Specifying and Verifying IP with Linear Logic

David Sinclair*‡, James Power†, Paul Gibson†, David Gray*, Geoff Hamilton*
*School of Computer Applications, Dublin City University, Glasnevin, Dublin 9, Ireland
†Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland
‡E-mail for correspondance: David.Sinclair@compapp.dcu.ie

Abstract

This paper presents a specification and verification of the Internet Protocol (IP) in linear logic. IP has the essential propoerties of a typical distributed system. This paper shows how linear logic can be used to prove some properties of this layer. Both the specification and the correctness proofs have been verified using the Coq proof assistant, via the authors' embedding of linear logic into this constructive framework.

Keywords: linear logic, verification, protocols, Coq, theorem proving

1 Introduction

Distributed systems are becoming a standard computing paradigm and raise new and difficult issues in the areas of validation and particularly verification. To date the main approaches used have been centred on model checking or proof theoretic approaches. In this paper we use the proof theoretic approach to the specification and verification of the IP layer [9] using linear logic [3]. The IP layer, albeit small and self-contained, typifies much of the essential behaviour of a distributed system.

We demonstrate here the linear logic concept of resource-consumption which is used as the key to describing the transitions between the "states" of the system. This contrasts with other proof-theoretic approaches where such transitions are represented either by sets of traces, as in [8] or by modal operators, as in [2].

2 A Crash Course in Linear Logic

One way to look at a logic is to divide its rules into three categories:

1. *Axioms.* These are the defined truths of the logical system. In classical logic, the sole axiom represents the tautology that from hypothesis A one can deduce hypothesis A.
2. *Structural rules.* These specify how hypotheses can be manipulated. In classical logic these are the Exchange, Contraction and Weakening rules.
3. *Logical rules.* These define the logical operators, which in the case of classical logic are negation ($\neg$) conjunction ($\wedge$), disjunction ($\vee$) and implication ($\Rightarrow$).

Linear logic belongs to the family of *sub-structural logics*, where we remove the Contraction rule, which allows hypotheses to be duplicated, and the Weakening rule, which allows hypotheses to be discarded. As a result of losing these two structural rules, linear logic now admits a more specific set of logical connectives, and thus an enhanced set of logical rules.

The effect of this is to make the logic "resource conscious", since you cannot now duplicate or discard hypotheses. If you have one instance of a hypothesis A then you cannot create additional instances of A out of mid-air; nor can you simply sweep A under the carpet and pretend you never had A.

Since we embed linear logic in a higher-order constructive system we need only implement a fragment of intuitionistic linear logic (ILL) rather than full classical or intuitionistic linear logic. The connectives for ILL are:

- Linear implication is written $A \multimap B$ and is pronounced "consume A yielding B". If you have two hypotheses A and $A \multimap B$ then you can derive B but the hypothesis A has been consumed and is no longer available.
- Multiplicative conjunction is written as $A \otimes B$ and is pronounced "both A and B". When used both hypothesis are consumed and are no longer available.
- Additive conjunction is written $A\&B$ and is pronounced "choose from A and B". When used you choose which hypothesis is consumed and is no longer available.

- Additive disjunction is written $A \oplus B$ and is pronounced "either A or B". It represents a non-deterministic choice as to which hypothesis is consumed and is no longer available. This non-deterministic choice can be viewed as a choice made by some external agency.

The extra specification power of these connectives is a direct result of their resource-sensitivity. In classical logic an assumption of the form $A \wedge B$ allows us to use either or both of A and B. In linear logic, an assumption of the form $A \otimes B$ gives us both A and B (and insists we use both of them), whereas $A\&B$ covers the case where we choose to use just one of A and B.

2.1 The Coq Proof Assistant

Our system was developed using the Coq proof assistant [1], which is based on higher-order constructive logic. As well as the normal benefits of a proof assistant such as uniformity of notation and verification of type-correctness, Coq also provides three enhancements to ordinary constructive logic:

- Coq implements a higher-order constructive logic, facilitating in particular, the descriptions of object logics within the framework
- Coq has two type hierarchies: `Set` of constructive types, and `Prop` for classical logic allowing both programs and proofs in the same framework.
- Coq supports inductive definitions, giving a natural logical extension of the definition-by-cases style of programming found in functional languages

Some systems designed specifically for encoding logics distinguish between the system's own meta logic, and the object logic that this can be used to define (as in, for example, [6] which encodes linear logic using Isabelle [7]). In contrast to this, Coq provides a single homogeneous system with a single built-in concept of deduction, and so our definition of ILL exists as an ordinary datatype within this system.

