.

3.1 Loss of elements

All lemmas in this section have their counterpart in section 2. By taking $\rho = 0$, one finds the simple version of all constants, lemmas, and actions.

Lemma 3.1: $St \leq S$, $Rt \leq R$, $\forall i\ Ut[i] \leq U$, $\forall$<..,*RPL*> in pool: $0 < RPL \leq MPL$.

Proof: As for lemma 2.1. □

Lemma 3.2: For all <..,*RPL*> in the AB-pool, $St \geq (1+\rho)(RPL + MPL + (1+\rho)R)$.

Proof: Initially the pool is empty so the lemma holds trivially. Upon sending <..,*MPL*>, *St* is set to $S \geq (1+\rho)(2MPL + (1+\rho)R) = (1+\rho)(RPL + MPL + (1+\rho)R)$. The increase of *St* leaves the relation invariant for already existing packets. TIME-ρ preserves this relation by observation 3. LOSS and DUP preserve this relation by observation 2. Other actions do not involve the variables involved in the lemma. □

Lemma 3.3: If $Rt > 0$ then $St > (1+\rho)((1+\rho)Rt + MPL)$.

Proof: Initially $St = Rt = 0$ so the relation holds. A2 increases St so A2 preserves this inequality. Upon receipt of <..,*RPL*>, B sets Rt to R (action B1). By 3.1, $RPL > 0$, and by lemma 3.2, $St \geq (1+\rho)(RPL + MPL + (1+\rho)R)$. So after action B1 $St > (1+\rho)((1+\rho)Rt + MPL)$ holds. TIME-ρ preserves this relation by observation 3. □

Lemma 3.4: For all <*ESN*, *RPL*> in the BA-pool, $St > (1+\rho)RPL$.

Proof: Initially the pool is empty so the statement holds trivially. A2 increases St, so A2 preserves this assertion. B sends an acknowledgement <*ESN*, *RPL*> with $RPL = MPL$ only when $Rt > 0$ (action B2) and, by the previous lemma, we then have $St > (1+\rho)MPL$. TIME-ρ preserves this relation by observation 3. LOSS and DUP preserve this relation by observation 2. □

Lemma 3.5: For all <*SoS*, *SN*, *M*, *i*, *RPL*> in the AB-pool, $SN = i + Base$.

Proof: As for lemma 2.5. □

Lemma 3.6:

(i) $\forall i < Low: ok(i)$,

(ii) For all <*true*, *SN*, *M*, *j*, *RPL*> in the AB-pool: $\forall i < SN - Base: ok(i)$,

(iii) If $Rt > 0$ then $\forall i < Next - Base: ok(i)$, and

(iv) For all <*ESN*, *RPL*> in the BA-pool: $\forall i < ESN - Base: ok(i)$.

Proof: As for lemma 2.6. □

Theorem 3.1: No element is lost undetectedly.

Proof: As for theorem 2.1. □

3.2 Sequencing and duplicates

This section corresponds to section 2.3.

Lemma 3.7: For all <*SoS*, *EN*, *M*, *i*, *RPL*> in the AB-pool: $Ut[i] > (1+\rho)(RPL - MPL)$.

Proof: The packet is sent with $RPL = MPL$ and $Ut[i] > 0$, hence $Ut[i] > (1+\rho)(RPL - MPL)$ holds. TIME-ρ preserves this relation by observation 3. LOSS and DUP preserve this relation by observation 2. □

Lemma 3.8: $accepted[i] \Rightarrow Rt \geq (1+\rho)((1+\rho)Ut[i] + (1+\rho)^2 MPL)$.

Proof: An increase of Rt (in action B1) leaves this relation invariant for earlier accepted elements. For the newly accepted element i we have $Ut[i] \leq U$, Rt is set to R, and $R = (1+\rho)((1+\rho)U + (1+\rho)^2 MPL)$, hence $Rt \geq (1+\rho)((1+\rho)Ut[i] + (1+\rho)^2 MPL)$. TIME-ρ preserves this relation by observation 3. □

Note in particular that $Rt \geq (1+\rho)((1+\rho)Ut[Last] + (1+\rho)^2 MPL)$.

