Modelling soft-state protocols with SDL

X. Fu and D. Hogrefe

Abstract: The notion of soft state has been introduced in packet-switched networks to achieve particular services for end-to-end communications, such as quality-of-service provisioning and configuration of stateful packet filters. Protocols built upon soft state principles were believed to be simple, however in practice they are far more complex. An important issue with such protocols is to ensure their operations to be error-free and deadlock-free. In the paper the use of formal techniques is proposed, specifically, Specification and Description Language (SDL) and Message Sequence Charts (MSCs), for modelling, analysis and validation of soft-state protocols. Based on a general state management system that identifies their most representative behaviour, an extensive study on modelling and validating soft-state protocols with SDL/MSCs is presented, and it is shown that design flaws and ambiguity introduced in informally specified, textual protocols can be avoided if a protocol is formally modelled.

1 Introduction

In communication networks, there is a need to maintain certain information ('state') in network nodes associated with endpoint-generated sessions or calls. For example, ATM switches maintain information about virtual circuits such as bandwidth allocation. As state generally reflects some requirements of an end-to-end session/call to the traversed nodes, it needs to be maintained properly, especially when network 'conditions' change (e.g., some link or node fails, or the traversing route changes).

The state maintained by the network can be categorised as either hard state or soft state. Hard state is installed in nodes upon receipt of a setup message and is removed only upon receipt of an explicit tear-down message. It is vital for the state initiator to know when the state has been installed or removed, and ensure that installation and removal are performed only once. Furthermore, since hard state remains installed unless explicitly removed, there must be a mechanism to remove an orphan state that is left after the state initiator has crashed or departed without removing the state. In contrast, 'soft state' refers to a certain non-permanent control state in network nodes that expires unless refreshed. Since a soft state eventually expires, this approach does not require explicit removal or a mechanism for removing an orphan state.

Conventionally, it was believed that state information would only be needed for end systems with packet-switching networks, such as the Internet, following the fundamental end-to-end principle [1]:

'An end-to-end protocol design should not rely on the maintenance of state (i.e. information about the state of the end-to-end communication) inside the network. Such state should be maintained only in the endpoints, in such a way that the state can only be destroyed when the endpoint itself breaks.'

However, the idea that 'end-to-end communications should not have state inside the network' has hardly been realised in real networks. For example, most routing protocols, ATM signalling protocol [2] and early Internet signalling protocol (ST-II) [3] use hard state in the network. In addition to hard state, soft state has been introduced into the Internet, too. Once it was applied to the Resource Reservation Protocol (RSVP) [4, 5], which allows QoS resource reservation to establish state in the network nodes along the path for end-to-end communications in IP networks, the soft state paradigm has been adopted by many other protocols. These include the Real-Time Control Protocol (RTCP) [6], Protocol Independent Multicast (PIM) [7], the Session Initiation Protocol (SIP) [8] and the Cross-Application Signalling Protocol (CASP) [9–11]. By the use of state – either hard state or soft state – inside the network, protocols can provide certain enhanced services for end-to-end communications. Note that both hard state and soft state can be installed either in intermediate nodes or end hosts only, or both. Owing to the nature of state, unlike other types of protocols, state management protocols often require extremely complex and powerful mechanisms to ensure that the state is perfectly synchronised and up-to-date with the session it is related to (otherwise problems may arise). With the informal, text-based IETF specifications, operations of these protocols tend to be error-prone. For example, a report [12] showed numerous problems or unexpected behaviours in the specification and implementation of TCP [13], the dominating end-to-end transport protocol that uses hard state in end hosts. With an ever-increasing number of soft-state protocols and the increase in their complexity, unfortunately the risk of design and implementation errors for soft-state protocols increases. Another example is the 'auto-refresh' loop in RSVP, possibly keeping a state alive forever [5]. In general, it is vital to ensure the correctness of state management operations in protocol specifications.

The methods for studying state management operations and correcting possible flaws can be classified into two basic groups. The empirical approach, which is also the most

IEE Proceedings online no. 20050071
doi:10.1049/ip-com:20050071
Paper first received 10th February and in final revised form 27th June 2005

The authors are with Telematics Group, University of Göttingen, Lotzestr. 16-18, 37083 Göttingen, Germany
E-mail: fu@cs.uni-goettingen.de

natural way, is to study protocol behaviours with an actual implementation (or a prototype). Another possibility involves a model-based approach in which protocol behaviours may be studied using a model of the protocol, e.g. as shown in [14]. The empirical approach is only effective for examining standard, ordinary behaviours of a protocol, whereas model-based approaches, based on simulation or analytical models, can be more effective in determining possible errors in the design. Applying the model-based approach to state management systems may reduce excessive problems (and costs!) in standards and implementations, unlike the empirical ('trial-and-error') method of debugging and correcting these specifications. From our experience [11], real soft-state protocols can be rather complex and difficult to be analysed through implementations. This is partly due to the fact that soft-state protocols have two or more network nodes, which must maintain independently several types of timers and soft states associated with a given end-to-end session. Moreover, they are generally specified informally and imprecisely. Therefore, model-based approaches are preferred.

In this paper we employ a model-based approach to study the basic functionalities and aliveness properties of general soft-state systems, using the Specification and Description Language (SDL) [15] and the Message Sequence Charts (MSCs) [16]. As successfully demonstrated in experiences in modelling other communication protocols (e.g. [17]), SDL and MSCs are efficient tools for such tedious tasks. By presenting a general architecture for state management systems and modelling one representative system with SDL/MSC, we demonstrate the feasibility of applying this approach in the modelling of soft-state protocols, analysing their basic properties and advocating a potential way of improving protocol descriptions. Current work is limited to the general state management system rather than a real protocol; a few minor weaknesses still remain in the tools. In this paper we focus on the functionality modelling and validation of different variants of soft-state protocols and hard-state protocols described in [18], including verification of (the absence of) deadlocks and livelocks. Our goal here is not to model a real soft-state protocol such as RSVP or CASP, or hard-state protocols like ST-II or TCP, but rather to model general soft-state protocols in order to capture and verify the essential concepts and general functionalities of interest. There are other, non-functional aspects of these protocols, such as complexity and performance, but they are beyond the scope of this paper.

The rest of this paper is organised as follows. We first shortly review the related work in Section 2. Given the importance and difficulty of analysing soft-state protocols, we present a general architecture covering all variants of soft-state protocols in Section 3, followed by SDL models of different soft-state variants in Section 4 and a verification of the models (Section 5). We summarise our modelling experiences and conclude this paper in Section 6.

2 Related work

2.1 Studies on soft-state protocols

System designers argue soft state is 'better' than hard state, and by using soft state the handling of network condition changes is 'easy' [19, 20]. However, these claims are based more on intuitive, high-level thoughts and explanations, rather than on formal, exhaustive modelling and analysis. In contrast to original expectations, soft-state protocols that have been developed so far (and also those currently under development) are still far from being simple, especially when coupled with channel reliability, multicast sessions or traffic control models.

There are two types of soft-state protocols which have been developed so far: end-to-end protocols and hop-by-hop protocols. The former only involves certain states in two communicating end systems, without involving any other nodes in between; examples of this type include RTCP and SIP. Alternatively, hop-by-hop protocols, such as RSVP and CASP, involve state manipulation in one or more intermediate node(s) in between in addition to the states in the communicating ends. For the purpose of demonstration and general discussions of soft-state operations, we have chosen to use the latter since it is more representative and comprehensive.

