How Stop and Wait Protocols Can Fail over the Internet

Jonathan Billington and Guy Edward Gallasch

Computer Systems Engineering Centre, University of South Australia, Mawson Lakes Campus, SA 5095, Australia,
jonathan.billington@unisa.edu.au, guy.gallasch@postgrads.unisa.edu.au

Abstract. The correct operation of computer protocols is essential to the smooth operation of the distributed systems that facilitate our global economy. Formal techniques provide our best chance to ensure that protocol designs are free from errors. This invited paper revisits the class of Stop-and-Wait protocols that incorporate retransmission strategies to recover from transmission errors. This is motivated by the fact that their basic mechanisms are important for practical protocols such as the Internet's Transmission Control Protocol (TCP). Stop-and-Wait protocols have been shown to operate correctly over media that may lose packets, however, there has been little discussion regarding the operation of these protocols over media that can re-order packets. The paper presents an investigation of these protocols operating over a medium, such as that provided by the Internet Protocol, that does allow reordering of data. Coloured Petri Nets are used to build a model of a Stop-and-Wait Protocol parameterized by its maximum sequence number and the maximum value of the retransmission counter. The model is analysed using a combination of hand proofs and automatic techniques. We identify four problems. We firstly prove the counter intuitive property for a Stop-and-Wait protocol that the number of packets that are stored in the network can grow without bound. This is true for any positive values of the maximum sequence number and the maximum number of retransmissions. We further show that loss of packets is possible and that duplicates can be accepted as new packets by the receiver. These first three properties hold even though the sender and receiver perceive that the protocol is operating correctly. The final problem is that the protocol does not satisfy the Stop-and-Wait service where sends and receives alternate. Finally, we provide a discussion of the relevance of these results to the Transmission Control Protocol.

Keywords: Stop and Wait Protocols, TCP, State space methods, Coloured Petri Nets, Protocol Analysis and Verification.

1 Introduction

Because of the importance of network protocols [27, 31] to the world's economy, it is vital that they do not fail [16], especially in safety critical or financially

H. König, M. Heiner, and A. Wolisz (Eds.): FORTE 2003, LNCS 2767, pp. 209–223, 2003.

sensitive situations. This is the main tenet of this conference, which uses *formal techniques* [13, 19, 35] to prove that protocols operate correctly and satisfy the requirements stated in the their service specifications [11, 17].

In this paper, we focus on the modelling and analysis of Stop-and-Wait protocols (SWP) [27, 31]. Stop-and-Wait is an elementary form of flow control [27, 31] between a sender and a receiver. The sender stops after transmitting a message and waits until it receives an acknowledgement indicating that the receiver is ready to receive the next message. Stop-and-Wait Protocols (SWP) often operate over noisy channels and combine flow control with error recovery [31] using a timeout and retransmission scheme, known as Automatic Repeat ReQuest (ARQ). In this case, a sequence number is appended to each message to ensure that duplicate messages from retransmissions are not accepted as new messages. A checksum [31] is also included to detect transmission errors.

Sliding Window protocols [31] improve the efficiency of SWPs allowing many messages (rather than one) to be sent before requiring an acknowledgement. Cumulative acknowledgements and more sophisticated error retransmission schemes (such as Go-Back-N or Selective Reject) [27] can also improve efficiency. These schemes are therefore used in many practical protocols such as the Internet's Transmission Control Protocol (TCP). However, the underlying principles of ARQ used in SWPs, are still relevant to protocol's such as TCP, where we note that a window size of 1 corresponds to a Stop-and-Wait protocol.

It is well known [31] that for sliding window protocols to work properly in detecting and discarding duplicates, the sequence number space needs to be one greater than the number of unacknowledged messages. In the case of Stop-and-Wait protocols which have just one outstanding unacknowledged message, the sequence number space can be just two numbers, usually {0,1}. Stop-and-Wait protocols realised with a one bit sequence number are known as *alternating bit protocols*, because the sequence number alternates between 0 and 1 as data is sent.

The Alternating Bit Protocol [22] was originally designed to provide reliable full duplex communication over unreliable half duplex physical lines. A large number of papers, articles and books [4, 5, 15, 18, 22, 26, 28–31] use the ABP to illustrate protocol mechanisms or to demonstrate the use of a formal description technique. This is the case in [28] where the ABP is used as an example to illustrate a new Timed Rewriting Logic (TRL) for capturing the static and dynamic aspects of SDL (Specification and Description Language) [1]. Another example of this is in [29, 30] where the original ABP is formally modelled and analysed using Temporal Petri nets. The Abracadabra Protocol [34] uses sequence numbers realised by an Alternating Bit, Retransmissions on timeout, Acknowledgements, Connection And Disconnection (ABRACAD), and is one of a graded set of examples used to provide guidelines for the application of three standardised formal description techniques, namely Estelle [10], LOTOS (Language Of Temporal Ordering Specifications) [9] and SDL [1]. Billington et al [18] use a variant of the ABP [12], proposed for the link layer of the ISDN user/network interface, to demonstrate a software tool and its ability to find

errors in the protocol. Reisig [26] develops the ABP in a series of steps as part of a case study on acknowledged messages, developed incrementally using simple Petri net models.