Lemma 3.9: For all $i_1 \le i_2 < High: Ut[i_1] \le Ut[i_2]$.

Proof: As for lemma 2.9. □

Lemma 3.10: If $accepted[i_2]$ and <SoS, SN, M, i_1, RPL> is in the AB-pool for some $i_1 \le i_2$, then $Rt > 0$.

Proof: $accepted[i_2]$ implies $Rt \ge (1+\rho)((1+\rho)Ut[i_2] + (1+\rho)^2 MPL)$ by 3.8. $i_1 \le i_2$ implies $Ut[i_1] \le Ut[i_2]$ by 3.9. <SoS, SN, M, i_1, RPL> in the AB-pool implies $Ut[i_1] > (1+\rho)(-MPL)$ by 3.7 and 3.1. $Rt > 0$ follows. □

Lemma 3.11: $Rt > 0 \Rightarrow Last = Next - Base - 1$.

Proof: As for lemma 2.11. □

Theorem 3.2: B accepts elements with strictly increasing element numbers.

Proof: As for theorem 2.2. □

4 Extensions

After having proven the correctness of the protocol skeleton in section 2, we will now discuss some remaining issues and extensions. See also Watson [Wa81].

4.1 The choice of U

The choice of the parameter U has considerable effect on the performance of the protocol. U must be large enough to allow for a sufficient number of retransmissions, so that an element is received by B with probability nearly 1. If *ARD* (Average Round-trip Delay) is the average time it takes for a packet to be acknowledged, and k is the number of times one wants to try retransmission, then $U = k \times ARD$ is a good choice. This choice is made in [FW78]. If U is large, R and S must be large also. Hence, stations must keep state information longer and, if an element is lost, this takes longer to be detected.

4.2 Multi-element packets

In the skeleton given in section 1 a packet contains one element only. Efficient protocols pack more elements in one packet to decrease overhead. The aim of this section is to show that this more efficient transmission of elements is possible within the given framework, without modification of the correctness proofs.

Within the restrictions stated explicitly in section 1 (and in the preconditions of the actions), any scheduling of atomic actions is "safe", i.e., guarantees correct transport of data.

Therefore it is possible to define larger actions for A and B, consisting of one or more old actions and (possibly) some control overhead. Any scheduling of the larger actions is also a possible scheduling of the old actions, and hence preserves correctness. For example, A can execute a series of A2 actions:

A2': $\{ Low \le f \le f + L - 1 < High,\ Ut[f] > 0 \}$
begin for $i := f$ **to** $f + L - 1$ **do**
execute A2 "with i"
end

The result of this action is a burst of L packets $<(f = Low), SN, M[f]>$, $<false, SN+1, M[f+1]>$, ..., $<false, SN+L-1, M[f+L-1]>$. We can introduce a single packet

$$<(f = Low), SN, L, (M[f], .., M[f+L-1])>$$

as an abbreviation for this burst. So, the result of A2' is the sending of this packet. Upon receipt of this packet B simulates the receipt of the single element packets one by one:

B1': **upon** receipt of $<SoS, SN, L, (M_0, .., M_{L-1})>$
begin for $i := 0$ **to** $L - 1$ **do**
execute B1 "with $<(SoS \wedge i = 0), SN+i, M_i>$"
end

It is left to the reader to verify that these new actions are equivalent to

A2': $\{ f \ge Low, f + L - 1 < High, Ut[f] > 0 \}$
begin send $<(f = Low, f + Base, L, (M[f], .., M[f+L-1])>$;
$St := S$
end

B1': { Receive $<SoS, SN, L, (M_0, .., M_{L-1})>$ }
if $Rt \le 0$
then if SoS **then** (* Open a connection *)
begin accept M_0 to M_{L-1} ;
$Next := SN + L$; $Rt := R$ **end**
else discard message
else if $Next \in SN..SN+L-1$ **then**
begin accept $M_{Next-SN}$ to M_{L-1} ;
$Next := SN + L$; $Rt := R$ **end**

The original A2 and B1 can be replaced by A2' and B1' to implement multiple element packets, without further modification of the correctness proof.

