Experiences in Using SDL to Support the Design and Implementation of a Logical Link Layer Protocol

Laila Daniel, Matti Luukkainen, and Markku Kojo

Department of Computer Science, University of Helsinki, Finland
{ldaniel, mluukkai, kojo}@cs.helsinki.fi

Abstract. We have used SDL to support the design and implementation of SLACP, a novel logical link layer protocol to enhance the performance of TCP over wireless WAN links. The protocol was modeled in SDL and successive refinements of its design were carried out based on feedback obtained from using the validation facilities of Telelogic Tau 4.4 SDT.

1 Introduction

Transmission Control Protocol (TCP) [1] is the dominant transport protocol in the Internet. TCP performs well in wired networks where packet losses occur mainly due to congestion. Unfortunately TCP performance suffers in wireless networks where packet losses due to link errors and handoff are predominant [2]. We have designed a link-aware protocol called Satellite Link Aware Communication Protocol (SLACP) to improve TCP performance over wireless WAN links such as satellite links[1].

We use SDL [3] to support the design and implementation of SLACP based on a *validation-oriented design approach*. In this approach an initial abstract executable model that covers the basic functionality of the protocol is developed. In the abstract model the number of data elements (both in signals and in processes) is kept to a minimum. We then validate the internal consistency and functionality of the abstract model. Subsequently we add details to the basic model in an incremental manner by specifying additional elements such as signal parameters, error handling and validate the enhanced model. This step is iterated until all the elements of the protocol have been added to the model. This incremental design approach helps in controlling the complexity of the protocol and in evolving a design that could be validated. This approach is facilitated by modeling SLACP in SDL and using the feedback obtained from the validation facilities provided by Telelogic Tau 4.4 SDT [4]. The SDL model of SLACP is the basis of the implementation of the protocol in Linux [5, 6].

[1] This work was done as a part of Transat project sponsored by European Space Agency.

A. Prinz, R. Reed, and J. Reed (Eds.): SDL 2005, LNCS 3530, pp. 187–197, 2005.
© Springer-Verlag Berlin Heidelberg 2005

The rest of the paper is organized as follows. Section 2 gives an overview of SLACP. Sections 3 and 4 are devoted to the modeling of SLACP using SDL. Section 5 deals with SLACP validation and section 6 compares the SDL model with the actual implementation of the protocol. In section 7 we discuss the conclusions of the paper.

2 Satellite Link Aware Communication Protocol (SLACP)

SLACP is a full-fledged logical link layer protocol which provides data transfer, error recovery and a Quality of Service (QoS) mechanism for enhancing TCP and other Internet protocol performance on wireless WANs. A detailed description of the protocol with performance results is given in [5, 6]. We briefly describe the salient features of SLACP needed to understand the design of the protocol.

The main goal of SLACP is to reduce the errors perceived by TCP. SLACP uses selective repeat sliding window mechanism [7] for data transfer. Error recovery is done using a combination of Automatic Repeat reQuest (ARQ) and Forward Error Correction (FEC). SLACP provides flow control mechanisms between the Internet Protocol (IP) layer and the satellite Medium Access Control (MAC) layer. SLACP performs a QoS mapping in which all packets belonging to an IP QoS class are directed to a SLACP channel with appropriate QoS parameters. SLACP supports several logical channels with independent choice of QoS parameters and error control schemes.

SLACP frames are of two types, *control frames* and *data frames*. Control frames carry the signals to setup, disconnect and reset a channel besides providing information regarding acknowledgments, packet loss and flow control. Data frames carry the IP packets. Frames transmitted over each logical channel are delivered independently of the frames sent over the other channels. A single high priority channel, called *control channel*, allows timely delivery of time-critical frames such as retransmitted data frames, acknowledgments and control frames. The frames sent through the control channel are FEC encoded to make them robust against errors. By default all original data frames are sent without FEC encoding as the satellite and other wireless channels are relatively error free most of the time and FEC encoding consumes additional bandwidth.