Some of the above papers demonstrate that the ABP will work as expected over FIFO (First-In First-Out) channels that may also include loss. It appears, however, that the situation in which messages may be re-ordered by the medium has not been considered. Because TCP [32] operates over a network that does not guarantee in sequence delivery and may also lose messages [31], it is useful to investigate this situation for SWPs.

We have found that, under these conditions, the SWP does not operate as expected in that: the number of messages or acknowledgements in the underlying medium can grow without bound; the receiving entity can unknowingly accept duplicates as new messages; undetected message loss can occur; and the SWP does not satisfy its service of alternating sends and receives.

The rest of the paper is organised as follows. Section 2 presents and explains our Coloured Petri Net (CPN) [21, 23] model of the SWP, assuming the reader has some familiarity with CPNs. Section 3 develops theorems and their proofs regarding the properties of the model, illustrating the above problems. A discussion of the impact of these results on the Transmission Control Protocol (TCP) is given in Section 4. Finally, areas of future work and some concluding remarks are presented in Section 5.

2 Stop-and-Wait Protocol CPN Model

We model a Stop-and-Wait protocol that includes error recovery using retransmissions operating over a lossy and re-ordering medium using Coloured Petri Nets (CPNs) [21**?**]. The CPN model is given in Fig. 1 and 2. Figure 1 provides the graphical representation of the system, while Fig. 2 defines all the constants, sets and functions required and declares the types of the variables used in the inscriptions associated with Fig. 1. The software tool, Design/CPN [25], was used to construct the model. CPN ML [14], a variant of Standard ML (SML/NJ) [24] is used for the net inscriptions in Fig. 1 and the declarations in Fig. 2.

The model in Fig. 1 comprises three parts: the *sender* (on the left), the *receiver* (on the right) and an underlying bidirectional communication medium, *network*, in the middle.

Sender

The sender consists of four places, four transitions and their interconnecting arcs. The places, sender_ready and wait_ack, represent the two states of the sender and are typed by the colour set, Sender, representing a single sender. The place, sender_ready, has an initial marking of one s token, indicating that the Sender is initially in the ready state. The seq_no place stores the sender sequence number, which is either the number of the message just sent (an unacknowledged message), or if acknowledged, the number of the next message to be sent. It is typed

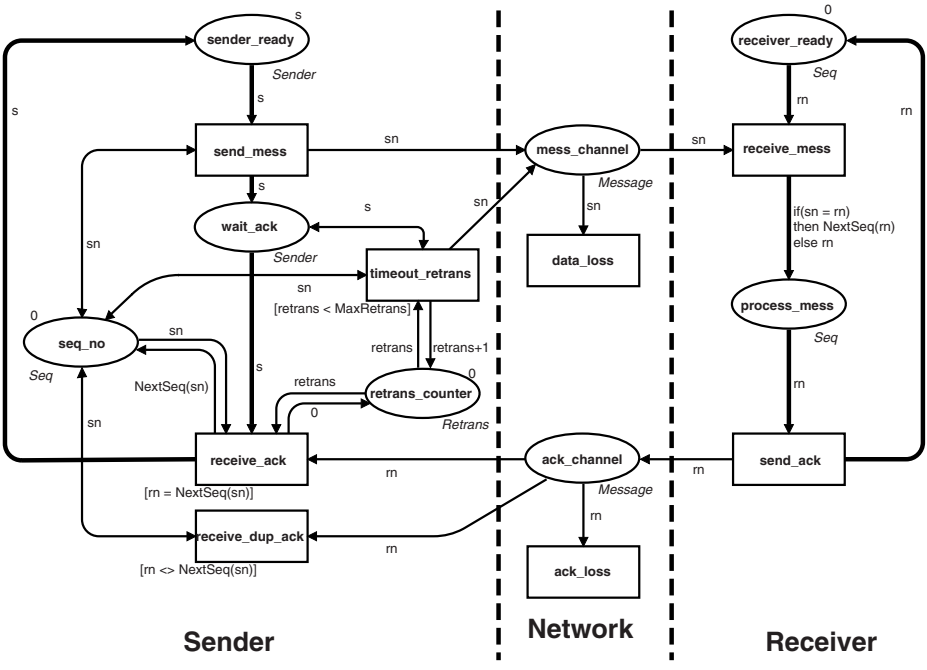

Fig. 1. CPN of the Stop-and-Wait Protocol.

```
(* --- Global Declaration Node --- *)

val MaxRetrans = 1;
val MaxSeqNo = 1;

color Sender = with s;
color Seq = int with 0..MaxSeqNo;
color Retrans = int with 0..MaxRetrans;
color Message = Seq;

var sn,rn   : Seq;
var retrans : Retrans;

fun NextSeq(n) = if(n = MaxSeqNo) then 0 else n+1;
```

Fig. 2. Declarations of the Stop-and-Wait Protocol CPN.

by the colour set Seq (sequence number) and has an initial marking of a single 0 token, indicating that the first message to be sent will have sequence number 0. The number of retransmissions is recorded in place retrans_counter typed by the colour set Retrans, and is initially 0.