While Coq does allow user-defined syntax rules for new operators, we have enhanced this considerably in the Coq code presented here in order to submerge some of the less familiar aspects of the Coq system. Unless otherwise indicated, there is a one-to-one mapping between the code presented here and the actual Coq vernacular.

2.2 Mixing the Logics

Linear logic is perhaps most usefully applied to state-based problems when used along with classical or intuitionistic logic. In particular, while state-specific assertions can be phrased in ILL, it is often useful to be able to express global invariants in classical logic, since it should be possible to use these as often as possible.

Classical intuitionistic predicates may be incorporated into ILL following [4] by marking them with a modal operator. Thus we can write $!A$ to denote "arbitrarily many A's", and the rules for this operator assert that we are free to use A zero, one or many times as we need. It is one of the particular features of our approach that we have an alternative to the use of this operator.

By encoding ILL as a simple consequence relation within the Coq system, we may freely mix linear assertions pertaining to the state with intuitionistic or classical assertions that are state-invariant. Since the objects that are used in the linear predicates range over Coq datatypes, these may also be constrained by ordinary (intuitionistic or classical) predicates, providing unrestricted use.

The significant benefit here is that all of the existing theories developed for Coq do not need to be changed for use with linear theorems, but can be integrated directly into the proofs. This is particularly useful when dealing with datatypes such as the natural numbers or lists, where re-encoding "linear" versions of the results would provide a significant overhead.

2.3 Representing Linear Logic in Coq

As indicated above, setting up the proof system involves two main steps: defining a type of ILL predicates and their associated connectives, and then defining a consequence operator and the associated sequent rules. In Coq terms we have:

- a type `ILinProp` to represent intuitionistic linear-logic propositions, which are defined inductively over the three logical units (**1**, **0** and $\top$) and the four connectives
- The sequent rules which define deduction in linear logic; each of the left and right sequent rules can be represented as an axiom in Coq

For example, the right sequent rule for multiplicative conjunction, $\otimes_R$ is usually expressed as:

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \otimes_R$$

In Coq, this becomes

```
Variable x,y:node; ttl:nat; m:message.

Axiom SendDG : (Send x y ttl m) ⊢ (Datagram x y ttl m).

Axiom LoseDG : (Datagram x y ttl m) ⊢ 1.

Axiom DuplDG : (Datagram x y ttl m) ⊢ (Datagram x y ttl m) ⊗ (Datagram x y ttl m).

Axiom RecvDG : (Datagram x y ttl m) ⊗ (Listen y) ⊢
               (Listen y) ⊗ ((Recv x y ttl m) ⊕ (Recv x y ttl (corrupt m))).
```

Figure 1: *Coq code for the axioms defining the user interface to the IP layer.*

```
Axiom TimesRight :
  (A,B : ILinProp)(Γ,Δ : (list ILinProp))
  ((Γ ⊢ A) ⇒ (Δ ⊢ B) ⇒ (Γ^Δ ⊢ A ⊗ B))
```

Note how both antecedents of the sequent rule are represented in Coq using a curried version of classical implication[1], represented by "⇒". This is effectively at the meta-level for the linear-logic encoding, and can thus represent deduction in the sequent calculus. A fuller description of this encoding may be found in [10].

3 A Quick Description of the IP Layer

Two quotes from the IP specification[9] sum up the IP layer:

> "There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols."[§1.2]
>
> "The internet protocol does not provide a reliable communication facility. There are no acknowledgements either end-to-end or hop-by-hop. There is no error control for data, only the header checksum. There is no retransmission. There is no flow control."[§1.4]

The IP specification specifies the format of the internet datagram header that is attached to each datagram transmitted through the IP layer. The key elements of the header for our purposes are:

- The *source address* field, specifying the originator of the message
- The *destination address* field, specifying the intended recipient of the message
- The *time-to-live* field, which indicates the maximum time, in seconds, that a datagram is allowed to remain in the internet system.
- The *header checksum* which is used to verify the validity of the header fields only (i.e. not the actual message contents)

These last two fields give rise to two properties that we *can* assert about the IP layer, since the header checksum validates both the source and destination addresses, and the time-to-live field imposes a bound on the time within which a datagram can be received.