A problem with the link layer recovery is that it may interfere with the TCP recovery if the link level recovery takes a long time. We designed SLACP to minimize the link recovery delay by providing a high priority channel for the time-critical frames, by using acknowledgments with selective repeat information (SACK blocks), by setting the ARQ persistence level to one, together with employing FEC-encoding to protect retransmissions, and by introducing special frames such as nothing_to_send frame and rxmtpkt_loss frame. SLACP can recover lost frames readily if the loss occurs at the beginning or in the middle of a burst of frames, because the loss can be detected immediately with the first successfully arriving frame. If the tail of a burst is lost, depending on the idle time until the next burst begins, SLACP might need to rely on a timer gener-

ated nothing_to_send frames getting through to trigger selective acknowledgment telling which frames were lost. Once this selective acknowledgment arrives, retransmissions take place as usual. When the SLACP receiver gets the signal rxmtpkt_loss from the sender indicating the permanent loss of a frame, it knows that the sender is not going to retransmit the frame again and that it can send all frames up to the lost frame in its buffer to the IP layer.

SLACP implements FEC encoding using Reed-Solomon codes. FEC encoding and recovery is done in a novel way to protect the frames from error bursts that tend to completely corrupt several consecutive frames. Therefore, the FEC-encoded redundancy is not added separately to each frame as usual. Instead, the frames to be FEC protected are organized as FEC blocks. Each FEC block consists of actual frames and redundancy frames. The FEC-encoded redundancy is added to the redundancy frames by computing the Reed-Solomon codeword vertically so that the i^{th} octet of each frame in a FEC block comprises a codeword. As soon as a predetermined amount of actual frames deserving FEC protection has been sent or a threshold timer expires, a proper amount of FEC-encoded redundancy frames are computed to complete the FEC block and are transmitted. If no actual frames in a FEC block are lost, the redundancy frames are not used at all at the receiver. If some actual frames are lost, the SLACP receiver waits till the last redundancy frame of that FEC block or a frame belonging to any of the later FEC blocks is received. If at least a minimum number of redundancy frames are present in an FEC block the lost actual frames can be recovered. This minimum number depends on the parameters of the error correcting code in use. Under the minimum number, lost frames cannot be recovered by error correcting.

3 Modeling SLACP Using SDL

The link layer model also includes its interface to the layers above and below it. So in modeling SLACP we model the interface between SLACP and IP layer as well as the interface between SLACP and the Media Access Control (MAC) layer. The modeling of the IP and MAC layers allows us to see the flow control mechanisms between IP and SLACP and between SLACP and MAC. The MAC layer is also needed to model the delay and link losses in the satellite and wireless links.

The SLACP protocol engine consists of both the SLACP sender and receiver. The first question is whether to model the SLACP protocol engine (the combination of the SLACP sender and receiver) as a single entity and have two such engines as the communicating peer entities. This approach is the usual way of implementing a protocol and it helps to test the full duplex operation of the protocol. Since we are designing the protocol from scratch we model the protocol in a simple way by separating the sender and the receiver and testing the functionalities of the sender and the receiver separately. This results in a model where the SLACP sender receives packets from the IP layer and sends them to the SLACP receiver via the MAC layer. As SLACP is not concerned with the processing of the received data by the higher layer, it is not necessary to model