Transition send_mess models the sending of a message to the receiver. Message content is not represented as it has no bearing on the operation of the protocol.

The same is true for the addresses of sender and receiver, as we only have one of each in this model. Consequently, a message (or an acknowledgement) is modelled as a sequence number. When the sender is ready send_mess may occur. It writes its sequence number (as the message) to the message channel and changes state to waiting for an acknowledgement.

The timeout_retrans transition models the expiration of the retransmission timer and the retransmission of the currently unacknowledged message. This transition can only occur if the sender is awaiting an acknowledgement and there have been less than MaxRetrans retransmissions of this message (see the guard in square brackets). When timeout_retrans occurs, the retransmission counter is incremented by 1 and the retransmitted message is placed into the message channel.

Transition receive_ack models the receipt of expected acknowledgements from the receiver, i.e. those that acknowledge the currently outstanding message. Duplicate acknowledgements are received and discarded by transition receive_dup_ack. These may result from acknowledged data retransmissions, where delay rather than loss was the cause of the retransmission.

The function NextSeq is used to increment the sequence number modulo (MaxSeqNo + 1), as shown in Fig. 2. An occurrence of receive_ack will remove the acknowledgement from the channel, return the sender to the ready state, reset the retransmission counter to 0, and increment the sequence number stored in seq_no using modulo arithmetic. An occurrence of receive_dup_ack only removes the acknowledgement from the channel.

Receiver

The places receiver_ready and process_mess model the states of the receiver and are typed by the colour set Seq. A sequence number token present on one of these places indicates that the receiver is in that state. Initially the receiver is expecting a message with sequence number 0. Transition receive_mess models the receipt of a message from the sender. The inscription on the arc from receive_mess to process_mess compares the sequence number of the message (sn) with the sequence number expected by the receiver (rn). If they match, then the message is the one expected (and is passed onto the user, not modelled) and the sequence number is incremented modulo (MaxSeqNo + 1) by the NextSeq function. If they don't match, a duplicate is detected (and discarded, not modelled) and the sequence number remains unchanged. Transition send_ack occurs when the receiver has finished processing the message, indicating that enough buffer space is available to receive another message. This transition sends an acknowledgement containing the next sequence number expected by the receiver, and returns the receiver to the ready state. Sending an acknowledgement when a duplicate message is received is necessary because if an acknowledgement of a (new) message is lost and subsequent retransmissions of the same message are not acknowledged, the system will fail to progress as no acknowledgement will ever be delivered to the sender.

Underlying Medium

The underlying communication medium is modelled by one place and one transition for each direction of communication. Both channel places are typed by the colour set Message (a sequence number, see Fig. 2) and are initially empty. This models the overtaking behaviour of the communication medium. The two transitions data_loss and ack_loss model the loss of messages and acknowledgements respectively. This corresponds to either loss in the network (due to congestion and buffer overflow in a router), or to discarding messages (acknowledgements) due to checksum failures.

3 CPN Model Analysis

With the SWP it is usual to place an upper bound (MaxRetrans) on the number of retransmissions that are allowed per message. When this limit is reached, the communication medium is considered to be down. In practice, an indication is given to a management entity that invokes a procedure to deal with the fault. This interaction (and procedure) is not modelled as it is not part of the SWP. In our model, the protocol will just terminate in a state where the retransmission counter has reached its maximum value (MaxRetrans). This is an expected terminal state, indicating that the network is down, and that the last message sent may have been lost.

In this paper we are not concerned with terminal states. Instead we focus on properties that are quintessential for correct operation of the SWP. The first concerns bounds on the channels, the second that duplicates are not accepted as new messages, the third that messages are not lost unknowingly and the fourth is that the protocol conforms to the Stop-and-Wait service of alternating sends and (correct) receives.

When the SWP operates over a lossy FIFO channel, we can show [8] that the bound on both channel places is given by 2MaxRetrans +1. We can also show [8] that the Stop-and-Wait property holds for FIFO channels, and by implication, no duplicates are received as new messages and no messages are lost. For lossy FIFO channels, the Stop-and-Wait property still holds, except when the maximum retransmission limit is reached and thus message loss is not recovered. In this case there is no corresponding receive for the last send. We also conjecture that in this case, no duplicates are accepted as new messages, and only one message can be lost, the last one sent before the SWP terminates.

We conclude that the SWP operates as expected over lossy FIFO channels. If we allow messages in the channel to be re-ordered, will these properties still be satisfied? The following two subsections show that they are not.