Given the particular importance of soft-state protocols, recent studies have looked at issues regarding their modelling and analysis. Raman and McCanne [20] presented a model for the soft-state notion based on Jackson queueing networks; a performance study of hard-state and soft-state signalling protocols was performed by Ji *et al.* [18]. Unfortunately, these studies lack more detailed formal modelling and validation. In a study performed by Bradley *et al.* [21] regarding the correctness and interoperability issues with the HTTP protocol, the authors established that multi-stage interactions between the HTTP server and clients are stateful, error-prone and can be automatically verified by using a formal checking tool SPIN [22]. However, their study is limited to application-layer hard-state management between two end nodes and does not consider the soft-state paradigm for packet-switching networks (which may involve multi-hop behaviours of state management systems).

2.2 An introduction to SDL and MSC

As described earlier, SDL and MSCs are two potential techniques for modelling soft-state protocols. In SDL, a system is divided into building blocks that communicate using channels, these blocks are composed of processes. Processes (within a block) are connected using signal routes. Each process is an extended finite state machine, which has its own infinite queue and is assumed to operate independently from all other processes. MSCs are another valuable description technique for visualising and specifying inter-system, asynchronous component interaction. The strength of MSCs lies in their ability to describe communication between cooperating processes. Each process is represented as an identifier and has a process lifeline that extends downward. There are arrows representing messages passed from a sending to a receiving process. Messages not starting or ending at a process lifeline are exchanged with users, be they human or mechanical (the 'environment').

Detailed descriptions of SDL and MSC can be found in [15, 16, 23] and an example of modelling systems using the combination of SDL and MSC can be found in [17]. Modern SDL development tools like Telelogic Tau SDT also support verification and validation based on developed models.

3 General architecture of state management systems

As described by Ji *et al.* [18], in contrast to hard-state (HS) protocols, soft-state (SS) protocols can be further classified into four variants: 'pure' soft state (pSS), soft state with explicit removal (SS + ER), soft state with reliable trigger (SS + RT) and soft state with reliable trigger/removal (SS + RTR). In this Section, we present a simple abstraction and typical operations for all these state management

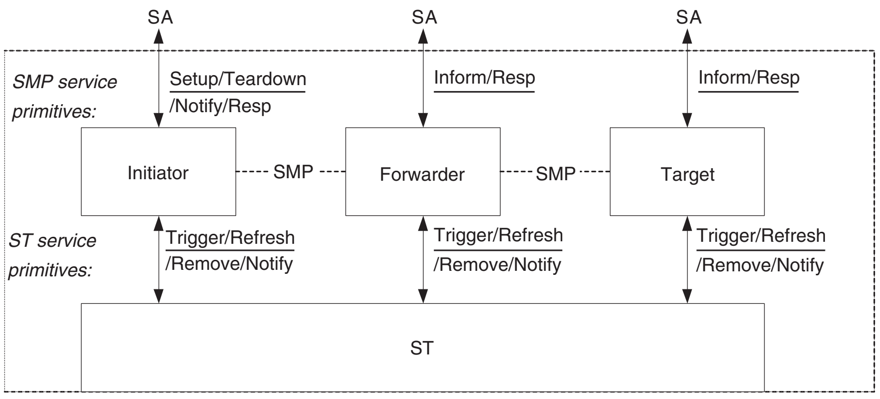

SA
SA
SA
SMP service primitives:
Setup/Teardown /Notify/Resp
Inform/Resp
Inform/Resp
Initiator
SMP
Forwarder
SMP
Target
ST service primitives:
Trigger/Refresh /Remove/Notify
Trigger/Refresh /Remove/Notify
Trigger/Refresh /Remove/Notify
ST

Fig. 1 *General architecture of state management systems*

protocols (both soft state and hard state), in addition to possible problems that may occur in their operations.

We begin our modelling with a generic architecture for state management systems as shown in Fig. 1, which covers two services and one protocol as below:

- The state message transport (ST) service, which transmits state messages over the lossy channel.
- The (soft or hard) state management service (SMP service), which is the service rendered by the state management protocol to the state application (SA).
- The state management protocol (SMP), which manages state in the network nodes.

3.1 Expected behaviour of state management protocols

Behaviours of a state management protocol are determined by timers and state messages used by the three types of protocol entities, namely the state initiator, forwarder and target. Timers and state messages: an important feature of soft-state systems is timers. In a soft-state system, there can be three types of timers dealing with state management: the state timer which will expire the state unless refreshed, the refresh timer which triggers periodical refreshes, and the retransmission timer which triggers periodical retransmission of trigger or removal messages (SS + RT or SS + RTR). In HS systems there are only retransmission timers for state trigger and removal messages, while state timers and refresh timers are necessary for SS systems for operating state refresh messages. The existence or absence of these timers and different operations for state messages in a state management protocol determines which protocol type it belongs to.

3.1.1 Protocol behaviour: We present the MSC description (shown in Fig. 2) to illustrate the messages and mechanisms of various state management protocols. The communications between the three types of entities are realised through several service primitives (Setup, Trigger, Teardown, and optionally, Refresh, Resp, Notify and Remove). For the purposes of simplicity, we omit the Inform and Resp primitives since they generally do not change state information.

The protocol communication takes place in three possible, distinct phases:

- *State setup*: This phase is initialised by SA in the entity Initiator with a Setup. Initiator can thereafter issue a Trigger towards the entity Target. Upon the receipt of the Trigger, every Forwarder entity creates a state (which is associated with a state timer for soft-state protocols), and then forwards the Trigger message on. When the Target receives the Trigger, it creates a state (which is associated with a state timer for soft state protocols).
If HS, SS + RT or SS + RTR is used, additionally the Target issues back a Notify to the Initiator when it receives a Trigger. In this case, the Initiator starts a retransmission timer before it issues a Trigger. If it does not receive a Notify after the Retransmission timer expires, the Trigger is transmitted again. After repeating certain times, the retransmission stops and the Initiator returns to the initial state.
- *State maintenance*: This phase is only used in soft-state protocols. Upon the expiration of the refresh timer, the Initiator sends a Refresh [Note 1] towards the Target and restarts the timer. Any forwarder receives the Refresh checks whether a corresponding state already exists: if yes, it refreshes the state timer, otherwise it recovers a state together with a state timer. Then it forwards the Refresh on towards the Target.
Upon receipt of the Refresh, similar to the Forwarder, the Target recovers state or just refreshes the state timer. If no Refresh is received in a Forwarder or Target before the state timer expires, state will be removed.
- *State teardown*: In pSS or SS + RT, there is no such phase; an Initiator remains inactive and state expires in all the other nodes when their state timers time out. In other state management protocols, a Remove is issued by the Initiator towards the Target to remove all state and associated timers (should they exist). Additionally, if HS or SS + RTR is used, after the Target receives Remove and removes its state (and timers), a Notify is sent back to the Initiator. An Initiator in a HS or SS + RTR system follows a way similar to the retransmission in the state setup phase if no Notify is received within a given time.

3.2 Possible problems in state management operations

Because the above general model captures the key concepts and operations of all the five possible types of state management protocols, we believe it forms the basis for studying actual behaviours of real state management systems. The most obvious problem that occurs in state management protocols is failure to install or remove state correctly. In addition to this, there are timing considerations to be taken into account, since a protocol of this kind needs to be able to react appropriately to timer events and the receipt of state messages. An installed state

Note 1: Refreshes described here are limited to those originating from the Initiator. Some SS protocols (for example, RSVP) further allow more reactive operations upon network condition changes, where Refresh messages can also be initiated at an intermediate node. This behaviour is not covered in this paper.