4 Specification of IP Layer Interface

We can represent the main operations of the IP layer using three predicates which encapsulate the status of a message being sent, being in transit, and being received:

1. (*Datagram x y ttl m*)
   This represents the existence of a datagram with message m, in transit from node x to node y, with a time-to-live value of *ttl*.
2. (*Send x y ttl m*)
   Denotes a user sending a message m from node x to node y with a time-to-live value of *ttl*.
3. (*Recv x y ttl m*)
   Denotes a user receiving a message m from node x at node y with a time-to-live value of *ttl*.
4. (*Listen y*)
   Denotes a user listening for incoming datagrams at node y.

[1] In fact, we also use currying here, which allows us represent "$X \wedge Y \Rightarrow Z$" as "$X \Rightarrow Y \Rightarrow Z$"

```
Variable x,y:node; ttl:nat; m:message.

Axiom ClockTick : (Datagram x y (ttl+1) m) ⊗ Clock ⊢ (Datagram x y ttl m).

Axiom InterNode : ∀ z:node · (y≠z) ⇒
                  (Datagram x y (ttl+1) m) ⊗ (Listen z) ⊢ (Datagram x y ttl m)) ⊗ (Listen z).

Axiom TimeUp :    (Datagram x y 0 m) ⊢ 1.
```

Figure 2: *Coq code for the axioms specifying the internal behaviour of the IP layer.*

$$\frac{(DG\ m), LS \vdash LS \otimes (RC^0\ m) \qquad \begin{array}{c}(DG\ m), LS \vdash LS \otimes (RC^n\ m) \\ \vdots \\ (DG\ m), LS \vdash LS \otimes (RC^{n+1}\ m)\end{array}}{(DG\ m), LS \vdash LS \otimes (RC^n\ m)}\ \textit{Induction } n$$

Figure 3: *General structure of the inductive proof of the* `RecvDGClosure` *lemma.*

Implicitly, a datagram represented by any of the these three predicates has a valid header checksum. The two latter predicates are an abstraction of the "SEND" and "RECV" function of [9, §3.3].

The axioms that define the user interface to the IP layer are shown in figure 1, and may be paraphrased as follows:

`SendDG` Sending a message adds a single datagram to the system.

`LoseDG, DuplDG` A datagram in transit can be lost or duplicated.

`RecvDG` If a datagram addressed to node y exists and node y is listening for it, then node y will receive the message m or some corrupted version *corrupt* m of the message m. It should be noted that since *Listen* y occures in both the antecedent and the consequent of this axiom the listening state of node y is preserved.

5 Specification of IP Layer Behaviour

The internal behaviour of the IP layer can be specified by the axioms in figure 2, and described as follows:

`ClockTick` The *Clock* proposition is generated by a source once every second; this axiom asserts that such a clock-tick decrements the time-to-live field of a datagram.

`InterNode` When a datagram arrives at any node z other than the destination node y the time-to-live value is decremented.

`TimeUp` Datagrams with a zero time-to-live value are destroyed.

6 Verification of the IP Layer

Now that we have specified the IP layer interface, what can we prove about it's behaviour? The IP specification guarantees very little about a message sent from one node to another, since it may be corrupted, duplicated or even lost.

However little the IP specification guarantees, it does say:

- a message will not appear out of mid-air; and
- if you wait sufficiently long enough between sending two messages, and if both messages are received, then a correct ordering of the received messages is guaranteed.

Our specification of IP using linear logic allows us to prove these properties; the axiomatisation of linear logic in Coq allows us to verify that our proofs are correct.

6.1 No message appearing from mid-air

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.

Axiom `recvAfter` :
$\forall$ m:`message` $\cdot$ $\forall$ ttl, ttl^R:**nat** $\cdot$ $(ttl^R < ttl) \Rightarrow T_s(Send\ x\ y\ ttl\ m) < T_s(Recv\ x\ y\ ttl^R\ m)$.

