Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 133 (2005) 255–273
www.elsevier.com/locate/entcs

Specifying and Verifying Communications Protocols using Mixed Intuitionistic Linear Logic

David Sinclair[1]

School of Computing
Dublin City University
Glasnevin, Dublin 9, Ireland

James Power[2]

Department of Computer Science
National University of Ireland, Maynooth
Maynooth, Co. Kildare, Ireland

Abstract

In this paper we present a technique for specifying and verifying communications protocols and demonstrate this approach by specifying and verifying two of the fundamental communications protocols, namely TCP and IP, which form the basis of many distributed systems. The logical formalism used is Mixed Intuitionistic Linear Logic in order to use both commutative and non-commutative operators to model the concurrent and sequential processes in these protocols. Key properties of both protocols are proved.

Keywords: complex systems, formal methods, mixed intuitionistic linear logic

1 Introduction

This paper presents an approach for specifying and verifyng communications protocols. This approach will be used to specify and verify the Internet Protocol (IP)[7] and elements of the Transmission Control Protocol (TCP)[8].

[1] Email: David.Sinclair@computing.dcu.ie

[2] Email: James.Power@may.ie

doi:10.1016/j.entcs.2004.08.068

The approach is based on mixed intuitionistic linear logic and describes how this logic can be used to prove some key properties of both protocols. We have previously presented a specification of IP in [10] using commutative linear logic. In this paper we extend this specification considerably to include the specification and verification of TCP. TCP, like many comunications protocols and distributed systems includes both sequential and concurrent processes. Specifying and verifying such systems with the commutative operators of linear logic is difficult. Linear logic is particularly suited to the description of state-based systems since it keeps track of the resources used in each deduction step. Mixed intuitionistic linear logic is a variant of linear logic that contains both commutative and non-commutative operators, and as such is useful where the order of the consumption of resources must be specified. The non-commutative operators of mixed intuitionistic linear logic are ideally suited to specifying systems with both sequential and concurrent processes. The main contribution of this research is to demonstrate how mixed intuitionistic linear logic can be used to specify and verify these types of distributed systems.

In the following sections we briefly describe IP and TCP and mixed intuitionistic linear logic. We then present an outline of our specification of the user interfaces for IP and TCP, demonstrating the role of the linear operators in the axioms. We present a specification of the data transfer component of the TCP protocol; and finally, we outline verification process undertaken to prove key properties of IP and TCP.

1.1 TCP/IP

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) are two essential elements of the communications stack at the heart of many network-based applications. Both of these protocols are typical of state-based distributed systems. IP is responsible for transmitting data from one internet node to another, but does not guarantee the delivery of data to the destination node. TCP is a protocol that sits on top of IP and it has the responsibility of establishing an end-to-end error free connection between peer TCP entities.

IP has no mechanisms to provide end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host communications protocols. There are no acknowledgements, either end-to-end or hop-by-hop, and the error detection provided by the IP checksum only covers the IP packet header and not the data itself. In IP there is no flow control or retransmission. IP packets can be lost, duplicated and delivered in any order.

TCP is layered on top of IP and it is its function to establish an error-free end-to-end connection between peer TCP entities. Since IP provides no guarantees in relation to data delivery, TCP provides all the necessary mechanisms, such as flow control, acknowledgements, and retransmission to ensure that the data is delivered in sequence and without duplication or error. TCP accomplishes this by segmenting the data and associating unique sequence numbers with each segment.

1.2 Mixed Intuitionistic Linear Logic

Linear logic[4] belongs to the family of sub-structural logics, which modify or eliminate the usual structural rules of Contraction, Exchange and Weakening. In linear logic the Contraction rule, which allows hypotheses to be duplicated, and the Weakening rule, which allows hypotheses to be discarded, are modified so that they are no longer structural rules but are specialised through the use of modalities. The effect of this is to make the logic "resource conscious", since each step in a deduction can be regarded in terms of its consumption or production of logical hypotheses.

Further, non-commutative linear logic[1] removes another structural rule, Exchange, which allows hypotheses to be reordered. The terms in the sequent are ordered and are typically represented as a list of terms. In a commutative linear logic the terms are not ordered and can be represented as a multiset or as a list with the inclusion of a structural Exchange rule to allow reordering of the terms in the sequent.

Mixed intuitionistic linear logic (MILL)[3] combines both commutative and non-commutative logics in the one system. Blocks are used to capture both ordered and non-ordered terms in the MILL sequents. The grammer of formulas in MILL is defined as:

$$\begin{aligned} F := {} & F \otimes F | F \odot F | F \& F | F \oplus F | F \multimap F | F \mathrel{-\!\!\bullet} F | F \mathrel{\bullet\!\!-} F | !F | \\ & \forall x.F | \exists x.F | \mathbf{1} | \top | \mathbf{0} | l \end{aligned}$$

where l is a literal.