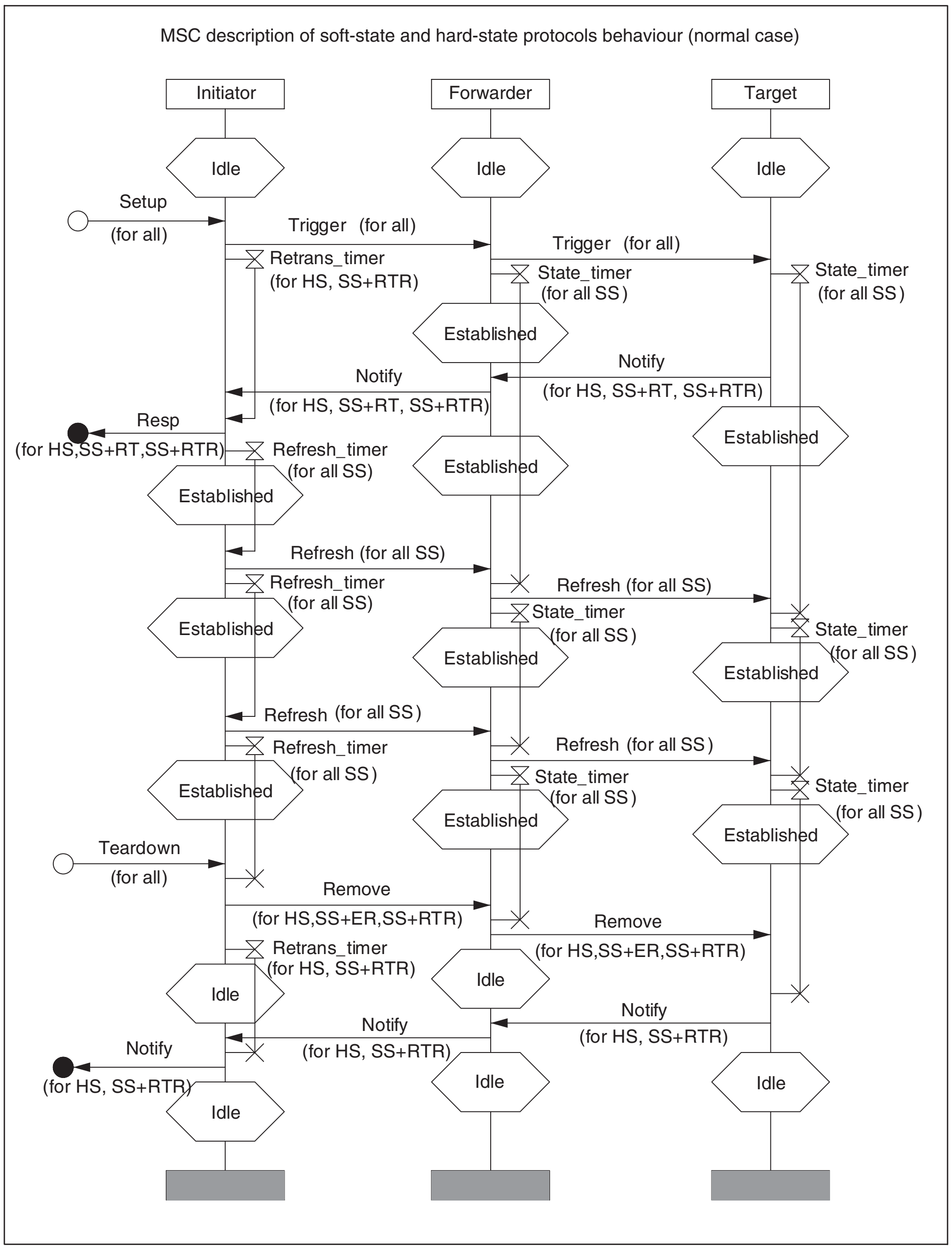

MSC description of soft-state and hard-state protocols behaviour (normal case)
Initiator
Forwarder
Target
Idle
Setup (for all)
Trigger (for all)
Retrans_timer (for HS, SS+RTR)
State_timer (for all SS)
Established
Notify (for HS, SS+RT, SS+RTR)
Resp (for HS,SS+RT,SS+RTR)
Refresh_timer (for all SS)
Refresh (for all SS)
Teardown (for all)
Remove (for HS,SS+ER,SS+RTR)
Notify (for HS, SS+RTR)

Fig. 2 *MSC for various state management protocols operation (normal case)*

can become invalid either owing to the receipt of a removal message or when the state timer expires. The latter may occur either when a state refresh or notify message is lost during its transmission, or when the state initiator crashes.

There are other potential problems with state management protocols. For example, there can be infinite state management loops, i.e., state messages enter a transmission circle somewhere between an initiator and a target. This kind of loop can result in two possible errors:

- Deadlocks: a state management system goes into a state without possibilities to enter another state. A deadlock is most often caused by two processes waiting for a message from each other; the result is that both wait infinitely. In pSS, deadlocks are theoretically impossible, as it only allows refresh and expiration operations for a state, and there is no resource contention. However, we cannot exclude this possibility in more comprehensive state management protocols, owing to more complex synchronisation and/or notification mechanisms.
- Livelocks: a state management system enters into an endless loop consisting of a set of interacting states which cannot progress towards a next expected step of the protocol's behaviour.

These cases are very undesirable since the state management behaviour in any individual node appears to be valid, but in reality it cannot further deliver other desired messages or jump out of certain running state(s). For example, the following operation could be possible in the original description of SS + RT [18].

Example 1: A deadlock behaviour in SS + RT. A brief explanation of such a scenario is as follows:

1) An Initiator initially sends a Trigger message to a Target through the Forwarder. The link between the Forwarder and the Target then suddenly goes down.

2) The Trigger message is lost before reaching the Target, and the Initiator cannot receive a desired Notify message that acknowledges the success of state installation along the path.

3) After the retransmission timer expires, the Initiator resends the Trigger towards the Target.

4) Again the Trigger is lost, and the Initiator still thinks it is simply the result of random loss and retransmits the Trigger.

5) If there is no specification for conditions that stop the sending of the Triggers, the result is a deadlock in which case the Initiator expects a Notify (but cannot receive it) after sending out the Trigger while the Responder expects a Trigger (but cannot receive it).

This error is somewhat easy to detect and fix. However, sometimes such flaws can be very subtle, so that even senior designers miss them in protocol specifications, particularly in IETF informal, text-based specifications for complex operations of state management protocols, which were designed typically without formal modelling, validation and verification. For example, the inadvert synchronisation problem [24] was noticed as an important issue for periodical soft-state systems. In real specifications of some soft-state protocols, for example RTCP [6] and RSVP [5], designers have tried to avoid this problem by setting the refresh timers to be varied randomly (e.g., over the range [0.5, 1.5] times the calculated interval). However, true randomness in a real implementation is hard to achieve. Moreover, as these interval management requirements are sometimes not mandatory in protocol specifications (e.g., as a 'should' requirement in RSVP), typically in practice default fixed values (30 seconds in the case of RSVP [24]) are used for implementation simplicity. All these contribute to the potential synchronisation problem of soft-state management.

4 Modelling soft-state protocols with SDL

4.1 Key modelling issues and the system model

Based on the general architecture covering the key concepts of state management protocols, we chose the most comprehensive state management protocol, SS + RTR in a hop-hop manner, as the target for modelling. Other variants may be derived easily from this model by removing certain messages, timers, state transitions and/or logic.