4.3 Duplex communication

Until now we assumed that only A had elements for B, but in most applications the reverse will be the case also. We solve this by establishing two connections of the kind we discussed, where in the second connection the roles of A and B are interchanged. Acknowledgements can be piggybacked upon data packets to reduce overhead. Packets occur in one format only: $<ESN; SoS, SN, L, (M,..)>$. If the packet contains data only, and is not to be interpreted as an acknowledgement, ESN can take some non sense value like -1, or an extra boolean field can indicate this (like in [FW78]). If the packet contains no data, and is to be interpreted as an acknowledgement only, L is set to 0, and eventually SoS, L, and M are left out.

4.4 Storage of out-of-sequence packets

In the protocol skeleton we gave an early arriving packet will be rejected. It is possible for B to store these packets temporarily, and accept the elements later when packets with lower sequence number have arrived, and the stored packets will fit in the receiving window. Packets must be stored for at most R, and it can be shown that their sequence number is still correct (i.e., a change of $Base$ did not occur). We do not give details here, but see [FW78].

4.5 Bounded sequence numbers

In the version given, sequence numbers can grow to infinity. In most cases it will be desirable to use packets with control fields of fixed size, so one wants to use bounded sequence numbers.

A simple way to do this (without modification of the skeleton) is the following. A always chooses $Base$ such that the first sequence number in a connection is 1. When the highest sequence number is reached, A stops sending, so that A and B will time out and close the connection. After the time-out, A starts a new connection and transmits the remaining elements, with sequence numbers starting at 1. This solution has the obvious disadvantage that every now and then communication has to be interrupted for a restart of sequence numbers.

If the creation rate of elements is bounded one can use cyclic sequence numbers within one connection. Assume a packet is created only if the P^{th} previous packet is at least $U + MPL + R$ old (no timer drift here):

A1': { $Ut[High - P] < -R - 2MPL$ }
begin $Ut[High] := U$;
$M[High] :=$ "new element" ;
$High := High + 1$
end

We will argue that it now suffices to transmit sequence numbers modulo $2P$. Suppose B receives a packet $<SoS, SN, M, i, RPL>$, and $Rt > 0$ (so that B really looks at the sequence number SN).

Lemma 4.1: $Next - P \leq SN < Next + P$.

Proof: From action A1' above it follows that $Ut[j] + U + 2MPL + R < Ut[j+P]$ for $j + P < High$. Element $Last$ is accepted, so $Ut[Last] < U$, and $Ut[Last-P] < -MPL$. However, $Ut[i] \geq -MPL$ (lemma 2.7), so $i > Last - P$ (lemma 2.9).
$i \geq Last + P$ implies $Last + P < High$ and hence $Ut[Last+P] > Ut[Last] + U + MPL + R$. From $Rt > 0$ follows $Ut[Last] > -MPL - R$, hence $Ut[Last+P] > U$, a contradiction. It follows that $Last - P < i < Last + P$. Thus, use lemmas 2.11 and 2.5, $Next - P \leq SN < Next + P$. □

Lemma 4.2: For all $<ESN, RPL>$ in the BA-pool, $High - P < ESN - Base \leq High$.

Proof: For all $<SoS, SN, M, i, RPL>$ in the BA-pool we have $Ut[i] > RPL - MPL > -MPL$ (Lemma 2.7). So, $Rt > 0$ implies $Ut[Last] > -MPL + Rt - R > -MPL - R$ or, equivalently (lemma 2.11), $Ut[Next - Base - 1] > -MPL - R$. Thus, if $<ESN, MLP>$ is sent, $ESN = Next$ and hence $Ut[ESN - Base - 1] > (RPL - MPL) - MPL - R$. This relation is preserved by TIME, and $Ut[ESN - Base - 1] > -2MPL - R$ follows. $Ut[High - 1] < U$, so $Ut[High - P - 1] < -2MPL - R$, and $ESN - Base > High - P$ follows. $ESN - Base$ must be $\leq High$ because B can not acknowledge unsent packets (use lemma 2.6). □

Theorem 4.1: It suffices to transmit sequence numbers modulo $2P$.