The grammar of a block is defined as:

$G := F|()|(G;G)|(G,G)$

The rules that define a block are:

A,B = B,A   A,(B,C) = (A,B),C   A,( ) = A
A;(B;C) = (A;B);C   A;( ) = A

Both the ',' and ';' operators are associative but only the ',' operator is commutative. A block with no ';' operator is called a free block. Blocks are Series-Parallel orders[11] and have associated syntactic trees. The notation $G[\ \ ]$ represents a block with one leaf of the syntactic tree empty. $G[D]$ represents the block $G[\ \ ]$ with the empty leaf replaced by D.

The removal of the structural rules of Weakening, Contraction and Exchange means that the ordinary logical operators for conjunction, disjunction and implication are replaced with commutative and non-commutative linear versions. These include:

- Commutative multiplicative conjunction is written as A⊗B. When used both hypotheses are consumed in any order and are no longer available.
- Non-commutative multiplicative conjunction is written as A⊙B and represents the consumption of B after A.
- Additive conjunction is written A&B. When used it represents a deterministic choice of the hypothesis to be consumed.
- Additive disjunction is written A⊕B. When used it represents an external (non-deterministic) choice as to the hypothesis to be consumed.
- Linear implication is written A⊸B and represents a process that consumes A and produces B.
- Direct implication is written A−•B. If you have two non-commutative hypotheses A and A−•B, occurring in that order, then you can derive the hypothesis B, consuming the hypotheses A and A−•B.
- Retro implication is written A•−B. If you have two non-commutative hypotheses A•−B and B, occurring in that order, then you can derive the hypothesis A, consuming the hypotheses A•−B and B.

These operators are defined by the rules in figure 1.

The basis of our specification consists of a series of axioms, presented using the linear operators, which specify the valid transitions that can take place in the system. For IP we use ordinary commutative linear logic, since IP datagrams may be reordered in transmission. However, since order of receipt is important for TCP, we make use of a combination of commutative and non-commutative operators in its specification.

1.3 Verification Tools

The Isabelle theorem prover[6] was used to type-check and verify our specifications of TCP and IP. The specifications developed in Linear Logic were

$$\frac{\Gamma[\Delta;\Sigma]\vdash C}{\Gamma[\Delta,\Sigma]\vdash C}\,Entropy \qquad \frac{}{A\vdash A}\,Id \qquad \frac{\Gamma\vdash A \quad \Delta[A]\vdash C}{\Delta[\Gamma]\vdash C}\,Cut$$

$$\frac{\Gamma[A,B]\vdash C}{\Gamma[A\otimes B]\vdash C}\otimes L \qquad \frac{\Gamma\vdash A \quad \Delta\vdash B}{\Gamma,\Delta\vdash A\otimes B}\otimes R \qquad \frac{\Gamma[A;B]\vdash C}{\Gamma[A\odot B]\vdash C}\odot L \qquad \frac{\Gamma\vdash A \quad \Delta\vdash B}{\Gamma;\Delta\vdash A\odot B}\odot R$$

$$\frac{\Gamma[A]\vdash C}{\Gamma[A\&B]\vdash C}\&L_1 \qquad \frac{\Gamma[B]\vdash C}{\Gamma[A\&B]\vdash C}\&L_2 \qquad \frac{\Gamma\vdash A \quad \Gamma\vdash B}{\Gamma\vdash A\&B}\&R$$

$$\frac{\Gamma[A]\vdash C \quad \Gamma[B]\vdash C}{\Gamma[A\oplus B]\vdash C}\oplus L \qquad \frac{\Gamma\vdash A}{\Gamma\vdash A\oplus B}\oplus R_1 \qquad \frac{\Gamma\vdash B}{\Gamma\vdash A\oplus B}\oplus R_2$$

$$\frac{\Gamma\vdash A \quad \Delta[B]\vdash C}{\Delta[\Gamma,A\multimap B]\vdash C}\multimap L \qquad \frac{A,\Gamma\vdash B}{\Gamma\vdash A\multimap B}\multimap R$$

$$\frac{\Gamma\vdash A \quad \Delta[B]\vdash C}{\Delta[\Gamma;A\mathrel{-\!\!\bullet} B]\vdash C}\mathrel{-\!\!\bullet}L \qquad \frac{A;\Gamma\vdash B}{\Gamma\vdash A\mathrel{-\!\!\bullet} B}\mathrel{-\!\!\bullet}R$$

$$\frac{\Gamma\vdash A \quad \Delta[B]\vdash C}{\Delta[B\mathrel{\bullet\!\!-} A;\Gamma]\vdash C}\mathrel{\bullet\!\!-}L \qquad \frac{\Gamma;A\vdash B}{\Gamma\vdash B\mathrel{\bullet\!\!-} A}\mathrel{\bullet\!\!-}R$$

$$\frac{\Gamma[()]\vdash C}{\Gamma[\mathbf{1}]\vdash C}\,\mathbf{1}L \qquad \frac{}{\vdash\mathbf{1}}\,\mathbf{1}R$$

$$No\ \top\ left\ rule \qquad \frac{}{\Gamma\vdash\top}\top R \qquad \frac{}{\Gamma[\mathbf{0}]\vdash C}\,\mathbf{0}L \qquad No\ \mathbf{0}\ right\ rule$$