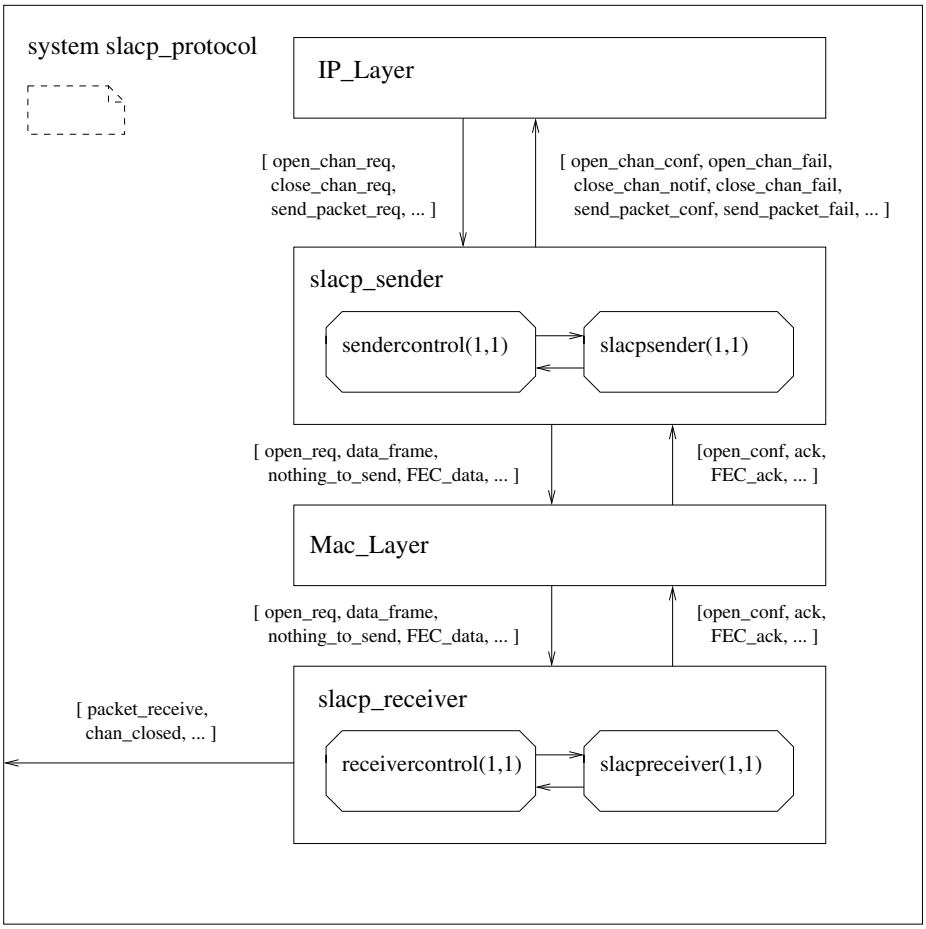

Fig. 1. SDL model of SLACP protocol

the IP layer at the receiver side, so in the SDL model the SLACP receiver sends the received packets to the system environment.

Thus the SDL system model of SLACP consists of the blocks *IP Layer, SLACP Sender, MAC layer and SLACP Receiver* (see fig. 1).

SDL allows us to define the system model in a top down manner. This helps to define processes for special functionalities inside each of the blocks above.

The data structures provided in SDL are convenient to model the various queues and frame types used in the protocol. Since the MAC buffer has different kinds of frames, the MAC frame is modeled using SDL choice construct. We use the SDL timer facility to implement the different timers associated with the protocol.

In the following sections we describe briefly the different blocks of the SDL model of SLACP. We only describe in words the implementation ideas behind

the various blocks. As the entire protocol model consists of roughly 100 pages of SDL code, no concrete SDL code is shown here.

3.1 IP Layer

The main functionalities at the IP layer are to generate the signals for SLACP channel establishment, channel reset and channel disconnect. The IP layer also generates data packets to be sent to the SLACP layer. The IP layer handles the flow control signals coming from the SLACP sender and the notifications from the SLACP sender to be given to higher layer protocol entities. The layers above IP are modeled as system environment. When the IP layer gives a packet to the SLACP sender it is assumed that the packet is given to the correct queue according to the QoS requirement.

The IP layer block consists of a *packet generator* process and a *signaling* process. The packet generator process handles the functionality associated with packet generation and the signaling process deals with opening, resetting and closing of the channel. The signaling process also gives signals to the packet generator process; for example, when it sends a signal to the SLACP sender to close the channel, it also signals the packet generator to stop generating packets.

Initially both the SLACP sender and receiver are in the idle state. When the sender gets a request from the IP layer to open a channel, it sends a corresponding SLACP frame open_req to the receiver, and both the sender and the receiver negotiate the parameters of the connection. If they agree, they go to the connected state and the SLACP sender sends an open_chan_conf packet to the IP layer. The SLACP sender and receiver remain in the connected state until a channel reset or a channel close request comes from the IP layer. If the opening of the channel between the sender and the receiver has failed, the SLACP sender sends a open_channel_fail to the IP layer. Closing of the channel is modeled using a timer called channel_close_timer at the IP layer and when it expires the signaling process issues a close_chan_req to the SLACP sender.

After the connection establishment, the IP layer issues a send_packet_req to the SLACP sender for each outbound packet to which SLACP sender replies with the status of the buffer. If the buffer is available at the SLACP sender, the IP layer sends packets to the SLACP sender. The SLACP sender also informs the IP layer whether a packet was successfully delivered to the receiver. The packet generator process controls the generation of packets using the information it receives from the SLACP sender regarding the buffer status. Thus flow control is modeled between the IP layer and the SLACP sender.

3.2 SLACP Sender

In principle there are n senders for the n different logical channels but as senders are independent it is adequate to model one sender. The SLACP sender consists of two processes, *sender* and *sendercontrol*. The sender process represents the entity carrying out the processing of signals for ordinary data channel and sendercontrol represents the entity carrying out the processing of signals in the

control channel. We model SLACP sender for a particular choice of QoS and ARQ-FEC parameters.