3.1 Unbounded Channels

It turns out that with a reordering medium the number of messages in the communication channel can grow without bound. We formalise this property using our CPN model in the following theorem for the message channel.

Theorem 1. *For the Stop-and-Wait CPN of Figs. 1 and 2 where* MaxRetrans $\geq$ *1 and* MaxSeqNo $\geq$ *1, the message channel (place* mess_channel*) is unbounded.*

Proof. To prove this theorem, we show that a cycle of transition occurrences exists where the total effect of each cycle is to increase the number of messages in the message channel by one, and that this cycle can be repeated indefinitely.

Consider our CPN model from fig. 1 with declarations as shown in fig. 2 with MaxRetrans ≥ 1 and MaxSeqNo ≥ 1. From the initial marking, M_0, the following sequence of transition occurrences, σ_0 = send_mess <sn=0>, receive_mess <rn=0, sn=0>, send_ack <rn=1>, timeout_retrans <retrans=0, sn=0>, receive_ack <retrans=1, rn=1, sn=0> can occur resulting in marking M_5, ($M_0 \xrightarrow{\sigma_0} M_5$), where we have written the binding of variables for each transition occurrence inside angular brackets for each transition. M_5 is given by

M_5(sender_ready)=1‘s	M_5(retrans_counter)=1‘0
M_5(seq_no)=1‘1	M_5(receiver_ready)=1‘1
M_5(ack_channel)=$\emptyset$	M_5(wait_ack)=$\emptyset$
M_5(process_mess)=$\emptyset$	M_5(mess_channel)=1‘0

M_0 and M_5 are the same, except that the sequence numbers stored at the sender and receiver have been incremented by one, and there is an additional message (‘0’) left in the message channel.

We now consider a further occurrence of the same transition sequence, only this time we require that the values of the sequence number variables are incremented modulo (MaxSeqNo + 1). To illustrate this we firstly assume MaxSeqNo = 1, so that the sequence of transition occurrences is given by σ_1 = send_mess <sn=1>, receive_mess <rn=1, sn=1>, send_ack <rn=0>, timeout_retrans <retrans=0, sn=1>, receive_ack <retrans=1, rn=0, sn=1>. σ_1 can occur from M_5, resulting in a marking M_{10}, with

M_{10}(sender_ready)=1‘s	M_{10}(retrans_counter)=1‘0
M_{10}(seq_no)=1‘0	M_{10}(receiver_ready)=1‘0
M_{10}(ack_channel)=$\emptyset$	M_{10}(wait_ack)=$\emptyset$
M_{10}(process_mess)=$\emptyset$	M_{10}(mess_channel)=1‘0 + 1‘1

We note that $M_{10} = M_0 + \{((\text{mess\_channel}, 0), 1), ((\text{mess\_channel}, 1), 1)\}$, so that M_{10} is a covering marking for M_0. Due to the transition rule of CPNs, additional tokens in a place will not disable any transitions that were previously enabled. Hence the occurrence sequence $\sigma_0\sigma_1$ can repeat indefinitely, each time adding a ‘0’ and ‘1’ mess_channel, so that the number of tokens increases without limit and thus we have proved Theorem 1 for MaxSeqNo = 1.

Generalising for MaxSeqNo ≥ 1, we have MaxSeqNo+1 sequence numbers and thus require $\sigma_0, \sigma_1, \ldots, \sigma_{\text{MaxSeqNo}}$, defined in the same way as σ_0 and σ_1 above. For $0 \leq j \leq$ MaxSeqNo, σ_j = send_mess <sn=j>, receive_mess <rn= sn=j>, send_ack <rn=(j+1)mod(MaxSeqNo + 1)>, timeout_retrans <retrans=0, sn=j>, receive_ack <retrans=1, rn=(j+1)mod(MaxSeqNo + 1), sn=j>.

The occurrence of $\sigma_0\sigma_1 \ldots \sigma_{\text{MaxSeqNo}}$ in marking M_0 leads to a marking M_m, where $m =$ 5MaxSeqNo and

M_m(sender_ready)=1‘s M_m(retrans_counter)=1‘0
M_m(seq_no)=1‘0 M_m(receiver_ready)=1‘0
M_m(ack_channel)=$\emptyset$ M_m(wait_ack)=$\emptyset$
M_m(process_mess)=$\emptyset$
M_m(mess_channel)=1‘0 + 1‘1 + ... + 1‘MaxSeqNo

M_m covers marking M_0 so that $\sigma_0\sigma_1 \ldots \sigma_{\mathsf{MaxSeqNo}}$ can repeat indefinitely from marking M_0, resulting in MaxSeqNo additional messages in the message channel for each repetition. Thus the message channel is unbounded. □

A similar theorem holds for the acknowledgement channel.