$$\frac{\Gamma[()]\vdash C}{\Gamma[!A]\vdash C}\,!w \qquad \frac{\Gamma[!A,!A]\vdash C}{\Gamma[!A]\vdash C}\,!c \qquad \frac{\Gamma[A]\vdash C}{\Gamma[!A]\vdash C}\,!d \qquad \frac{!\Gamma\vdash C}{!\Gamma\vdash !C}\,!p(\Gamma\ is\ free)$$

$$\frac{\Gamma[A[t/x]]\vdash C}{\Gamma[\forall x.A]\vdash C}\forall L \qquad \frac{\Gamma\vdash C}{\Gamma\vdash\forall x.C}\forall R(x\ \notin\ FV(\Gamma))$$

$$\frac{\Gamma[A]\vdash C}{\Gamma[\exists x.A]\vdash C}\exists L(x\ \notin\ FV(\Gamma[()],B)) \qquad \frac{\Gamma\vdash C[t/x]}{\Gamma\vdash\exists x.C}\exists R$$

Fig. 1. Rules for operators in Mixed Intuitionstic Linear Logic

originally based on an encoding of Linear Logic in the Coq proof assistant[2][9], but later both this encoding, and an extension to deal with MILL were implemented in Isabelle.

The Isabelle tool provides:

- a uniformity of notation;
- a verification of type-correctness; and
- a semi-automated system in which properties of the specifications can be proved.

1.3.1 Implementing MILL in Isabelle

The MILL specifications were verified using the Isabelle theorem assistant using the HOL library. In addition to the advantages enumerated above, Isabelle/HOL:

- provides predefined types such as lists, sets and multisets;
- provides the ability to define new types; and
- supports inductive definitions in addition to the definition-by-cases style found in functional languages.

MILL is implemented in two theory files. One file defines the Isabelle/HOL theory for blocks and defines, using primitive recursion, the operations that can be performed on blocks. The second file defines the Isabelle/HOL theory for MILL. Valid MILL judgements are represented by the HOL proposition $Truelin$. A valid MILL judgement consists of an antecedent which is a block of linear terms and a consequence which is a single linear term. A valid MILL sequent is modeled as a consequence relation between two valid MILL judgements.

2 The Specification of TCP/IP

In this section we briefly outline the main axioms that describe both the IP and TCP user interfaces. It should be noted that the full specification also involves a description of the TCP protocol which links these interfaces, but these axioms have been elided in this article.

2.1 The Internet Protocol User Interface

There are two main operations available to the user of the IP layer:

$Send(x, y, ttl, m)$ Sends a message m from node x to node y with a "time to live" value of ttl.

$Rcv(x, y, ttl, m)$ Receives a message m from node x to node y with a "time to live" value of ttl.

We use three axioms to define the operation of the IP layer, describing the sending of datagrams, their possible loss or duplication during transmission, and their receipt. To improve readability in all subsequent axioms, all the parameters are assumed to be universally quantified unless specifically stated in the axiom.

First, sending a message adds a single datagram to the system.

$$\boxed{Send(x, y, ttl, m) \multimap Datagram(x, y, lower(ttl), m)} \tag{1}$$

Here $lower$ is a function that reduces its operand to some non-negative integer in the range [0,ttl).

Second, when in transmission a datagram can be duplicated or lost.

$$\boxed{\begin{aligned} Datagram(x, y, ttl, m) \multimap (&Datagram(x, y, lower(ttl), m) \\ &\otimes Datagram(x, y, lower(ttl), m)) \oplus \mathbf{1} \end{aligned}} \tag{2}$$

1 is the unit for multiplicative conjunction, and is commonly used to represent the consumption of resources with no corresponding product.

Finally, if a datagram addressed to node y exists and node y is listening for it, the node y will receive the message m or some corrupted version $corrupt(m)$ of that message.

$$\boxed{\begin{aligned} Datagram(x, y, ttl, m) \otimes Listen(y) \multimap\ &Listen(y) \\ &\otimes(Rcv(x, y, lower(ttl), m) \\ &\oplus Rcv(x, y, lower(ttl), \\ &\qquad corrupt(m))) \end{aligned}} \tag{3}$$

Of course many issues relevant to IP have been omitted here (most notably routing and fragmentation), but this specification provides a sufficient base for the verification of the relevant properties of TCP.

2.2 TCP User Operations

We will define the user interface to the TCP layer as consisting of the following operations:

$New(s, d)$ Create a socket for use in making a connection between a source internet address s and a destination address d.

$Accept(s, d)$ Accept an attempt from some internet address s to make a connection to the internet address d.

$Write(s, d, m)$ Write the data m on the connection from internet address s to internet address d.

$Read(s, d, l)$ Attempt to read l octets from a connection from internet address s to internet address d. If there are less than l octets in the connection, *Read* will read all the octets in the connection.

$Close(s, d)$ Terminate the connection between internet addresses s and d.