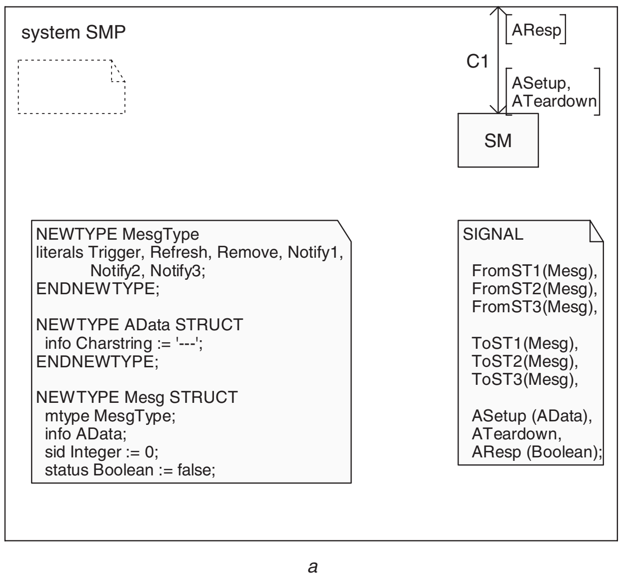
The address of each node is represented by a pre-assigned integer, 1, 2, 3, respectively. The state message format is also simplified as shown in Fig. 3*a*. There are further issues vital for the modelling process:

1) How do we model ST, in order to allow state messages (generally from an Initiator to a Target) to be visible for an intermediate Forwarder?

2) How do we model the lossy channel, which can be of a given loss rate?

3) How do we model the duration (start and end) of a session state? This also naturally reflects the system model, namely which information should be visible inside the system, and which needs to be put in the environments.

Figure 3*b* shows the SMP system model for SS + RTR. We assume the SMP system model to be composed of three nodes: an Initiator, a Forwarder and a Target, each of which is represented by a process. Furthermore, an additional process ST is used to transmit SS messages from the Initiator towards the Target, or reverse. All four processes form the only block SM of the system.



```
NEWTYPE MesgType
literals Trigger, Refresh, Remove, Notify1,
        Notify2, Notify3;
ENDNEWTYPE;

NEWTYPE AData STRUCT
  info Charstring := '---';
ENDNEWTYPE;

NEWTYPE Mesg STRUCT
  mtype MesgType;
  info AData;
  sid Integer := 0;
  status Boolean := false;
```

```
SIGNAL

  FromST1(Mesg),
  FromST2(Mesg),
  FromST3(Mesg),

  ToST1(Mesg),
  ToST2(Mesg),
  ToST3(Mesg),

  ASetup (AData),
  ATeardown,
  AResp (Boolean);
```

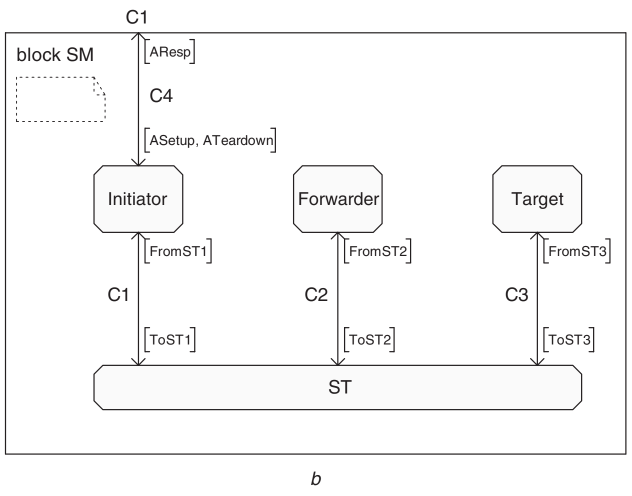


Fig. 3 *Description of SDL system and block*
a System SMP
b Block SM

4.2 ST modelling

ST is modelled in Fig. 4. It can be used for all variants of state management protocols. Here we use random Abstract Data Type (ADT) to simulate a given loss rate of the link. Note the transport service should determine the direction (forward or backward) according to the type of the message it receives. A more comprehensive ST can have different loss rates in different links and this feature is to be added in the next step.

4.3 SMP entities modelling

Figures 5–7 show the detailed models for each of the three SMP entities.

To model the duration of a session state, the Initiator process communicates with environments through an SA-SMP interface. When it receives an ASetup with certain state information data, it assigns a new session identifier (sid) and installs a soft state locally, before going through the state setup and maintenance phases. When Initiator receives an ATeardown from the SA-SMP interface, it enters into the teardown phase. For simplicity sid is currently set as a fixed value. To avoid confusion different notification messages have to be identified: Notify1 for notifying the success of a state setup, Notify2 for notifying

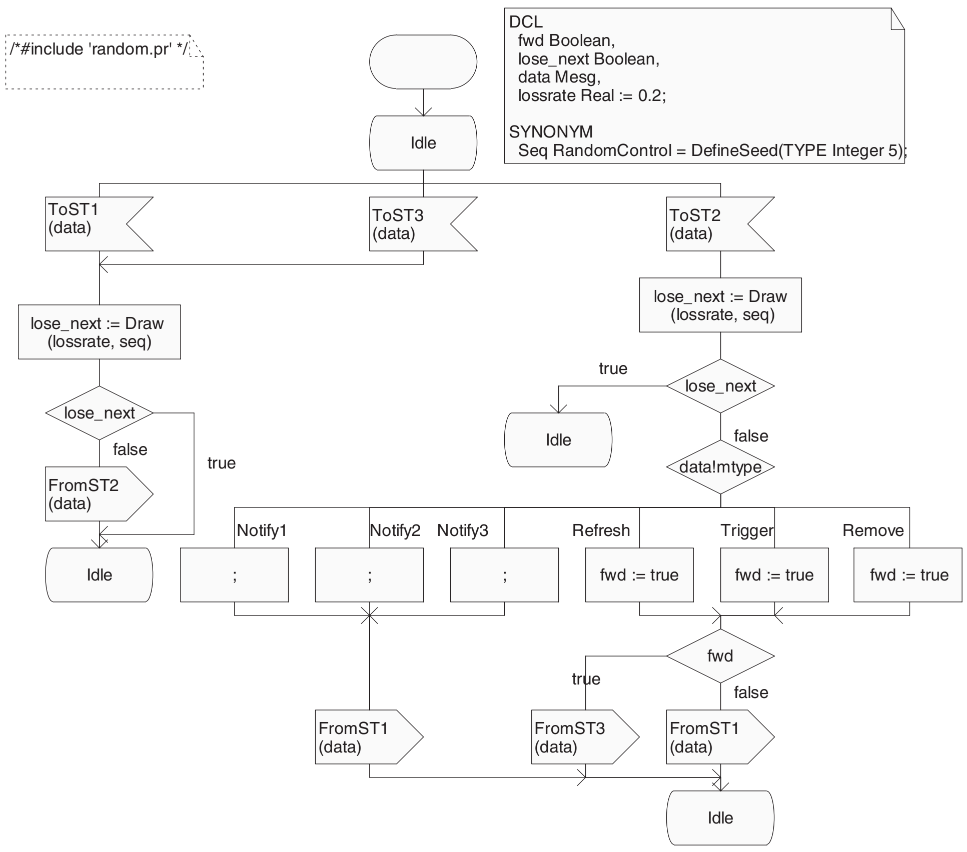


Fig. 4 *SDL model for state message transport*

state timer expiration in the Target, and Notify3 for notifying the success of a state teardown.

The Forwarder process first listens to ST input. If a Trigger or Refresh message arrives, it installs a soft state locally and forwards the message on. Expiration of a local state in Forwarder only removes itself.