Theorem 2. *For the Stop-and-Wait CPN of Figs. 1 and 2 where MaxRetrans $\geq$ 1 and MaxSeqNo $\geq$ 1, the acknowledgement channel (place ack_channel) is unbounded.*

Proof. The proof of this theorem is similar to that of Theorem 1, hence we just provide a sketch. A suitable transition sequence in this case is send_mess, receive_mess, send_ack, timeout_retrans, receive_ack, receive_mess and send_ack. Transition occurrence sequences $\sigma_0, \sigma_1, \ldots \sigma_{\mathsf{MaxSeqNo}}$ are defined in a similar way. The occurrence sequence $\sigma_0, \sigma_1, \ldots \sigma_{\mathsf{MaxSeqNo}}$ can be repeated indefinitely from M_0, resulting in MaxSeqNo additional acknowledgements in the acknowledgement channel for each repetition. □

3.2 Message Loss and Incorrect Acceptance of Duplicates

When the Stop-and-Wait protocol operates as required, one message will be received correctly at the receiver for every original message sent by the sender. It turns out that this is not the case for the Stop-and-Wait protocol operating over a medium that reorders messages. This demonstrates that the protocol does not satisfy the Stop-and-Wait service. Further we can show that sequences of sends and receives exist where there are more receives than sends, indicating that duplicates are accepted. Finally we can also demonstrate that there are sequences in which there are more sends than receives, indicating that messages can be lost. We summarise these results in the following three theorems, and prove them together in the one proof.

Theorem 3. *The Stop-and-Wait CPN of Figs. 1 and 2 where MaxRetrans $\geq$ 1 and MaxSeqNo $\geq$ 1, does not satisfy the Stop-and-Wait service.*

Theorem 4. *For the Stop-and-Wait CPN of Figs. 1 and 2 where MaxRetrans $\geq$ 1 and MaxSeqNo $\geq$ 1, the receiver may incorrectly accept duplicate messages as new messages.*

Theorem 5. *For the Stop-and-Wait CPN of Figs. 1 and 2 where MaxRetrans $\geq$ 1 and MaxSeqNo $\geq$ 1, messages can be lost without the sender or receiver being aware of it.*

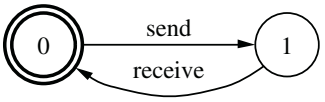

Fig. 3. FSA for the SWP service.

Proof. We use language analysis [7, 20] to prove the above theorems. The first step is to define the *service* [17] provided by the Stop-and-Wait protocol. The service defines a set of *service primitives* and their global sequences as observed by the SWP users. Service primitives are abstract representations of the interactions between the service provider and a service user.

For the SWP service, we define two primitives: a *send* at the sender entity interface; and a *receive* at the receiver entity interface. We can then define the *service language* as 0 or more repetitions of the sequence (send, receive). This can be represented by the regular expression $(send\ receive)^*$ or by the Finite State Automaton (FSA) shown in Fig. 3.

The next step is to generate the *protocol language* from the *protocol specification*. The protocol language just contains service primitive events. This allows direct comparison of the service and protocol languages. Our protocol specification is the CPN model of Figs. 1 and 2. In this CPN we can consider that the send primitive occurs when the send_mess transition occurs and similarly that the receive primitive occurs when receive_mess occurs where the bindings of sn and rn are the same. (Otherwise the occurrence of receive_mess represents the discarding of duplicates, which does not correspond to a receive primitive.)

The protocol language can be obtained from the CPN's reachability graph by treating it as a FSA. All non-service primitive transitions (i.e. those associated with sending and receiving acknowledgements, or with loss or retransmission) are replaced by empty (ϵ) transitions and the resulting FSA minimised [6] to produce the minimum deterministic FSA. This FSA represents all possible sequences of service primitives, generated from the protocol, and is thus the protocol language. We use the suite of tools available in the FSM package [2] for FSA minimisation and comparison.

Due to the unbounded communication channel in our original model shown in Fig. 1, the resulting reachability graph is infinite. However, to prove our theorems, we only need to demonstrate that it is possible for the system to malfunction. We therefore limit the capacity of the communication channels to two. The rationale behind this is that if the protocol operates incorrectly with a channel capacity of two messages, the same incorrect behaviour will also be present in a channel with capacity greater than two. Capacities of 0 and 1 are not appropriate, as a capacity of 0 results in no communication and a capacity of 1 prohibits overtaking. Thus a capacity of size two is the minimum needed to show interesting behaviour.

To obtain the smallest reachability graph of interest, we also set MaxRetrans $= 1$ and MaxSeqNo $= 1$. We argue that if incorrect behaviour is evident when MaxRetrans $= 1$ then the same behaviour can occur for MaxRetrans ≥ 1 and similarly for MaxSeqNo.

Channel capacity has been implemented as follows. The initial marking of the mess_channel and ack_channel places has been modified so that each place contains a certain number of empty tokens, in this case two each, representing empty buffers. Each time a message (data or acknowledgement) is placed in the channel, an empty token is removed from the corresponding channel place, and whenever a message is removed from the channel it is replaced by an empty token.