2.3 Specification of the TCP Layer Interface

In order to describe the TCP user interface we define a predicate to describe the current status of a connection. $Stream(s, d, mw, mr, b)$ represents one side of the connection from address s to address d. The data written by s is split into mw, the data not yet read by d, and mr, the data that has been read by d. The variable b is a boolean flag which is false once an end of stream (EOS) character is written to the stream. A full TCP session will thus consists of a pair of these streams, one each for s and d.

A connection is set-up between internet addresses s and d if both of the hosts of internet addresses s and d actively specify the connection or if one host actively specifies the connection and the other host passively accepts connections from a specified internet address.

$$(4) \qquad \boxed{\begin{array}{l} (New(s, d) \otimes New(d, s)) \oplus (Accept(s, d) \otimes New(d, s)) \multimap \\ Stream(s, d, nil, nil, true) \otimes Stream(d, s, nil, nil, true) \end{array}}$$

Axiom (5) specifies the effect of writing some data at s, which is then appended to the data previously written (we use "::" to represent list concatenation).

$$\boxed{\begin{array}{l} Stream(s,d,m1,m2,true) \otimes Write(s,d,m) -\!\!\bullet \\ Stream(s,d,m1 :: m,m2,true) \end{array}} \tag{5}$$

Axiom (6) specifies the operation of reading some data at d, which transfers exactly l octets from $m1$ to $m2$. The function $drop(n,l)$ returns a list with n elements removed from the beginning of list l. The function $take(n,l)$ returns a list of the first n elements from list l.

$$\boxed{\begin{array}{l} \forall l \leq length(m1). \\ Stream(s,d,m1,m2,b) \odot Read(s,d,l) -\!\!\bullet \\ Stream(s,d,drop(l,m1),m2 :: take(l,m1),b) \end{array}} \tag{6}$$

Finally, axiom (7) specifies the normal closing sequence, where it is only necessary to change the flag on the relevant stream from $true$ to $false$. Note that the premises in equations (5) and (6) will ensure that this stream can still be read from, but not written to.

$$\boxed{\begin{array}{l} Close(s,d) \otimes Stream(s,d,m1,m2,true) -\!\!\bullet \\ Stream(s,d,m1,m2,false) \end{array}} \tag{7}$$

We define a valid TCP connection as one consisting of two streams, where the data read at one address is exactly the data written by the other. Specifying this within the system is straightforward: a valid session is any block of Linear Propositions (including, of course, the commands defined above) that gives rise to a consistent, completed pair of streams.

Definition 2.1 ValidSession
$\forall s, d$:node; $read\_by\_x$, $read\_by\_y$:(list data);
session: (linear block)
session $\vdash Stream(s,d,nil,read\_by\_y,false) \otimes Stream(d,s,nil,read\_by\_x,false)$.

3 TCP Data Transfer Protocol

In this section a simple, naive specification of the data transfer protocol in TCP is presented. Since the aim is to show that this protocol specification on top of the IP layer interface specification has the properties of a TCP connection, namely the data delivered and the data in transit forms a sequence, many of the extensions required for a practical TCP system will be ignored. In particular, it will be assumed that acknowledgments are not lost and that segment fragmentation due to internetworking will not occur. These are not serious limitations since zero window probes can be used to overcome lost acknowledgments and IPv6 allows a connection to determine the smallest Maximum Transfer Unit between a source and destination. It is assumed that the checksums of all IP datagrams and TCP segments are initially checked and that datagrams and segments that fail this test are discarded.

The TCP user interface of the previous section represented each direction of the data transfer using a single predicate. At the protocol level we represent this state using two predicates, one each for the sender and receiver. The task of the TCP protocol specification then is to ensure that these predicates, while maintained separately, are kept in a consistent state by each of the protocol axioms. Keeping track of the sequence numbers of the data sent and received will be central to this task.

The two predicates representing a stream of data from s to d are:

$SS(s, d, s_{nxt}, s_{una}, w_{buf}, rtq, r_{win})$

This is the state at s of data transfer to d. Here, s_{una} is the oldest unacknowledged sequence number, s_{nxt} is the next sequence number to be sent. The write buffer w_{buf} holds data written by s but not yet sent, while the retransmission queue, rtq, holds data that has been sent,but not yet acknowledged. The current size of the receiver window is r_{win}.

$RS(d, s, r_{nxt}, r_{win}, r_{buf})$

This is the state at d on a stream receiving data from s. The sequence number r_{nxt} is the start sequence number of the next data to be received. The available space left in the receiver's buffer is r_{win}. Once received, the data will be stored in r_{buf}, where the size of the receive window.

In addition we make use of two predicates $HaveWritten$ and $HaveRead$ to record all the data written by the sender and read by the receiver respectively. These are necessary in order to state the basic consistency property of the TCP protocol - that all the data written by the sender is eventually read, in the same order, by the receiver.

The axioms defining the TCP data transfer protocol are as follows.