After connection setup, when the SLACP sender receives a packet from the IP layer it adds a header and a trailer to it to form a frame and enqueues the frame in the buffer. As SLACP is a sliding window protocol the sender always ensures that only a limited number of unacknowledged frames can exist at a time. The maximum number of unacknowledged or outstanding frames is called the the window size at the sender. If the window is not full, the sender process sends the SLACP frame to the MAC layer if the MAC buffer is free and updates the scoreboard data structure. The scoreboard records the information regarding the frame such as its sequence number, its address in the SLACP buffer and whether it has been retransmitted or not. The scoreboard data structure is used to implement the selective repeat mechanism. If the MAC buffer is not free the sender process keeps the data frame in its buffer. This way the SLACP sender controls the flow between the MAC and IP layers.

The sendercontrol gets the FEC encoded acknowledgment (ACK) frames from the SLACP receiver and it forwards the ACK frames to the sender process. If the data frame has been received correctly, the frame is dequeued from the sender buffer, otherwise it is retransmitted. When the sender receives a selective acknowledgment (SACK) from the SLACP receiver, it retransmits the lost frame to the sendercontrol. The details of the lost frame are obtained from the scoreboard. The sendercontrol forwards the FEC encoded retransmitted frames in an FEC block. SLACP is not meant to be a fully reliable link protocol and in the normal operation the ARQ persistence is set to one. If a retransmitted frame is lost, the SLACP sender sends a packet_loss_notification to the IP layer and also a rxmtpkt_loss to the SLACP receiver. When the SLACP sender has no data to send it sends a nothing_to_send frame to the receiver.

When the SLACP sender gets a request from the IP layer to close the channel it sends a data frame with no data to the SLACP receiver to indicate the closing of the channel. After receiving the ACKs for all the data sent to the receiver, the SLACP sender sends a close_channel_notif to the IP layer and goes to the idle state.

3.3 MAC Layer

The MAC Layer pictured here is the part of the MAC layer as seen by SLACP. So we abstract from the segmentation and reassembly at the MAC layer. The MAC layer has to perform the function of delivery of frames between SLACP sender and receiver. It also gives flow control information to both the SLACP sender and receiver. The MAC layer queues all the frames it receives from the SLACP layer and sends a frame to the SLACP layer when a link timer expires. The timeout of the link timer represents the delay in the wireless link. The MAC layer is also modeled to drop frames to simulate the behavior of frame losses due to bit corruption on the link. This is an important feature in order to validate the error recovery functionality in the SLACP protocol. We used a simple modulo based counter to determine which packet is to be dropped, though one can use random number generation to drop frames in an arbitrary manner.

3.4 SLACP Receiver

The SLACP receiver consists of two processes, namely *receiver* and *receivercontrol*. The receiver process represents the entity carrying out the processing of signals for ordinary data channel and the receivercontrol process represents the entity carrying out the processing of signals in the control channel.

SLACP receiver checks whether the frame received is a frame within the receiver window. The lower edge of the window represents the sequence number of the next frame to be received or the frame expected. When the frame received is the frame expected, the receiver process sends the received packet to the system environment. If the frames are received out of order within the window, the frames are kept in a reassembly queue. When the receiver process receives a number of data frames equal to an acknowledgment threshold value or when it receives an out of order frame, it sends an ACK frame. The ACK frame carries both a cumulative ACK sequence number and the SACK block. The SACK block indicates any out of order frames received after the cumulative ACK sequence number. The SLACP sender constructs an FEC block to retransmit all frames indicated as lost in the SACK block.

When the receivercontrol process receives an FEC block it sends an FEC_ACK to the sendercontrol process. Similar to the ACK frame, the FEC_ACK frame also has a cumulative ACK and an FEC-SACK block. The receivercontrol decodes the FEC block when it arrives and forwards the correctly received frames to the receiver process. Instead of using FEC coding to compute redundancy frames, each actual frame in the FEC block is duplicated as a redundancy frame. If a duplicate for a lost actual frame is present among the redundancy frames, it simulates the recovery of the frame.