The reachability graph of this CPN was generated using Design/CPN [25], with the configuration as shown in the CPN in Figs. 1 and 2 (noting the modifications above) and with the following additions:

- Two Empty tokens in mess_channel (Capacity of 2)
- Two Empty tokens in ack_channel (Capacity of 2)
- No message loss on either channel

The reachability graph contains 410 nodes and 848 arcs. After interpreting this as a FSA, the FSM package was used to obtain the equivalent minimum deterministic FSA as shown in Fig. 4. We have replaced send_mess with s and receive_mess with r in the figure due to size constraints.

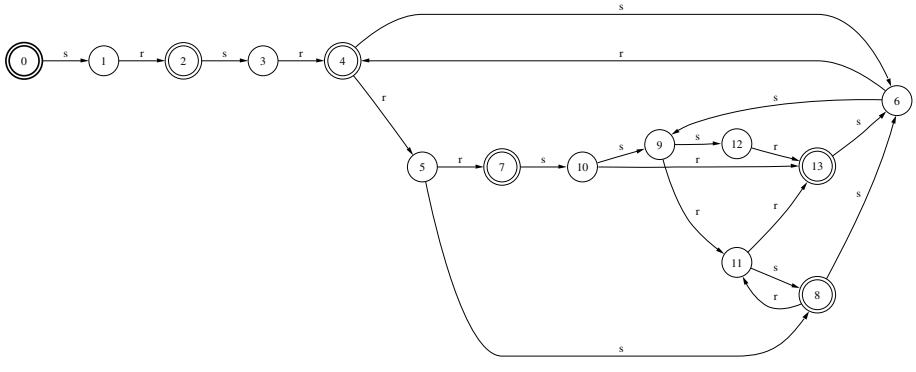

Fig. 4. FSA showing erroneous sequences of send and receive primitives.

This FSA shows that the SWP operating over a medium that reorders messages does not satisfy its service. There are incorrect sequences of send and receive primitives, indicating that the receiver can mistakenly accept duplicate messages as new messages. For example, the cycle $7 \xrightarrow{s} 10 \xrightarrow{r} 13 \xrightarrow{s} 6 \xrightarrow{r} 4 \xrightarrow{r} 5 \xrightarrow{r} 7$ shows that it is possible for the system to enter a loop where the receiver accepts four messages as legitimate messages for every two sent by the sender. A second sequence $13 \xrightarrow{s} 6 \xrightarrow{s} 9 \xrightarrow{s} 12 \xrightarrow{r} 13$ demonstrates that another cycle exists where for every 3 messages sent, only one is received, demonstrating message loss even though there is no loss in the medium! It is interesting to note that problems with acceptance of incorrect messages do not occur until the sequence numbers wrap, i.e. at node 4 in Fig. 4.

We also generated results when the channel was lossy (in addition to reordering). The reachability graph contained 624 nodes and 2484 arcs. The reduced FSA showing the protocol language for this configuration contains 29 nodes and 47 arcs. There are many incorrect sequences in this language also. A figure has not been included due to size limitations. □

4 Relevance to Practical Protocols

In order to understand the relevance of these results to the real world, let us consider the error recovery and flow control strategies implemented in TCP [32]. TCP uses retransmission on timeout to recover from packet loss and a sliding window mechanism for flow control, which includes dynamic window changes. TCP operates over IP (the Internet Protocol), which allows packets (known as segments) to be dropped or reordered. The correctness of TCP's data transfer procedures can thus be related to the correctness of the Stop-and-Wait protocol operating over a medium that allows reordering.

TCP uses a 32 bit sequence number, giving $2^{32} = 4294967296$ sequence numbers. Each sequence number is associated with 1 byte of data. The problems associated with the Stop-and-Wait protocol only arise after sequence numbers wrap, so that old duplicates can cause desynchronisation of the acknowledgement mechanism resulting in loss or acceptance of duplicates. This will only happen in TCP after 4Gbytes of data have been transmitted, and old duplicates still remain in the network.

The threat posed by old duplicates was recognised by the designers of the Internet. They introduced the concept of a life-time for a packet in the IP layer, known as *time-to-live*. (This is implemented as a 'hop count' in practice.) The idea is that duplicate packets left floating around a network will be discarded once their time-to-live expires.

RFC (Request for Comment) 793 [32], the protocol specification for TCP maintained by the Internet Engineering Task Force (IETF) [3], states that the maximum lifetime for a segment of data (MSL) is two minutes. Thus there will not be a problem with duplicate packets if they are destroyed before sequence numbers can wrap. Every byte is given a sequence number, thus for a transmission rate of 100 megabits/sec (12.5 megabytes/sec) we see that the sequence numbers will wrap after $2^{32}/(1.25 * 10^7) \approx 5$ minutes and 45 seconds. This is getting close to the maximum packet lifetime, but should not pose a problem unless the hop count mechanism takes longer than 2 minutes to quash packets. With the introduction of Gigabit networks [31] the sequence numbers of TCP could wrap after only 34 seconds of data transfer at 1 gigabit/second. Although the maximum throughput of a network rarely approaches the theoretical maximum, it would not be unreasonable to assume that with a very large window size and very large data transfers, wrapping of sequence numbers would occur after about one minute, allowing for the possibility of old duplicates being in the channel at the same time as new segments with the same sequence numbers.