The Target process installs and (if it should exist) refreshes a soft state locally upon receipt of a Trigger or Refresh message. When receiving a Trigger or Remove message, Target needs to notify Initiator about it. Expiration of a local state in Target also accompanies a Notify2 to trigger a teardown.

5 Model verification, validation and further discussions

5.1 Methodology of modelling verification and validation

The finite state machines derived from the SDL model is shown in Fig. 8.

Since state management protocols involve distributed timers and message interaction, their correctness is hard to verify using an informal description. We use the Tau integrated package SDT 4.4 which includes support for SPIN, to verify the SDL model of the SS + RTR protocol against deadlocks, unspecified receptions, livelocks and unreachable states. In order to verify the model, we have chosen the following scenario case study to validate our design against some of the specific properties of the modelled SS + RTR protocol:

1) Establish a session between Initiator and Target.

2) Stop the state message transmission between Forwarder and Target.

3) Change the ST loss rate between Forwarder and Target to different values between 0 and 1.

4) Stop the Target process.

5) Let Initiator teardown the session.

With the above scenario, we have covered all ST service primitives and SMP service primitives as well as all important scenarios, but not all possible scenarios. Therefore, after checking the scenario, we have used Tau validator to validate all possible walk algorithms.

5.2 Results and further discussions

We generated a number of MSCs for the above scenario case study to check the protocol functionality at each stage of the simulation. For the SDL models designed in Section 3, MSCs shown in Fig. 9–11 represent some interesting protocol behaviours. Through a comparison with the general description, we are able to refine the abstract system and make more concrete descriptions.

Through this procedure, we have found there may be two sources in the observed protocol misbehaviour. There may be flaws in the original description, such as the missing default behaviour for certain message types, and unreachable states. For example, for SS + RT and SS + RTR protocols, [18] states:

'… First, trigger messages are transmitted reliably in SS + RT. Each time a trigger message is transmitted, the sender starts a retransmission timer (with value R). On receiving an explicit trigger message, the destination not only updates signalling state, but also sends an acknowledgment to the sender. If no trigger acknowledgment is received before the retransmission timer expires, the signalling sender resends the trigger message. Secondly, SS + RT also employs a notification mechanism in which

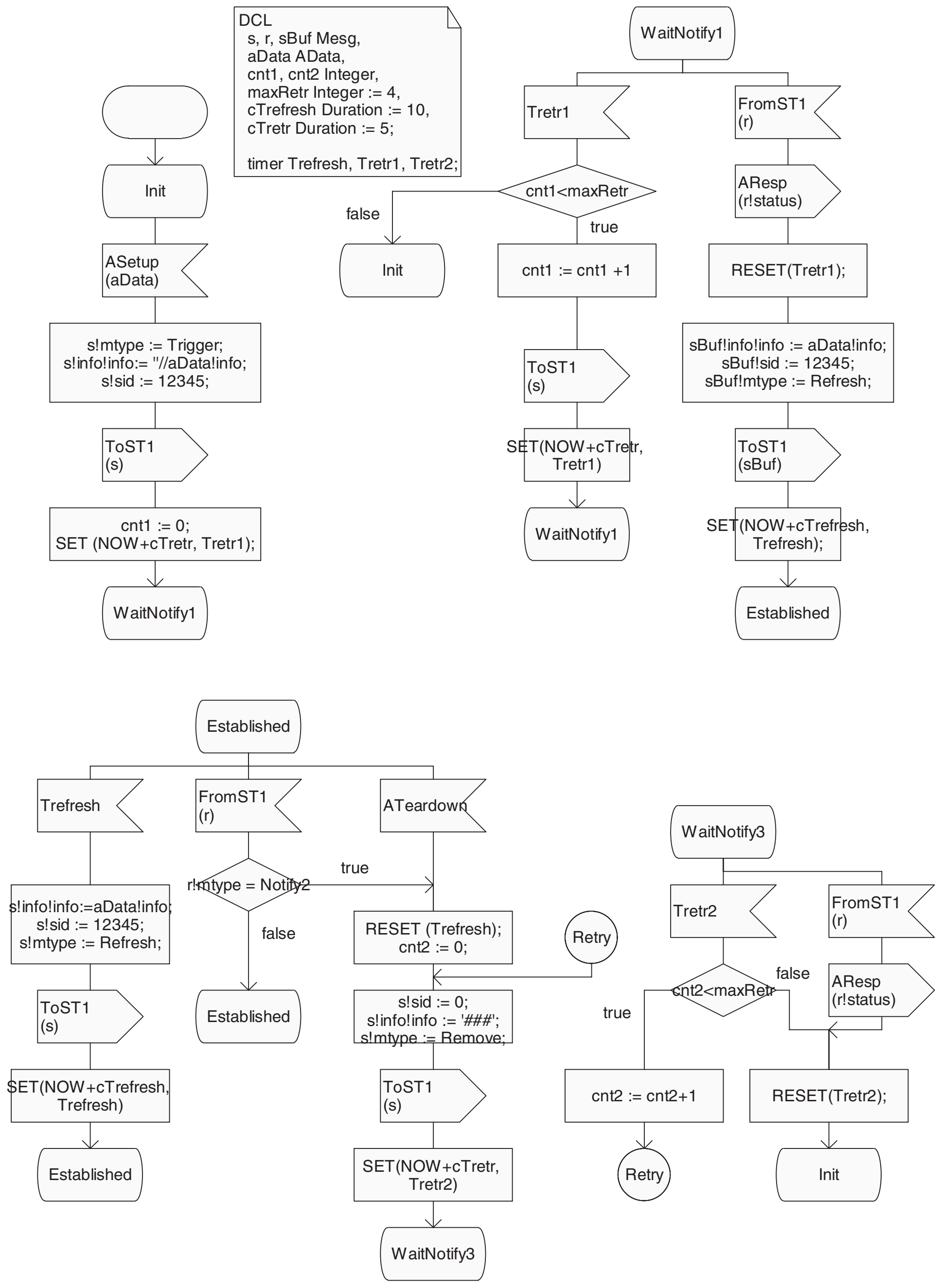


Fig. 5 *SDL model for Initiator (SS+RTR)*

the signalling destination informs the signalling sender about state removals due to state-timeout timer expiration. This allows the signalling sender to recover from false removal by sending a new trigger message.'

'SS + RTR is similar to the SS + RT approach, except that the SS + RTR approach uses reliable messages to handle not only state setup/update but also state removal.'

When we strictly followed this description, and validated the developed SDL model of SS + RTR, we easily detected a deadlock (through the integrated SDL simulation and validation environment) retransmission counters were missing in the protocol description. This confirms our statement that an imprecise informal specification can result in deadlocks and livelocks. In practice, many design flaws can be more subtle but easily validated with formal tools. Actually the models presented in Section 4 were the updated version according to a specification slightly different from [17].

Alternatively, the modelling process can introduce some modelling errors, especially if the specification is not very clear. In our work, for example, initially we introduced several steps that were anticipated to be necessary in order to work with the environments. However they were later detected to be a misunderstanding of the description, by tracing the obtained MSCs.

Some other results show that in the first version of the models Notify1, Notify2 and Notify3 messages received by Forwarder were not processed but consumed; they need to be sent back to ST. Also, misused values of timers can exclude the Initiator from entering the Established state.