4 The SDL Model as an Aid to the Design of SLACP

We develop the SDL model of SLACP in an incremental manner by successively refining the protocol design at each stage. Initially the protocol skeleton is built by including all the blocks and processes with minimal functionality in the blocks and processes. This helps to establish the basic interaction patterns between the processes and the blocks. Initially the functionalities for opening and closing a channel are included to build the basic model. We select these functionalities first since they are basic for communicating processes and in our model they involve all the protocol blocks in fig. 1. With this choice we could model the basic signaling mechanisms and see the patterns of interaction between the various design elements. The interactive mode of use of SDT helps to exercise the preliminary design by suitable choice of the parameters such as those for opening and closing a channel. For example, closing of a channel can be simulated by setting the duration of the channel_close_timer to a small value. After opening the channel, the IP layer will send a close_chan_req to the SLACP sender and this in turn closes the channel. From the Message Sequence Charts (MSC) generated we can see that the basic model is working.

In the second level of modeling the objective is to ensure that the data transfer can take place in the simplest possible setting. So we build enough additional functionality into the model to enable a single packet to be transferred from the sender to the receiver when there are no losses due to errors and there is no flow control. At this stage we extend the model with functionalities necessary for generating a packet at the IP layer, and delivering it to the SLACP receiver. This scenario is simulated by the IP layer opening the channel and sending a single packet to the SLACP sender and then closing the channel.

The third level of modeling is to include the flow control between IP and SLACP layers as well as between SLACP and MAC layers. We enhance the model from the previous stage by adding functionalities for queuing the frame at the SLACP sender and MAC buffers and for flow control mechanisms based on the buffer availability. The composite model is tested by setting the size of the SLACP sender buffer and MAC buffer to small values.

At the fourth level of modeling we incorporate ARQ-FEC error control schemes. We add the SACK mechanism, buffering and timer features to handle FEC coding. As it is not necessary to choose the actual FEC encoding in the model, we use a simple frame replication instead. The FEC recovery scheme follows the description in section 2.

In order to examine the working of the error recovery scheme we send several data frames to the receiver of which only the first data frame is lost. This helps to isolate the errors if they are present. Once we find that the model is able to recover from a single frame loss, we can simulate the loss of several frames at the MAC layer and check that the model works for different error scenarios.

At this point we have essentially built the SLACP model with all the features of the protocol included in it. As we build the model cumulatively, and at each stage make use of the functionalities at the earlier levels as well in extending the model, we have confidence in the basic correctness of the design.

The next phase deals with testing and simulation. We exercise the protocol model using the test scenarios and observe its working. Having gained confidence about its working, we begin the validation phase. The SDL model of SLACP is about 100 pages long and it took about 3 man months to develop and validate the protocol.

5 Validation of SLACP Using SDL

Telelogic Tau 4.4 SDT tool is used for validating SLACP. We formulate the following validation scenarios which represent the protocol operations. The scenarios correspond to the different levels of modeling described earlier.

- Opening / Closing a channel
- Sending and Receiving a frame without frame loss
- Sending and Receiving a frame without frame loss and flow control
- Sending and Receiving frames with frame loss (using ARQ and FEC)
- Sending and Receiving frames with frame loss and flow control
- Resetting the channel

Each of the above test scenarios is validated using the automatic validation methods available in SDT. In automatic state space exploration SDT builds a reachability graph for the system model. The state space is the set of all the states of the system that can be reached from its initial state by systematically exploring all the transitions. A number of general properties of a protocol such as deadlock freedom, absence of unspecified signal reception and unreachable code can be verified by exploring the state graph. As exhaustive state space verification becomes infeasible with a large state space, alternatives such as bitstate exploration and random walk methods that explore a large fraction of the state space are used [8]. By choosing as small a value as possible for the sizes of data frame, SACK block and buffers, we can reduce the complexity of the state space to some extent.

The validation reports point to the errors encountered in the state exploration. This is especially valuable as it examines scenarios that are unlikely to be considered in the test suite. For example, consider the following scenario: the SLACP sender sends a frame and the frame is lost; the receiver asks for a retransmission and the sender retransmits it; this FEC encoded retransmitted frame gets delayed and arrives at the receiver at a later time; by that time the receiver might have sent the frames in the reassembly queue to the higher layer assuming that this frame is lost. Such a delayed frame may create problems if we do not have the sanity check at the receiver to see whether the frame received is within the receiver window. It is difficult to create the above situation manually but the exhaustive state space exploration can readily set it up. Automatic validation also helps to exercise the scoreboard and the various queues.

The SDT option to display the execution trace that leads to an error state is useful to detect errors and fix them. The flexibility of the tool to show the path of error both in the SDL graphs and in the MSC supports debugging.

Automatic validation helps to ensure that the special frames such as nothing_to_send and rxmtpkt_loss perform as intended in the protocol design.