A user $Write$ operation has the effect of appending the data to the sender's write buffer, as described in axiom (8). A sequence of data leaves the write buffer through axiom (9) which sends this data through the IP layer via the

$IPSend$ predicate. The size of the data to be sent can be determined from the current value of the receiver's window, along with the maximum transmissible unit (MTU). This value is then used in preparing the IP message and in recalculating the sequence numbers in the sender's state. As the data is sent to the IP layer it is also lodged in the retransmission queue, pending an acknowledgement.

$$\boxed{\begin{array}{l}
(SS(s,d,s_{nxt},s_{una},w_{buf},rtq,r_{win}) \otimes HaveWritten(s,d,hw)) \\
\odot Write(s,d,m) \multimapdot \\
SS(s,d,s_{nxt},s_{una},w_{buf} :: m,rtq,r_{win}) \otimes HaveWritten(s,d,hw :: m)
\end{array}} \tag{8}$$

$$\boxed{\begin{array}{l}
length(m) \leq r_{win} \Rightarrow \\
SS(s,d,s_{nxt},s_{una},m :: w_b,rtq,r_{win}) \multimap \\
IPSend(s,d,mkMsg(s_{nxt},m)) \otimes \\
SS(s,d,s_{nxt}+length(m),s_{una},w_b,rtq :: m,r_{win}-length(m))
\end{array}} \tag{9}$$

$$\boxed{\begin{array}{l}
(length(m) \leq r_{win}) \wedge (s_n + length(m) > r_{nxt}) \wedge (s_n \leq r_{nxt}) \Rightarrow \\
RS(d,s,r_{nxt},r_{win},r_{buf}) \odot IPRecv(s,d,mkMsg(s_n,m)) \multimapdot \\
IPSend(d,s,mkAck(s_n + length(m),r_{win}-length(m))) \otimes \\
RS(d,s,s_n + length(m),r_{win}-length(m), \\
\qquad r_{buf} :: drop((r_{nxt}-s_n),m))
\end{array}} \tag{10}$$

$$\boxed{\begin{array}{l}
s_{una} \leq r_{ack} \leq s_{nxt} \Rightarrow \\
SS(s,d,s_{nxt},s_{una},w_{buf},rtq,r_{win}) \odot IPRecv(d,s,mkAck(r_{ack},r'_{win})) \multimapdot \\
SS(s,d,s_{nxt},r_{ack},w_{buf},update(r_{ack},rtq),r'_{win})
\end{array}} \tag{11}$$

$$\boxed{\begin{array}{l}
s_n + length(m) \leq r_{nxt} \Rightarrow \\
RS(d,s,r_{nxt},r_{win},r_{buf}) \odot IPRecv(s,d,mkMsg(s_n,m)) \multimapdot \\
RS(d,s,r_{nxt},r_{win},r_{buf})
\end{array}} \tag{12}$$

$$\boxed{IPRecv(s, d, corrupt(m)) \mathrel{-\!\!\bullet} \mathbf{1}} \tag{13}$$

$$\boxed{\begin{aligned}&(RS(d, s, r_{nxt}, r_{win}, m :: r_b) \otimes HaveRead(s, d, hr))\odot\\&Read(s, d, length(m)) \mathrel{-\!\!\bullet}\\&RS(d, s, r_{nxt}, r_{win} + length(m), r_b) \otimes HaveRead(s, d, hr :: m)\end{aligned}} \tag{14}$$

The axiom (11) is triggered by the receipt of an acknowledgement from the receiver, contained in an $IPRecv$ message from the IP layer. Assuming that this is acknowledging a message still in the retransmission queue (i.e. a message with sequence number in between s_{una} and s_{nxt}), we then use the function update to filter all acknowledged messages out of the queue.

Axioms (10), (12) and (13) are each triggered by the receipt of an $IPRecv$ signal from the IP layer. Corrupted messages are automatically rejected, but if the message is not corrupt, then its acceptance depends firstly on there being enough room in the read buffer ($msgSize < r_{wnd}$) and secondly on not having received the message already, based on comparing its sequence number with r_{nxt}.

Finally, axiom (14) handles the Read request from the user layer by supplying data from the read buffer. The predicate $HaveRead$ is also updated at this point.

4 Verification of MILL Specifications

4.1 Verification of IP

The IP specification guarantees very little about a message sent from one node to another, since messages may be corrupted, duplicated or even lost. However little the IP specification guarantees, it does imply that a message that arrives at a node must have been created at some node. If a node receives a message with a correct header - as validated by the checksum - then some node must have sent a message with the same header (the actual message itself may be corrupted, of course). In fact, if the initial datagram is not lost, a message sent from node A to node B may result in one or more multiple messages, with correct header, being received by node B. These messages will not appear out of mid-air.