We also come to the following conclusion: by adding a reliable trigger and explicit removal to soft-state protocols, the usage of state (reflected as network resources especially memories) can be more efficient. This can be explained, for example for the reliable explicit removal case, as if a user tries to remove a state, but the teardown message is lost during transmission, the state will remain in place until it

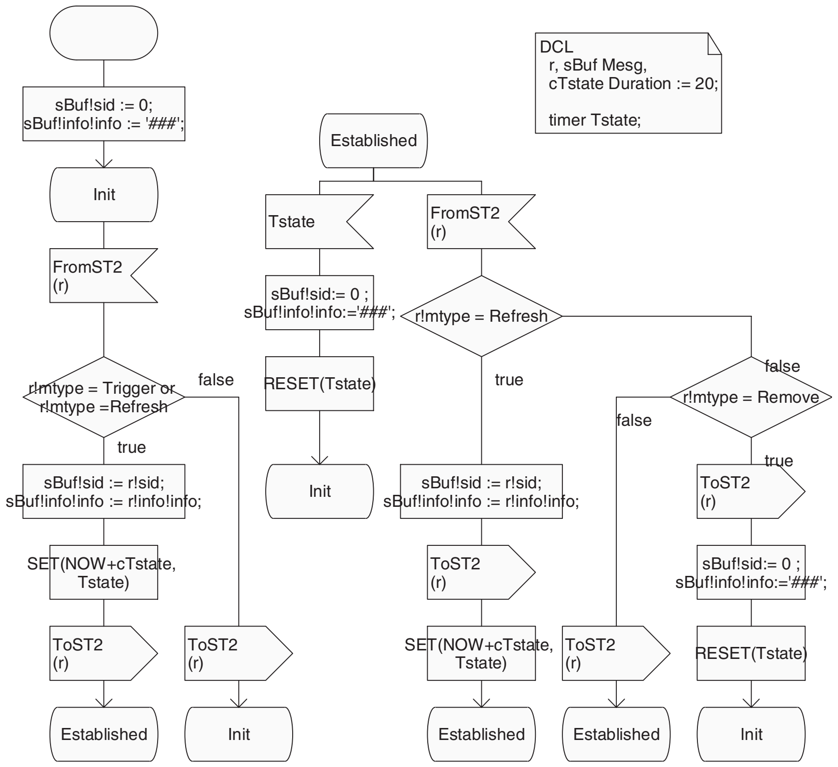


Fig. 6 *SDL model for Forwarder (SS+RTR)*

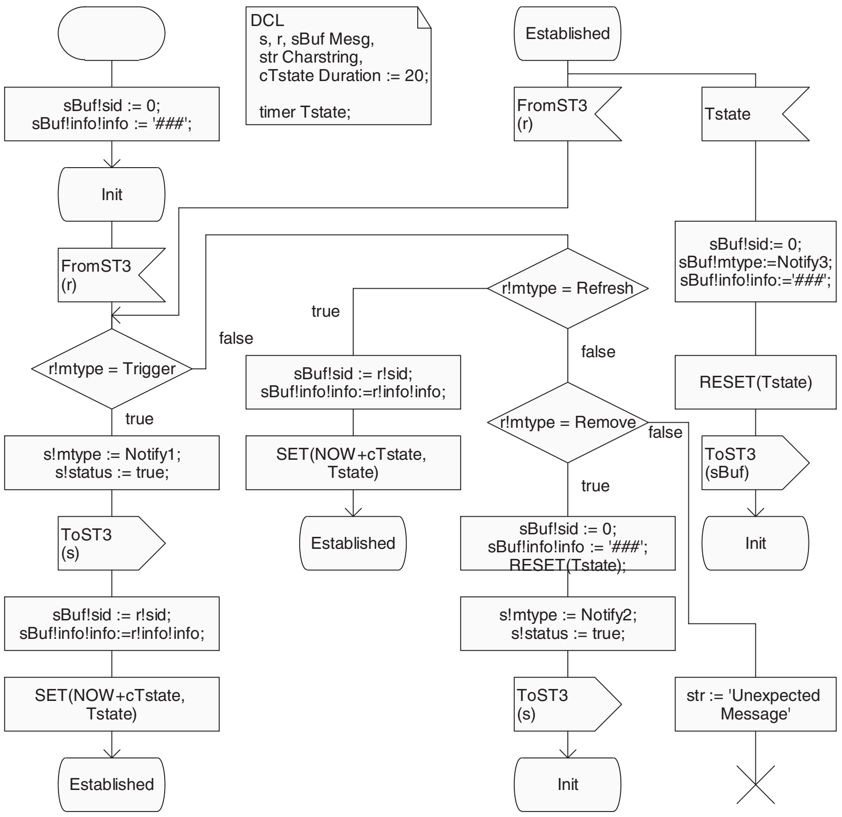


Fig. 7 *SDL model for Target (SS+RTR)*

times out after a relatively long time. Since state typically implies enhanced services for end-to-end communications, maintaining a state incurs costs, thus users must pay for the extra time that has been spent waiting for the state expiration.

As a result, we have found (for example, the retransmission counters and refresh durations – either explicit or implicit – are missing in many text-based IETF specifications); through verification and validation of the SDL models, these could be avoided. With the developed model,

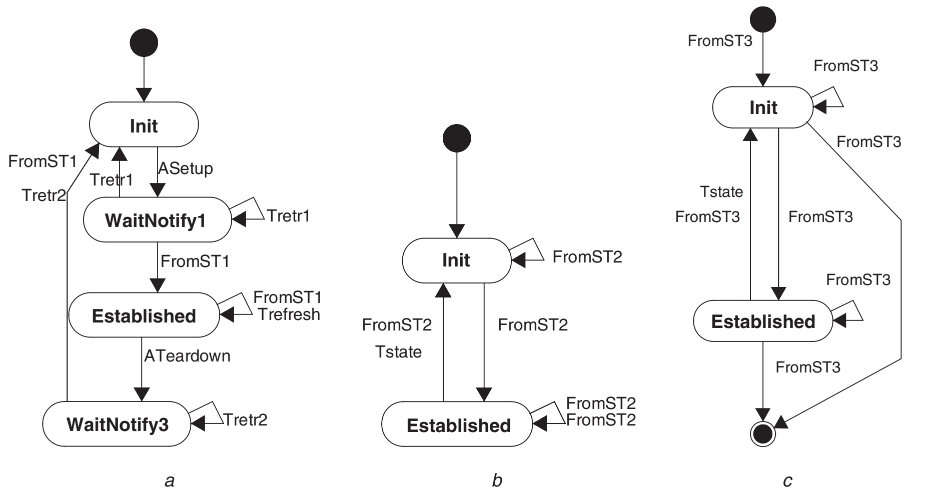


Fig. 8 *State diagram of a soft-state protocol (SS+RTR)*
a Initiator
b Forwarder
c Target