6 SLACP-SDL Model Versus SLACP Implementation

After validating the SDL model, we implemented the SLACP protocol in C for the Linux operating system. The C implementation closely followed the SDL model. However, there are some differences between the SDL model and its implementation in addition to those differences discussed earlier (FEC encoding/decoding not modeled, only a single SLACP channel modeled). In the following we highlight some of these more or less subtle differences.

In the SDL graphs, explicit flow control signals exist between the SLACP and IP layer. In the Linux implementation these signals are implicit as the SLACP protocol engine accepts a packet from the IP underbelly interface only when buffer space is available at the SLACP layer, effectively implementing the flow control between the layers. In addition, the flow control details between the SLACP and MAC layer in the Linux implementation depend on the link

technology and device in use. Therefore, an abstract model of the SLACP-to-MAC flow control can be used in the SDL graphs.

In the SLACP implementation the SACK block in the ACK frame is implemented as a bit map. Each bit in the SACK block together with the cumulative acknowledgment sequence number of the ACK frame represents whether a frame has been successfully received (bit set to 1) or not (hole, bit set to 0). Since bitmaps and their operations are not easy to represent in SDL, each bit in the SACK block is actually a structure with a field for the corresponding sequence number and a boolean value indicating whether the frame was successfully received or not. The code for iterating through such a structure to check whether the SACK block is empty is clearly not as trivial as the simple check for zero bit map in C. Therefore, an additional field indicating whether a SACK block is empty was added in the SDL model.

In the automatic validation of SLACP, the symbol coverage was 94.99 %. If the symbol coverage is not 100%, the validation cannot be considered finished without a clear understanding of the missing coverage. We analyzed the uncovered symbols in the SDL graphs. Most of them were either invalid end states or conditions that do not occur. In the Linux implementation there are even more invalid states as a real protocol implementation has to test for many error conditions that do not occur except in some very exceptional conditions such as in case of broken or otherwise misbehaving implementation of the peer. Achieving 100 % symbol coverage with such sanity checks included into the SDL model would require that any such misbehavior is implemented and enforced in the peer protocol engine. Therefore, we decided to exclude most of these checks in the SDL model.

7 Concluding Remarks and Future Work

In this paper we have described the design and development of SLACP protocol using SDL modeling. A validation oriented approach to protocol design was employed to develop the protocol in a hierarchical stepwise refinement manner. This helped to control the complexity of the design and to better understand the interaction of the protocol components. The SDL model was used as a basis for the implementation of the protocol in Linux.

We found that the SDL model developed in this approach was convenient to adapt by incorporating implementation specific details such as FEC encoding scheme using Reed-Solomon codes. With hindsight we observe that this ability to adapt the SDL model to actual implementation is quite valuable as the design is not encumbered with implementation specific aspects. We believe that the approach to SLACP design can be used for general protocol design.

The SLACP protocol validation has no doubt increased the quality of the produced protocol implementation. However, we still cannot say that the protocol model has been formally verified: a 100% guarantee that the protocol works as intended in all possible cases has not been yet achieved. Partly this is because the SDT validator tool does not really support all possible kinds of behavioral

properties that we would like to verify from the system. For example a property such as "*a channel open in IP-layer will in all cases either lead to channel establishment or to channel open failure signal*" is simply not expressible within SDT validator. Another thing that limits the reliability factor of our validation is the fact that our protocol model is huge in terms of size of reachability graph and since the SDT validator does not really have any sophisticated state space reduction algorithms, we have to rely on approximate methods such as bit state hashing. For these reasons we consider starting a follow-up project where we are planning to do a full fledged formal verification of SLACP with Spin model checker [8]. Partly this will be a challenge to the Spin tool itself: it will be interesting to see how a state of the art verification tool can treat a complex real world protocol.

References

1. Transmission Control Protocol. RFC 793, Internet Society, Sep 1981.
2. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.
3. *Recommendation Z.100 - CCITT Specification and Description Language*. ITU, 1993.
4. *Telelogic Tau 4.4 Manual* . Telelogic, 2002.
5. Enhancing TCP Performance Over Satellite Networks - A Link Aware Approach. Technical Report C-2004-50, University of Helsinki, Department of Computer Science, August 2004.
6. Improving TCP Performance Over Wireless WANs using TCP/IP-Friendly Link. In *1st International Conference on E-Business and Telecommunication Networks (ICETE)*, August 2004.
7. *Data Networks: 2nd edition*. Prentice Hall, 1992.
8. *The Spin Model Checker*. Addison Wesley, 2003.