Proof: B1 uses the value SN only when $Rt > 0$. In this case $SN = Next$ is equivalent to $SN \equiv Next \pmod{2P}$ by lemma 4.1. Acknowledgements modulo $2P$ (in fact, even modulo P) are unambiguous by lemma 4.2. □

With a changed bound on element creation rate the lemma and theorem hold in the timer drift case also.

5 Conclusions, comments

In this paper we have proven the correctness of a skeleton for transport protocols. All proofs are formalizable. A protocol skeleton can be refined to a complete protocol, allowing the programmer to tune the protocol to his/her needs. So, in fact a large class of protocols is validated in this paper. The Δ-t protocol of Fletcher and Watson [FW78] belongs to this class. Proving its correctness now reduces to showing that it is a refinement of our protocol skeleton.

Elements for which A does not receive an acknowledgement are reported as possibly lost. It is possible, however, that B has accepted these elements. If A chooses to time out and send the elements again in a new connection, this may result in a duplicate accepted by B. If A chooses not to send the elements again, this may result in a loss of elements. Therefore these elements must be reported to the higher level protocol. It is impossible to solve this dilemma in a protocol that guarantees that connections (in which finitely many elements are transmitted) are closed in finite time. In the protocols of [Sc87] connections may have to remain open forever.

This work has again demonstrated the usefulness of assertional proofs. It is shown that the method is useful not only for data transfer protocols [Kr78], [SvL85], and other asynchronous distributed algorithms [La82], but also for (definitely more complex) timer-based algorithms. Other well-known methods for protocol verification (Finite State Machines, Petri Net Models, see [Ta81]) seem to fail at this point. Currently it is investigated how assertional proofs can be given for fault-tolerant algorithms. We believe that assertional proof methods can be used in combination with modular design techniques for distributed programs.

Assertional proofs can sometimes be lengthy: for each invariant I and each action A one must show that A does not violate I. However, the resulting proof consists of many independent, small proofs and is thus highly modular. Many of the small proofs are rather trivial, for example if I and A have no variables in common. In these cases the proofs were left out. For some actions we could give "classes" of formulas that are not violated by these actions, see the end of sections 2.1 and 3.1.

The way we modeled time in our proof reveals clearly the importance of the use of clocks in distributed programming. An important characteristic of distributed systems is that atomic actions may not involve variables of different processes. This *lack of global control* is one of the fundamental difficulties in distributed programming. We see, however, that when timers are used, it is realistic to consider atomic actions involving variables of different processes. Thus, **the use of timers gives us a sense of global control in distributed algorithms.**

Acknowledgement: I like to thank Jan van Leeuwen for suggesting this research topic to me.

6 References

[Be76] Belsnes, D., *Single-Message Communication,* IEEE Trans. Communications COM-24 (1976) 190-194.

[FW78] Fletcher, J.G., and R.W. Watson, *Mechanisms for a Reliable Timer-based Protocol,* Computer Networks 2 (1978) 271-290.

[Kr78] Krogdahl, S., *Verification of a Class of Link-level Protocols,* BIT 21 (1978) 436-488.

[La82] Lamport, L., *An Assertional Correctness Proof of a Distributed Algorithm,* Science of Computer Programming 2 (1982) 175-206.

[Sc87] Schoone, A.A., *Verification of Connection Management Protocols,* Techn. Rep. RUU-CS-87-14, Dept. of Computer Science, University of Utrecht, Utrecht, 1987. An abstract appears in: J. van Leeuwen (ed), Proceedings 2nd International Workshop on Distributed Algorithms, Springer Verlag, 1987.

[SvL85] Schoone, A.A., and J. van Leeuwen, *Verification of Balanced Link-level Protocols,* Techn. Rep. RUU-CS-85-12, Dept. of Computer Science, University of Utrecht, Utrecht, 1985.

[Ta81] Tanenbaum, A., *Computer Networks,* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Wa81] Watson, R.W., *Timer-based Mechanisms in Reliable Transport Protocol Connection Management,* Computer Networks 5 (1981) 47-56.