Axiom `timeBound` :
$\forall$ m:`message` $\cdot$ $\forall$ ttl, ttl^R:**nat** $\cdot$ $(ttl^R < ttl) \Rightarrow T_s(Recv\ x\ y\ ttl^R\ m) < T_s(Send\ x\ y\ ttl\ m) + ttl$.

Figure 4: *Temporal Axioms for IP.* Here we assert that messages are received sometime *after* they are sent, and that a message can be received only within its time-to-live limit

---

Variable m_1, m_2 : `message`; ttl_1, ttl_1^R, ttl_2, ttl_2^R : **nat**.

Hypothesis `ttlDec1` : $(ttl_1^R < ttl_1)$.

Hypothesis `ttlDec2` : $(ttl_2^R < ttl_2)$.

Hypothesis `SendGap` : $T_s(Send\ x\ y\ ttl_1\ m_1) + ttl_1 < T_s(Send\ x\ y\ ttl_2\ m_2)$.

Theorem `boundedReliabilty` : $T_s(Recv\ x\ y\ ttl_1^R\ m_1) < T_s(Recv\ x\ y\ ttl_2^R\ m_2)$.
Proof Outline.
$T_s(Recv\ x\ y\ ttl_1^R\ m_1) < T_s(Send\ x\ y\ ttl_1\ m_1) + ttl_1 < T_s(Send\ x\ y\ ttl_2\ m_2) < T_s(Recv\ x\ y\ ttl_2^R\ m_2)$

Figure 5: *The "Ordering of Received Messages" theorem.* Here the assumptions `ttlDec1` and `ttlDec2` assert the natural property that time in transit decreases the time-to-live field, and `SendGap` asserts that the delay between sending m_1 and m_2 is greater than the time-to-live value of m_1.

Since, for the duration of this proof, the source, destination and time-to-live values are all fixed, we will adopt the following abbreviations:

- $(DG\ m)$ for $(Datagram\ x\ y\ ttl\ m)$.
- $(S\ m)$ for $(Send\ x\ y\ ttl\ m)$.
- $(RC\ m)$ for $(Recv\ x\ y\ ttl\ m)$.
- LS for $(Listen\ y)$.

There are at least two styles of proofs in linear logic; one which is applicable when there is an overall goal (such as planning), and the other when there is no overall goal but there is an evolution of state.

When there is an evolution of state with no overall goal the right-hand side of the judgement should be "empty", which is modelled by $\mathbf{0}$, the impossible goal. Then in order to show that some state Δ_0 can evolve into state Δ_1, we must prove that:

$$\begin{array}{c} \Delta_1 \vdash \mathbf{0} \\ \vdots \\ \Delta_0 \vdash \mathbf{0} \end{array}$$

Thus, the property that a receipt of any number n copies of a message m evolves from that message having been sent, can be expressed as the theorem:

Theorem `NoMidAirMsg` :
$\forall$ `m:message; n:nat` $\cdot$
`(LS, (RC`^{n} `m)` $\vdash$ `0)` $\Rightarrow$ `((SN m), LS` $\vdash$ `0)`.

Here we use $(RC^n\ m)$ as a shorthand for the receipt of n possible corrupted version of message m, which we can define inductively over conjunction as follows:

$$\begin{array}{rcl} (RC^0\ m) & = & \mathbf{1} \\ (RC^{n+1}\ m) & = & ((RC\ m) \oplus (RC\ (corrupt\ m))) \\ & & \otimes (RC^n\ m) \end{array}$$

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

Lemma `RecvDGClosure` :
$\forall$ `m:message; n:nat` $\cdot$ `(DG m), LS` $\vdash$ `LS` $\otimes$ `(RC`^{n} `m)`.

The proof of `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 Coq library 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 proof is given in figure 3; the base and inductive cases are given in figure 6.

6.2 Ordering of received messages

The other property that we can prove relating to IP concerns the timing of *Send* and *Recv* operations.

