A knowledge-based algorithm for the Internet protocol TCP

Freek Stulp and Rineke Verbrugge
Cognitive Science and Engineering, University of Groningen
Grote Kruisstraat 2/1, 9712 TS Groningen, The Netherlands
E-mail: {freeks,rineke}@tcw3.ppsw.rug.nl

Abstract

Using a knowledge-based approach, we derive a protocol for the sequence transmission problem, which provides a high-level model of the Internet protocol TCP. The knowledge-based protocol is correct for communication media where deletion and reordering errors may occur. Furthermore, it is shown that both sender and receiver eventually attain depth n knowledge about the values of the messages for any n, but that common knowledge about the messages is not attainable.

1 Introduction

In their classical paper [5], Halpern and Zuck showed that epistemic logic enables perspicuous specification and verification for a number of protocols (like the alternating-bit protocol) that had been introduced for error-free transmission of sequences of messages over a distributed network. In particular, they introduced two knowledge-based protocols, A and B, that could solve the following problem. Let two processors be given, called the sender S and the receiver R. The sender has an input tape with an infinite sequence X of data elements. S reads these elements and tries to send them to R, which writes the elements on an output tape. The protocols are required to guarantee that (a) at any moment the sequence of data elements received by R is a prefix of X (safety) and (b) if the communication medium satisfies certain so-called fairness conditions, every data element of X will eventually be written by R (liveness). Fairness here means that infinitely many messages from S to R and from R to S are delivered, guaranteeing that every message arrives eventually.

It is easy to see that no protocol can guarantee these properties in an environment were deletion errors, mutation errors, and insertion errors may all occur. For, suppose that the symbols transmitted over the channel are 0, 1, and λ (where λ denotes that nothing is sent), and that the elements of the input sequence X are 0s and 1s. Now any sequence of messages in $\{0, 1, \lambda\}^*$ sent by S may be changed by the communication channel to any other sequence of the same length as the original.

Halpern and Zuck did however solve the sequence transmission problem for communication media where any two kinds of the above-mentioned errors occurred together. In order to do this, they used for each combination of two

errors a special encoding of messages ensuring unique decodability and error detection. Thus, the knowledge-based protocols A and B were implemented in different ways to solve the sequence transmission problem in different kinds of communication media. (See [5] or for more background [7, 4]).

In this paper it is our goal to use epistemic methods to specify and analyze a protocol that is actually used in today's technology: the Transmission Control Protocol (TCP). Because this protocol is indispensable for the Internet it is probably the most frequently used protocol around today. The epistemic analysis of the TCP will be done in much the same fashion as has been done with other protocols in the past. Before doing this we will have to abstract from irrelevant technical aspects. We will eventually acquire a knowledge-based protocol, represented by a simple algorithm. As we shall see, this algorithm beautifully demonstrates the windowing principle used by the TCP.

The analysis of this algorithm yields some interesting results. We will show that the depth of knowledge the sender and receiver can accumulate about messages sent is dependent upon the length of the tape and the position of information on the tape. If an infinite tape models the transmitted data, it can be shown that an n-fold depth of knowledge arises for any n on some position on the tape, although common knowledge can never be achieved. Another interesting aspect of the TCP is that it may almost be viewed as a generalization of protocol B - but not quite, as protocol B uses only a finite message alphabet and the knowledge-based protocol for TCP does not.

The rest of the paper is structured in the following way. Section 2 gives a short introduction to the Transmission Control Protocol and its role for the Internet. In Section 3, we present knowledge-based algorithms that model the TCP. Section 4 contains an epistemic analysis of these algorithms, giving bounds on the state of knowledge achieved by sender and receiver. Finally, Section 5 gives some conclusions.