This is the condition necessary for loss and acceptance of the old duplicate (as a new segment) to occur. However, to get duplicates, there must be retransmissions caused by additional delay due to network congestion or lack of responsiveness in the receiver (e.g. an overloaded web server) which will reduce throughput. This delay, however, does not need to be very great to cause retransmissions, and hence the effect on throughput may not be significant. Another factor limiting throughput is TCP's window size and the round trip delay (RTD). In standard TCP implementations, the maximum window size is 2^{16} bytes, limiting the throughput to (2^{16}/RTD) bytes/sec. The speed of light propagation delay contribution to RTD will then provide a limit irrespective of the transmission speed. However, to allow users to take advantage of high-speed networks, RFC 1323 [33] proposes to increase the maximum window size to 2^{30} bytes or 1Gbyte, in which case the speed of light delays are no longer a limiting factor.

It is unlikely that duplicates are a problem for TCP with the current speed of networks, however these problems may become more probable if network speed were to increase by another order of magnitude, i.e. 10 gigabit/second. There are additional ramifications to be considered if incorrect acceptance of duplicates or loss of data becomes a problem. For safety critical applications operating over the Internet the consequences could be catastrophic.

There are a number of suggested ways in which this problem could be solved, or at least alleviated. RFC 1323 [33] specifies a number of TCP extensions for high performance. The extension for a larger window size has already been mentioned. Another extension is Protect Against Wrapped Sequence Numbers (PAWS) which proposes a solution to wrapping sequence numbers within a connection, by including a 32 bit time-stamp in every segment. Another solution involves extending the sequence number space, to 2^{64}, i.e. 64 bit sequence numbers. Even at 10 gigabit/second, a 64 bit sequence number field would take 470 years to wrap. The procedure of sequence numbering may also be reviewed, as currently every byte is given a sequence number. Providing a sequence number for every segment would extend the usefulness of the existing 32-bit sequence numbers.

How likely is it that unbounded growth of messages in the communication channels will actually occur? The unbounded growth is caused by retransmissions due to delayed acknowledgements. Given the variability of the round trip delay (due to the unpredictability of network congestion or overloaded servers) it is not uncommon for these delays to occur. This is countered to some extent by TCP measuring round trip delay and setting its retransmission timeout period accordingly. However, due to transients, unnecessary retransmissions will always occur. The unbounded growth, however, only occurs because the duplicates are not received by the receiver. This is highly unlikely. Also those that are delayed in the network will be expunged after their time-to-live limit has expired. Thus TCP already has mechanisms in place to prevent unbounded growth. TCP has also developed sophisticated techniques to cope with network congestion [31], so we don't see that our unboundedness result for re-ordering media will cause

major difficulties with protocols such as TCP. Nonetheless, as network speeds increase the problem will get worse, particularly if the time to live value is maintained at 2 minutes. In general we can say that the contribution to congestion over FIFO channels is well contained (determined by the maximum number of retransmissions allowed) whereas over channels that re-order messages, it requires other mechanisms to contain congestion.

5 Conclusions and Future Work

This paper is concerned with the class of Stop-and-Wait protocols (SWPs) that include retransmission on time-out procedures to recover from transmission errors or from dropped packets in a communication medium such as the Internet. These protocols were originally designed to operate over a physical wire where the sequence of packets is maintained. The recovery procedure requires packets to include a sequence number in order to detect duplicates. Duplicates are due to retransmissions that occur because packets (or their acknowledgements) are delayed or their acknowledgements are lost. It is well known that duplicates are successfully detected, so long as the sequence number space is greater than the number of packets sent without first receiving acknowledgement. For SWPs this allows the sequence numbers to be just two: {0,1} (the class of alternating bit protocols (ABPs)), but also allows a larger sequence number space to be used. It is not too difficult to show that ABPs work as expected over lossy FIFO channels and we have also shown [8] that the communication channels are bounded by one more than twice the maximum value of the retransmission counter.

Protocols operating over the Internet Protocol (such as TCP) have to contend not only with loss due to transmission errors and packets dropped at routers, but also with the possibility that the order of packets is not maintained. Since TCP can behave as a Stop-and-Wait protocol under certain conditions it is interesting to investigate the behaviour of SWPs over a reordering medium. To do this we constructed a Coloured Petri Net model of the SWP, parameterised by the maximum number of retransmissions and the maximum sequence number. We analysed this model with respect to four properties and proved that: the communication channels are unbounded; messages can be lost, although the sender believes they have been confirmed by the receiver; duplicates can be accepted as new messages by the receiver; and that the SWP does not satisfy its service of alternating sends and receives. The last three results depend on sequence numbers wrapping before any problems occur. The first is independent of sequence number wrap.