Let $T_s(E)$ be a function that return the time of an event E as seen by the *sender's* clock, where events can be either *Send* or *Recv* operations. Then

- A corollary of theorem `NoMidAirMsg` is that before a message is received it must have been sent, and therefore the time a message is sent (as seen by the sender's clock) is less than the time the message was received (as seen by the sender's clock).
- In addition axiom `TimeUp` leads us to infer that the time a message is received at, as seen by the sender's clock, is less than the time the message was sent at plus its time-to-live value.

Both of these properties may be expressed as axioms, as shown in figure 4

If two messages m_1 and m_2 are sent and successfully delivered, then the precondition necessary to guarantee that m_1 arrives before m_2 is that the interval n between sending m_1 and m_2 is greater than m_1's time-to-live value. This property is expressed in Coq's vernacular as shown in figure 5. Here ttl_1 and ttl_2 are the time-to-live values when message m_1 and m_2 are sent, and ttl_1^R and ttl_2^R are the time-to-live values when these messages are received.

7 Conclusions

This paper demonstrates that linear logic can be used in the specification and verification of the essential properties of distributed systems. In particular we believe that this represents a viable alternative to traditional proof-theoretic approaches using classical or temporal logics.

The results presented above are part of an ongoing collaboration [5] between the authors in the general area of formal methods and communication protocols. Specifically it is hoped that this work can now be used as a case study and basis for the specification and verification of more complex distributed systems.

References

[1] C. Cornes et al. The Coq proof assistant reference manual. Rapport Technique 177, INRIA, July 1995.

$$\dfrac{\dfrac{\dfrac{}{LS \vdash LS}\ Id \quad \dfrac{}{(DG\ m) \vdash \mathbf{1}}\ LoseDG}{(DG\ m), LS \vdash LS \otimes \mathbf{1}}\ \otimes_R}{(DG\ m), LS \vdash LS \otimes (RC^0\ m)}\ Unfold\ RC^n$$

$$\dfrac{\dfrac{}{(DG\ m) \vdash (DG\ m) \otimes (DG\ m)}\ DuplDG \quad \dfrac{\dfrac{}{(DG\ m), LS \vdash LS \otimes (RC\ m)}\ RecvDG \quad \dfrac{\dfrac{\dfrac{}{(RC\ m) \vdash (RC\ m)}\ Id \quad \dfrac{}{(DG\ m), LS \vdash LS \otimes (RC^n\ m)}\ Ind.\ Hyp.}{(DG\ m), LS \otimes (RC\ m) \vdash LS \otimes (RC\ m) \otimes (RC^n\ m)}\ \otimes_R}{(DG\ m), LS \otimes (RC\ m) \vdash LS \otimes (RC^{n+1}\ m)}\ Unfold\ RC^n}{(DG\ m) \otimes (DG\ m), LS \vdash LS \otimes (RC^{n+1}\ m)}\ Cut\ LS \otimes (RC\ m)}{(DG\ m), LS \vdash LS \otimes (RC^{n+1}\ m)}\ Cut\ (DG\ m) \otimes (DG\ m)$$

Figure 6: *Proof of the* `RecvDGClosure` *lemma.* Here we show the base and inductive case of the lemma; some of the context-manipulation details (dealing with e.g. commutativity and associativity) have been omitted for clarity.

[2] A. Felty, D.J. Howe, and F.A. Stomp. Protocol verification in Nuprl. In Al.J. Hu and M.Y. Vardi, editors, *Computer Aided Verification, 10th International Conference*, pages 428–439, Vancouver, BC, Canada, June 1998.

[3] J-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

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

[5] D. Gray, G. Hamilton, D. Sinclair, P. Gibson, and J. Power. Four logics and a protocol. In S. Flynn and A. Butterfield, editors, *3rd. Irish Workshop in Formal Methods*, NUI, Galway, July 1999.

[6] S. Kalvala and V. de Paiva. Mechanizing linear logic in Isabelle. In L.C. Paulson, editor, *Proceedings of the Isabelle Users Workshop*, Cambridge, England, Sept 1995.

[7] L.C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.

[8] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[9] J. Postel, editor. *RFC 791, Internet Protocol*. Defense Advanced Research Projects Agency, Sept 1981.

[10] J. Power and C. Webster. Working with linear logic in Coq. In Y. Bertot et al., editors, *12th International Conference on Theorem Proving in Higher Order Logics*, Nice, France, September 1999. (Work-in-Progress Report).