Here we use $(\otimes^n Rcv(x, y, ttl, m))$ as a shorthand for the receipt of n possibly corrupted version of message m, which we can define inductively over multiplicative conjunction as follows:

$$\begin{aligned}(\otimes^0 Rcv(x,y,ttl,m)) &= \mathbf{1}\\ (\otimes^{n+1} Rcv(x,y,ttl,m)) &= (Rcv(x,y,ttl,m) \oplus Rcv(x,y,ttl,corrupt(m))) \otimes\\ &\quad (\otimes^n Rcv(x,y,ttl,m))\end{aligned}$$

In order to prove that messages are not created from mid-air we need to prove:

Theorem 4.1 *NoMidAirMessages*
$\forall x, y$*:node; ttl:nat; m:message* $\cdot$
$Send(x,y,ttl,m) \otimes Listen(y) \multimap ((\otimes^n Rcv(x,y,ttl',m)) \otimes Listen(y)) \oplus Listen(y)$
where ttl' $\leq$ *ttl*

Our specification of IP using MILL allows us to prove this property and the axiomatisation of MILL in Isabelle/HOL allows us to verify that the proof steps are correct.

In fact, to isolate the inductive step, we prove the following lemma which asserts that a message in transit, represented by $Datagram(x,y,ttl,m)$, may generate arbitrary many receipts of that message:

Lemma 4.2 *RecvDGClosure :*
$\forall x, y$*:node; ttl:nat; m:message* $\cdot$
$Send(x,y,m) \otimes Listen(y) \multimap (\otimes^n Datagram(x,y,m)) \otimes Listen(y)$

The proof of NoMidAirMessages and RecvDGClosure proceeds by induction over n. This inductive property of the natural numbers, along with many of the other usual properties, is part of the Isabelle/HOL system and, since this exists at the meta-level for our linear logic encoding, can be used to structure our proof here. The general format of the inductive proofs are given in figures 2 and 3. These figures show the general outline of the proofs that have been verified by the theorem assistant, Isabelle. In these outlines the following abbreviations have been used.

$L = Listen(y)$
$D = Datagram(x,y,ttl,m)$
$S = Send(x,y,m)$
$R = Rcv(x,y,ttl,m) \oplus Rcv(x,y,ttl,corrupt(m))$

4.2 Verification of TCP

In order to verify that the specification meets the basic requirements of TCP data transfer, we must show that each of the rules from the TCP protocol presented in the previous section maintains the integrity of the transmitted

$$\dfrac{\dfrac{}{\vdash S \otimes L \multimap D \otimes L} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash D \multimap D \otimes D} \quad \dfrac{}{(\otimes^n D) \vdash (\otimes^n D)}}{(\otimes^n D) \otimes D \vdash \otimes^n D) \otimes D \otimes D}}{(\otimes^n D) \vdash (\otimes^n D) \otimes D} \qquad \dfrac{}{L \vdash L}}{(\otimes^n D) \otimes L \vdash (\otimes^{n+1} D) \otimes L} \qquad \dfrac{}{S \otimes L \vdash (\otimes^n D) \otimes L}}{S \otimes L \vdash (\otimes^{n+1} D) \otimes L}}{\dfrac{S \otimes L \vdash (\otimes^n D) \otimes L}{\vdash S \otimes L \multimap (\otimes^n D) \otimes L}}$$

Fig. 2. Outline Proof of the `RecvDGClosure` lemma

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\otimes^n D) \otimes L \vdash (\otimes^n R) \otimes L} \quad \dfrac{}{R \vdash R}}{(\otimes^n D), R \otimes L \vdash (\otimes^n R) \otimes R \otimes L} \qquad \dfrac{}{\vdash D \otimes L \multimap R \otimes L}}{(\otimes^n D), D, L \vdash (\otimes^n R) \otimes R \otimes L}}{(\otimes^{n+1} D) \otimes L \vdash (\otimes^{n+1} R) \otimes L} \qquad \dfrac{}{\vdash D \otimes L \multimap R \otimes L}}{(\otimes^n D) \otimes L \vdash (\otimes^n R) \otimes L}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash S \otimes L \multimap (\otimes^n D) \otimes L} \quad (\otimes^n D) \otimes L \vdash (\otimes^n R) \otimes L}{S \otimes L \vdash (\otimes^n R) \otimes L}}{S \otimes L \vdash ((\otimes^n R) \otimes L) \oplus L}}{\vdash S \otimes L \multimap ((\otimes^n R) \otimes L) \oplus L}$$

Fig. 3. Outline Proof of the `NoMidAirMessages` theorem

data. That is, a TCP connection which closes successfully will have the property that all data written by the sender is received, in the same order, by the receiver.

The approach taken here is to formulate an invariant that is maintained by each axiom and, in the final state where all data has been sent, this implies the ordered transmission and receipt of this data. Each of the data structures involved in the sender's and receiver's state is expressed as a subsequence of the total data being transmitted, indexed by the sequence numbers.

Specifically, at any point during transmission, the state of the system should be as in figure 4. Each piece of data written (as captured by the $HaveWritten$ predicate) should reside either in the write buffer, the retransmission queue, the read buffer, or else be captured by the $HaveRead$ predicate. The total data is not the exact concatenation of these sequences, since there is potentially an overlap between the retransmission queue and the read buffer. The retransmission queue contains data that has been transmitted by the sender and acknowledged by the receiver, but the acknowledgement has not yet been received by the sender.