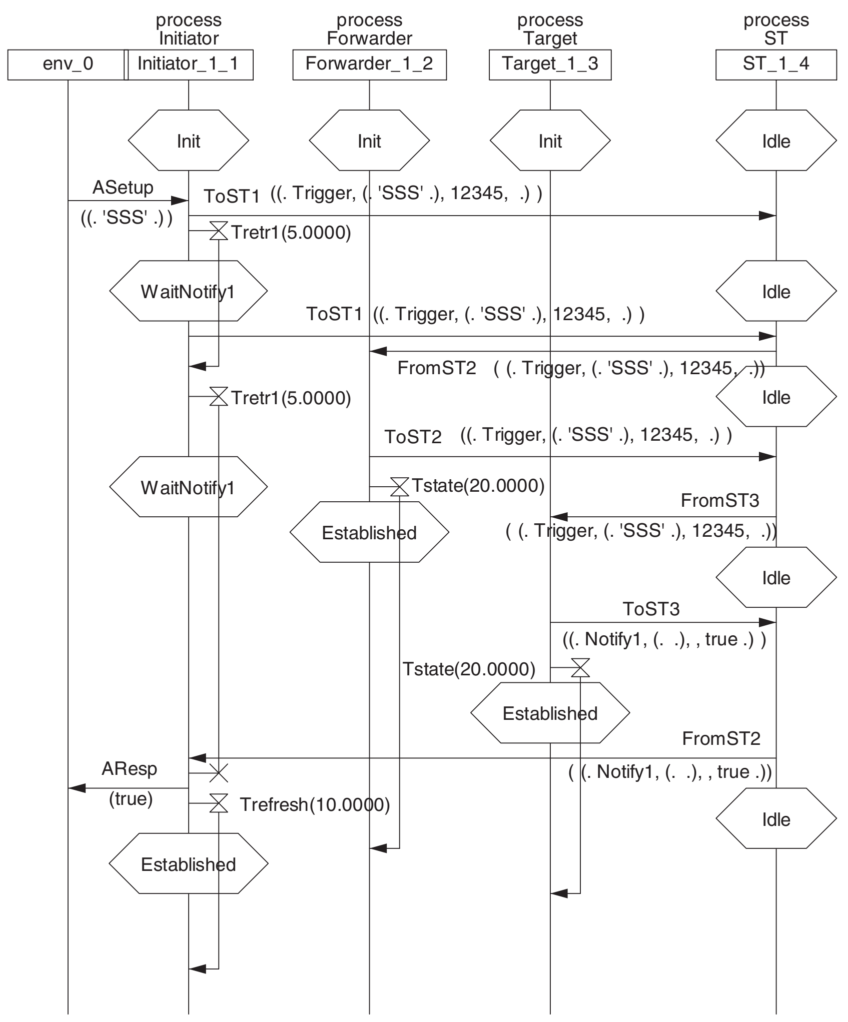
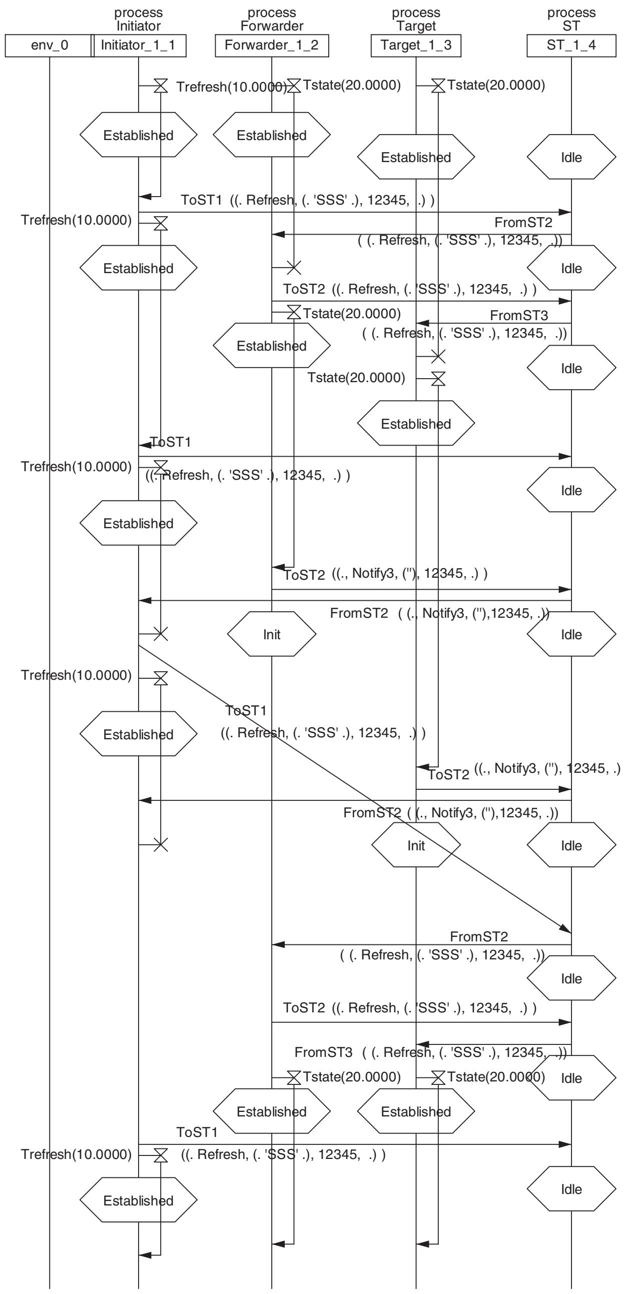


Fig. 9 *An MSC for state setup phase (SS+RTR)*

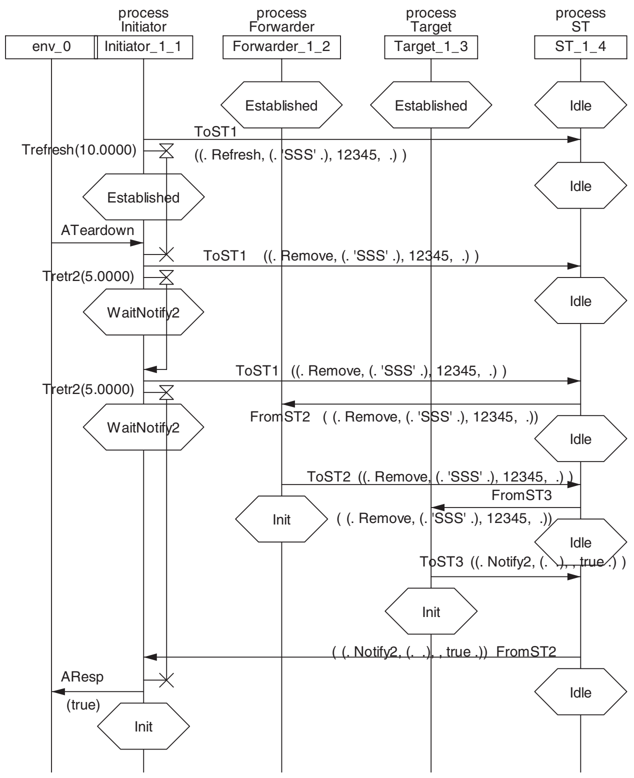
we have verified these aspects against the description of SS + RTR, and improved upon it.

Finally, while extending this model for other state management systems, we found it is fairly easy since we have abstracted the most critical and representative behaviours. We simply removed some unused message types and state machine entries, and validated successfully for all the five types of state management systems. Two particular issues were noted: first, the models developed in SDL usually do not cover performance aspect, because it is not the major objective of the current SDL development tools. However, with certain extensions, it is possible to also support protocol performance analysis in the tools [26, 27]. Second, when performance analysis is desired, under the



process Initiator
process Forwarder
process Target
process ST
env_0
Initiator_1_1
Forwarder_1_2
Target_1_3
ST_1_4
Trefresh(10.0000)
Tstate(20.0000)
Established
Idle
Init
ToST1 ((. Refresh, (. 'SSS' .), 12345, .))
FromST2 ((. Refresh, (. 'SSS' .), 12345, .))
ToST2 ((. Refresh, (. 'SSS' .), 12345, .))
FromST3 ((. Refresh, (. 'SSS' .), 12345, .))
ToST2 ((., Notify3, (''), 12345, .))
FromST2 ((., Notify3, (''),12345, .))