We discuss the relevance of these results to the 'real world' using TCP. We conclude that sequence number wrap is possible in Gigabit networks, particularly if the extended window size option is used. RFC 1323 discusses this problem and suggests a mechanism (PAWS) using 32 bit time stamps to reject old duplicates, which hopefully will eliminate the problems associated with sequence number wrap. The problem with unbounded channels is not serious, but could add to congestion problems as the speed of networks increases. Our discussion is at a

high-level and does not investigate in detail TCP's procedures for data transfer, nor the suggested PAWS scheme. This is a subject for future work.

Acknowledgement

We are indebted to our colleague, Mat Farrington, who pointed out the relevance of RFC 1323 to our investigations.

References

1. CCITT Recommendation Z.100 (2002). Functional Specification and Description Language (SDL).
2. FSM Library, AT&T Research Labs. `http://www.research.att.com/sw/tools/fsm/`.
3. Internet Engineering Task Force. `http://www.ietf.org`.
4. P. Aziz Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. *Information and Computation*, 127(2):91–101, June 1996.
5. Y. Afek and G.M. Brown. Self-Stabilization of the Alternating Bit Protocol. In *Proceedings of the 8th Symposium on Reliable Distributed Systems*, pages 80–83. IEEE Comput. Soc. Press, 1989.
6. W.A. Barrett and J.D. Couch. *Compiler Construction: Theory and Practice*. Science Research Associates, 1979.
7. J. Billington. Formal specification of protocols: Protocol Engineering. In *Encyclopedia of Microcomputers*, volume 7, pages pages 299–314. Marcel Dekker, New York, 1991.
8. J. Billington and G. E. Gallasch. An Investigation of the Properties of Stop-and-Wait Protocols over Channels which can Re-order messages. Technical Report 15, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Australia., 2003.
9. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Comput. Networks and ISDN Sys.*, 14(1):pages 25–59, 1987.
10. S. Budkowski and P. Dembinski. An Introduction to Estelle: A Specification Language for Distributed Systems. *Comput. Networks and ISDN Sys.*, 14(1):pages 3–23, 1987.
11. C. A. Vissers and L. Logrippo. The Importance of the Service Concept in the Design of Data Communications Protocols. In *Protocol Specification, Testing and Verification, V*, pages pages 3–17. North Holland, Amsterdam, 1986.
12. CCITT. ISDN user-network interface data link layer specification. Technical report, Draft Recommendation Q.921, Working Party XI/6, Issue 7, Jan. 1984.
13. E.M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.
14. CPN ML: An Extension of Standard ML. `http://www.daimi.au.dk/designCPN/sml/cpnml.html`.
15. M. Diaz. Modelling and Analysis of Communication and Co-operation Protocols Using Petri Net Based Models. In *Protocol Specification, Testing and Verification.* North-Holland, 1982.
16. G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International Editions, 1991.

17. ITU-T. *Recommendation X.210, Information Technology - Open Systems Interconnection - Basic Reference Model: Conventions for the Definition of OSI Services*. Nov. 1993.
18. J. Billington, G.R. Wheeler and M.C. Wilbur-Ham. PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. In *IEEE Transactions on Software Engineering*, volume 14, pages 301–316. IEEE Press, 1988.
19. J. Billington, M. Diaz and G. Rozenberg (Eds.). *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
20. J. Billington, M.C. Wilbur-Ham and M.Y. Bearman. Automated Protocol Verification. In *Protocol Specification, Testing and Verification, V*, pages 59–70. North Holland, Amsterdam, 1986.
21. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Springer-Verlag, 1992.
22. K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *Communications of the ACM*, 12(5):260–261, May 1969.
23. L.M. Kristensen, S. Christensen and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
24. Standard ML of New Jersey. `http://cm.bell-labs.com/cm/cs/what/smlnj/`.
25. Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.
26. W. Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets, Volume 4*. Springer-Verlag, 1998.
27. W. Stallings. *Data and Computer Communications*. Prentice Hall, 6th edition, 2000.
28. L.J. Steggles and P. Kosiuczenko. A Timed Rewriting Logic Semantics for SDL: a case study of the Alternating Bit Protocol. *Electronic Notes in Theoretical Computer Science*, 15, 1998.
29. I. Suzuki. Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets. *IEEE Transactions on Software Engineering*, 16(11):1273–1281, 1990.
30. I. Suzuki. Specification and Verification of the Alternating Bit Protocol by Temporal Petri Nets. In *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*. IEEE Press, 1990.
31. A. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.
32. The Internet Engineering Task Force. Transmission Control Protocol. RFC 793, 1981.
33. The Internet Engineering Task Force. TCP Extensions for High Performance. RFC 1323, 1992.
34. K. J. Turner (Ed.). *Using Formal Description Techniques: An Introduction to Estelle, Lotos and SDL*. John Wiley & Sons, 1993.
35. J. M. Wing. A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9):8–24, September 1990.