To this end, we define each of the data structures in terms of the data captured by $HaveWritten$. The invariant itself is expressed in classical logic, operating here as a meta-logic for the embedded MILL deduction. The func-

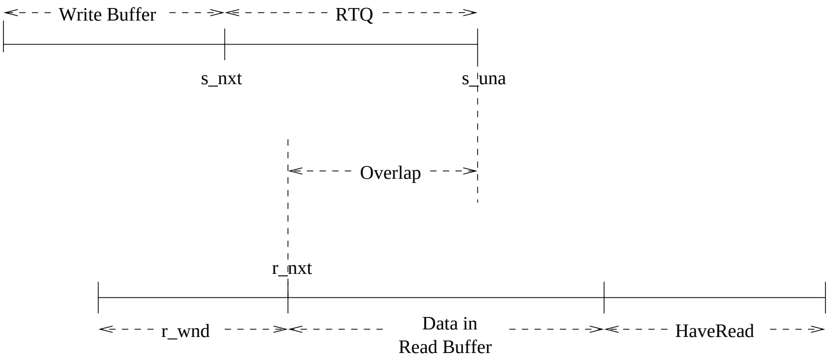

Fig. 4. The state of the TCP data structures during transmission.

tion *sublist* returns a subsection of a list specified by indices, *datacat* concatenates the data from the messages in the retransmission queue into a single list of data, and *drop* is a function removing an initial segment from a list, as specified by the index. From this, it is a relatively straightforward matter to demonstrate that whenever this invariant holds we must have:

$$(hr :: r_{buf} :: drop(datacat(r_{nxt} - s_{una}, rtq)) :: w_{buf}) = hw$$

where

$hr = sublist(hw, 0, length(hr))$
$r_{buf} = sublist(hw, length(hr), r_{nxt})$
$datacat(rtq) = sublist(hw, s_{una}, s_{nxt})$
$w_{buf} = sublist(hw, s_{nxt}, length(hw))$
$length(hr) < r_{nxt}$
$s_{una} < r_{nxt}$
$r_{nxt} \leq s_{nxt} \leq length(hw)$

which is equivalent to:

$$(sublist(hw, 0, length(hr)) :: sublist(hw, length(hr), r_{nxt}) :: sublist(hw, r_{nxt}, s_{nxt}) :: sublist(hw, s_{nxt}, length(hw)) = hw$$

$\Rightarrow$

$$\boxed{0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw)} \tag{15}$$

Specifically, when all the data has been sent, and the read buffer, the retransmission queue and the write buffer are empty, this equality collapses into a simple statement that $hr = hw$ i.e. the data received is exactly the same as the data that was sent.

The linear implication and direct implication operators of MILL represent the consumption of a linear resource and the production of another linear resource. Therefore in statements such as $A \multimap B$ and $A -\!\!\bullet\, B$, A can represent the state before the implication and B represents the state after the impli-

$$
\begin{aligned}
&(0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw)\\
&\Rightarrow 0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw :: m))\\
&\quad \Rightarrow length(hw) \leq length(hw :: m)\\
&\quad \Rightarrow length(hw) \leq length(hw) + length(m)\\
&\quad \Rightarrow 0 \leq length(m)
\end{aligned}
$$

Fig. 5. Proof of invariant for axiom (8)

cation. The axioms used to define the data transfer protocol (axioms (8) to (14)) have the form of a linear or direct implication where the left-hand side of the implication is a state with one or more events or processes occurring concurrently or sequentially. The right-hand side of these implications is a state with possible concurrent events or processes. Therefore a statement such as:

$$(S(a, b, c) \otimes A) \odot B \multimap\!\!\bullet\; S(d, e, f)$$

can be interpreted as saying that if A is true while the state variables of state S have the values a, b and c respectively and then B occurs, then the events A and B are consumed and the state variables of state S now have the values d, e, and f respectively.

Therefore for axioms (8) to (14) we need to show that if the invariant (equation (15)) holds before the applications of each axiom then it must hold after the application of each axiom. The proofs that axioms (11), (12) and (13) preserve the invariant is trivial since these axioms do not modify the elements of the invariant (namely hr, r_{nxt}, s_{nxt} and hw). For example axiom (11) only modifies rtq and s_{una}. Figures 5, 6, 7 and 8 give the proofs that axioms (8), (9), (10) and (14) maintain the invariant. The proof that axiom (10) maintains the invariant requires the assumption by the receiver that the sender will only send data that has been placed in the write buffer. The sequence number of the last octet received must be less than the next octet the sender will transmit, i.e. $(s_n + length(m) < s_{nxt})$. Since the sender follows axiom (9) then this assumption is valid.