Fig. 10 *An MSC for state maintenance phase (SS+RTR)*



process Initiator
process Forwarder
process Target
process ST
env_0
Initiator_1_1
Forwarder_1_2
Target_1_3
ST_1_4
Established
Idle
Init
WaitNotify2
Trefresh(10.0000)
Tretr2(5.0000)
ATeardown
AResp
(true)
ToST1 ((. Refresh, (. 'SSS' .), 12345, .))
ToST1 ((. Remove, (. 'SSS' .), 12345, .))
FromST2 ((. Remove, (. 'SSS' .), 12345, .))
ToST2 ((. Remove, (. 'SSS' .), 12345, .))
FromST3 ((. Remove, (. 'SSS' .), 12345, .))
ToST3 ((. Notify2, (. .), , true .))
((. Notify2, (. .), , true .)) FromST2

Fig. 11 *An MSC for state teardown phase (SS+RTR)*

current modelling methodology, the ST is the central message transport entity, thus there is a scalability issue of the system with the number of the intermediate nodes. We have recently attempted an alternative methodology using decentralised message transport entities [28], but its implications require further study.

This work is derived from the first attempt we made on this topic as published in [29], in a revised and more accurate fashion. First, the state concept is clarified under the Internet end-to-end principle as the motivating fundamental background, not just soft-state versus hard-state discussion, to indicate the proposed method can be of potential use in the future Internet evolution and its relevance to the other network state protocol designs. Moreover, the state diagrams for the SS + RTR protocol are now derived to better illustrate the overall behaviour. In addition, this paper also fixes a few modelling bugs introduced in [29], and the verification results are now checked against a concrete excample of an imprecise specification given in [18].

6 Conclusions

Given the fundamental importance of soft-state protocols, it is vital to ensure that their behaviour is specified correctly. We have generalised and modelled behaviour for different variants of soft-state communications with the aid of formal techniques SDL and MSC. We observed that formal techniques are of great help for efficient designing and engineering of soft-state communication models and protocols, and in particular functional behaviour (through checking for possible deadlocks and livelocks). We concluded that two sources of protocol misbehaviour are possible, namely the protocol design and specification flaws and the modelling errors (in our case, owing to the ambiguity of the specifications), and the results based on SDL/MSC modelling help to correct these problems. However, we found that a weakness exists with the utilised formal tools concerning protocol performance; a more realistic performance evaluation has to rely on tools with better real-time support. We are currently exploring various ways on studying performance aspects of this approach. We believe our ongoing activities with modelling some real soft-state protocols such as RSVP and CASP will bring more insights in this field and help effectively making more robust standards.

7 Acknowledgment

We would like to thank M. Zoric, S. Müller and T. Tsenov for their helpful comments.

8 References

1 Carpenter, B.: 'Architectural principles of the Internet', Internet Engineering Task Force RFC 1958, June 1996
2 ITU-T Recommendation Q.2931: 'Broadband integrated services digital network (B-ISDN) digital subscriber signalling system No. 2 (DSS 2) – user-network interface (UNI) layer 3 specification for basic call/connection control', Feb. 1995

3 Delgrossi, L., and Berger, L.: 'Internet stream protocol version 2 (ST2) protocol specification - version ST2+', RFC 1819, Aug. 1995
4 Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D.: 'RSVP: a new resource ReSerVation protocol', *IEEE Netw.*, 1993, **7**, (5), pp. 8–18
5 Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S.: 'Resource ReSerVation protocol (RSVP) – version 1 functional specification', RFC 2205, Sept. 1997
6 Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V.: 'RTP: a transport protocol for real-time applications', RFC 1889, Jan. 1996
7 Estrin, D., Farinacci, D., Helmy, A., Thaler, D., Deering, S., Handley, M., Jacobson, V., Liu, C.-G., Sharma, P., and Wei, L.: 'Protocol independent multicast-sparse mode (PIM-SM): protocol specification', RFC 2362, June 1998
8 Rosenberg, J., Schulzrinne, H., Columbia, U., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E.: 'SIP: session initiation protocol', RFC 3261, June 2002
9 Schuzrinne, H., Tschofenig, H., Fu, X., McDonald, A.: 'CASP – cross-application signaling protocol', Internet draft (draft-schulzrinnensis-casp-01), work in progress, Mar. 2003
10 Schulzrinne, H., Fu, X., Pampu, C., and Kappler, C.: 'Design of CASP – a technology independent lightweight signaling protocol'. Proc. IPS'03, Salzburg, Austria, Feb. 2003
11 Fu, X., Hogrefe, D., and Willert, S.: 'Implementation and evaluation of the cross-application signaling protocol (CASP)'. Proc. ICNP 2004, Berlin, Germany, Oct. 2004
12 Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., and Semke, J.: 'Known TCP implementation problems', Internet Engineering Task Force RFC 2525, Mar. 1999
13 Postel, J.: 'Transmission control protocol', RFC 793, Sept. 1981
14 Zhang, Y.-X., Takahashi, K., Shiratori, N., and Noguchi, S.: 'An interactive protocol synthesis algorithm using a global state transition graph', *IEEE Trans. Softw. Eng.*, 1988, **14**, (3), pp. 394–404
15 ITU-T Recommendation Z.100: 'Specification and Description Language (SDL)', 1999
16 ITU-T Recommendation Z.120: 'Message Sequence Chart (MSC)', 1999
17 Schaible, P., and Gotzhein, R.: 'View-based animation of communication protocols in design and in operation', *Comput. Netw.*, 2002, **40**, (5), pp. 621–638
18 Ji, P., Ge, Z., Kurose, J., and Towsley, D.: 'A comparison of hard-state and soft-state signaling protocols'. Proc. SIGCOMM 2003, Karlsruhe, Germany, Aug. 2003
19 Sharma, P., Estrin, D., Floyd, S., and Jacobson, V.: 'Scalable timers for soft state protocols'. INFOCOM'97, Kobe, Japan, Apr. 1997
20 Raman, S., and McCanne, S.: 'A model, analysis, and protocol framework for soft state-based communication'. Proc. SIGCOMM 1999, Cambridge, MA, USA, Sept. 1999
21 Bradley, A., Bestavros, A., and Kfoury, A.: 'Safe composition of web communication protocols for extensible edge services'. Proc. Workshop on Web Content Caching and Distribution (WCW), Boulder, Colorado, USA, Apr. 2002
22 Holzmann, G.: 'The model checker SPIN', *IEEE Trans. Softw. Eng.*, 1997, **23**, (5), pp. 1–17
23 Ellsberger, J., Hogrefe, D., and Sarma, A.: 'SDL – object oriented language for communication systems' (Prentice Hall, 1997)
24 Floyd, S., and Jacobson, V.: 'The synchronization of periodic routing messages', *IEEE/ACM Trans. Netw.*, 1994, **2**, (2), pp. 122–136
25 Pan, P., and Schulzrinne, H.: 'Staged refresh timers for RSVP'. Proc. of 2nd Global Internet Conference, Phoenix, AZ, USA, Nov. 1997
26 Steppler, M.: 'Performance analysis of communication systems formally specified in sdl'. WOSP '98: Proc. of the First International Workshop on Software and Performance, New York, NY, USA, ACM Press, 1998, pp. 49–62
27 Malek, M., 'Perfsdl: interface to protocol performance analysis by means of simulation'. Proc. SDL'99, Montreal, Canada, June 1999 (Elsevier Science, Amsterdam), pp. 441–455
28 Werner, C., Fu, X., and Hogrefe, D.: 'Modeling route change in soft state signaling protocols using SDL: a case of RSVP'. Proc. of SDL Forum 2005, Grimstad, Norway, June 2005
29 Fu, X., and Hogrefe, D.: 'Modeling soft state protocols with SDL'. Proc. IFIP Networking 2005, (*Lect. Notes Comput. Sci.*, **3262**), pp. 289–302