5 Conclusions

In this article we have presented a specification of two key protocols used in networked systems, namely the Internet Protocol (IP) and the Transmission Control Protocol (TCP). In particular we have focussed on the data transfer component of the TCP protocol. Key properties of both protocols

$$
\begin{aligned}
&(0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw)\\
&\Rightarrow 0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} + length(m) \leq length(hw))\\
&\quad \Rightarrow s_{nxt} + length(m) \leq length(hw)\\
&\quad \Rightarrow length(m) \leq length(hw) - s_{nxt}\\
&\quad \Rightarrow length(m) \leq length(m :: w_b)\\
&\quad \Rightarrow length(m) \leq length(m) + length(w_b)\\
&\quad \Rightarrow 0 \leq length(w_b)
\end{aligned}
$$

Fig. 6. Proof of invariant for axiom (9)

$$
\begin{aligned}
&(0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw)\\
&\Rightarrow 0 \leq length(hr) \leq s_n + length(m) \leq s_{nxt} \leq length(hw))\\
&\quad \Rightarrow length(hr) \leq s_n + length(m) \leq s_{nxt}\\
&\quad \Rightarrow length(hr) \leq r_{nxt} < s_n + length(m) \leq s_{nxt}\\
&\quad \Rightarrow s_n + length(m) \leq s_{nxt}
\end{aligned}
$$

Fig. 7. Proof of invariant for axiom (10)

$$
\begin{aligned}
&(0 \leq length(hr) \leq r_{nxt} \leq s_{nxt} \leq length(hw)\\
&\Rightarrow 0 \leq length(hr :: m) \leq r_{nxt} \leq s_{nxt} \leq length(hw))\\
&\quad \Rightarrow 0 \leq length(hr :: m) \leq r_{nxt}\\
&\quad \Rightarrow 0 \leq length(hr) + length(m) \leq r_{nxt}\\
&\quad \Rightarrow length(hr) + length(m) \leq r_{nxt}\\
&\quad \Rightarrow length(m) \leq r_{nxt} - length(hr)\\
&\quad \Rightarrow length(m) \leq length(m :: r_b)\\
&\quad \Rightarrow length(m) \leq length(m) + length(r_b)\\
&\quad \Rightarrow 0 \leq length(r_b)
\end{aligned}
$$

Fig. 8. Proof of invariant for axiom (14)

have been proven. The formalism used to specify these protocols was Mixed Intuitionistic Linear Logic (MILL). MILL combines both commutative and non-commutative operators and it is a convenient logical formalism to model events that can occur concurrently and/or in sequence. While IP can be specified and verified using linear logic[10], using linear logic to specify TCP is problematical since TCP requires the ordering of certain events to be maintained. By using MILL, commutative and non-commutative operators can be combined in the specification. The specification of TCP defines an existing state and the events that operate concurrently or sequentially on this state to generate a new state and new events. This has the effect of defining the changes to the local state. In addition, by embedding our formalism of MILL in the Isabelle/HOL theorem prover we use the premises of each axiom to define the global state in which the axiom is valid.

We have outlined the verification of the "no mid-air messages" property of the IP protocol and this proof was implemented in the Isabelle/HOL system in order to check the proof. To verify that the data transfer section of the TCP protocol maintained the integrity of the transmitted data, we formulated an invariant that must be maintained by all the data structures involved in the data transfer. We have shown that the invariant is maintained by each of the axioms used to define the data transfer protocols. These proofs have also been checked using the Isabelle/HOL systems as a theorem assistant.

The MILL formalism has proved successful for both the specification and verification of this distributed system containing "stateless" (IP) and "stateful" (TCP) subsystems. Future work will extend these techniques and apply them to other complex systems with concurrent and sequential processes.

Acknowledgement

The authors would like to thank the anonymous reviewrs for their helpful comments and suggestions.

References

[1] Abrusci, V.M., *Phase semantics and sequent calculus for pure non-commutative classical linear propositional logic*, Journal of Symbolic Logic, **56(4)** (1991), 1403–1451.

[2] Cornes, C et al., "The Coq Proof Assistant Reference Manual", Rapport Technique 177, INRIA, (1995).

[3] Damaille, A., "Yet Another Mixed Intuitionistic Linear Logic", Technical Report, École Nationale Supérieure des Télécommunications, (1998).

[4] Girard, J-Y., *Linear Logic*, Theoretical Computer Science. **50** (1987), 1–102.

[5] Girard, J-Y., *On the unity of logic*, Annals of Pure and Applied Logic. **59** (1993), 201–217.

[6] Paulson, L.C., “Isabelle: A Generic Theorem Prover”, Springer Verlag LNCS 828, (1994).

[7] Postel, J. “RFC 791, Internet Protocol”, Defense Advanced Research Projects Agency, (1981).

[8] Postel, J. “RFC 79, Transmission Control Protocol”, Defense Advanced Research Projects Agency, (1981).

[9] Power, J. and C. Webster, *Working with Linear Logic in Coq*, 12th International Conference on Theorem Proving in Higher Order Logics, Work-in-progress Report, Nice, France, (1999).

[10] Sinclair, D., P. Gibson, D. Gray, G. Hamilton and J. Power, *Specifying and Verifying IP with Linear Logic*, International Workshop on Distributed Systems Validation and Verification, Taipei, Taiwan, (2000).

[11] Retoré, C. “Réseaux et Séquents Ordonnés”, Ph.D. thesis, University of Paris VII, (1993).