2 The Transmission Control Protocol

In this section we will discuss the history of the Internet, and the role which the TCP plays in it. The birth of the Internet as we now know it goes back as far as 1969 [9]. It was then that the U.S. Defense sponsored the development of the Advanced Research Projects Agency Network (ARPANET). The ARPANET consists of four layers. The lowest one is called the Network Interface Layer and comprises the physical link between devices. The second is the Internet Layer, which insulates hosts from network-specific details. The Internet Protocol (IP) was developed for this purpose. The third layer, the Service Layer, is very important because it guarantees that packages are delivered. Two protocols were developed for the Service Layer. The TCP was introduced in 1973 and is used when a very reliable delivery is necessary. The User Datagram Protocol is used when the reliability requirements are not so high. The combination of the TCP and IP protocols, called TCP/IP, is so frequently used that they are almost always found together. In this article we will only discuss the TCP. The highest layer is the Process/Application Layer, which supports user-to-host and host-to-host processing. This layer includes applications such as Telecommunications Network (TELNET) and File Transfer Protocol (FTP).

The ARPANET was a network that made communication between differ-

ent computers and servers efficient, and, as soon became clear, viable for non-military uses. A number of ARPANET spin-offs were developed for this purpose. Collectively these spin-offs were called the Internet. Although there were differences, they all shared the TCP/IP and related protocols. ¿From this small U.S.-based network the global Internet as we now know it evolved.

The Internet introduces a lot of problems when it comes to the correct delivery of a package from one computer to the other. First of all it should facilitate communication between a wide variety of servers, operating systems, and Internet browsers. This part is taken care of by the IP. It also has to deal with the limited capacity a network might have, and possible deletion and reordering problems that may arise from overloading such a network. The TCP does not only deal with these problems, but it also provides efficient use of a network, adapting transmission speed to the network load.

3 The Knowledge-Based Algorithm

To give a knowledge-based explanation of the TCP we will first have to convert the technical Internet language to more knowledge-based terms.

The Internet sends Application Data (a *file*) from one computer to the other. Because this data is often too large to send as a whole, it is cut up into different segments. A TCP- and IP-header are attached to this segment, converting it to an IP-Datagram. These IP-Datagrams are sent over the network [3].

The Application Data can be thought of as a tape, which sender S wants to send to receiver R. Here, as an example, the tape contains the roman alphabet. This tape will not be sent as a whole, but is cut up into slots. In this example every slot contains one letter. A slot can be referred to by a natural number, called a position, which is a pointer to the element in that slot. Our *IP-datagram*, which we will call a package, only needs the actual data (a letter), and a *TCP-header* which will be the position of the slot the data came from. This will make our packages look like $(\mathrm{a}, 1)$,$(\mathrm{z}, 26)$ or (α, i), in which α is a variable that can be any letter of the roman alphabet. Acknowledgements are of the form $(\mathrm{ack}, 3)$ or $(@, 3)$, acknowledging the receipt of the third package. Packages will sometimes be referred to by either only their data or position.

In Halpern and Zuck's protocols A and B the sender sends one package, and just resends it until the receiver has acknowledged it [5, 7]. These protocols are reliable, but do not make use of the bandwidth of the Internet. The TCP uses the windowing principle to exploit the bandwidth: it sends a whole series of packages at once. Which series of packages are to be sent is determined by a *window* which is placed across the tape. In figure 1, the window is represented by a box, enclosing the first four packages. In this example packages 1,2,3,4 may be sent without waiting for any acknowledgements. After sending these four packages the sender waits a while for acknowledgements. If none are received all the packages in the window are sent again [3].

Let us suppose that sender S has sent the packages 1,2,3,4, but package 3 goes lost somewhere, as shown in figure 1. Receiver R now receives package 1,2 and 4. Because R knows that it has not yet received $(\alpha, 3)$ it will acknowledge package 2, and not 4. Package 2 is called the highest consecutive package received. After receiving $(\mathrm{ack}, 2)$ from R, S knows that is not necessary to send $(\mathrm{a}, 1)$ or $(\mathrm{b}, 2)$ again. This is why S will shift the window so that it no longer

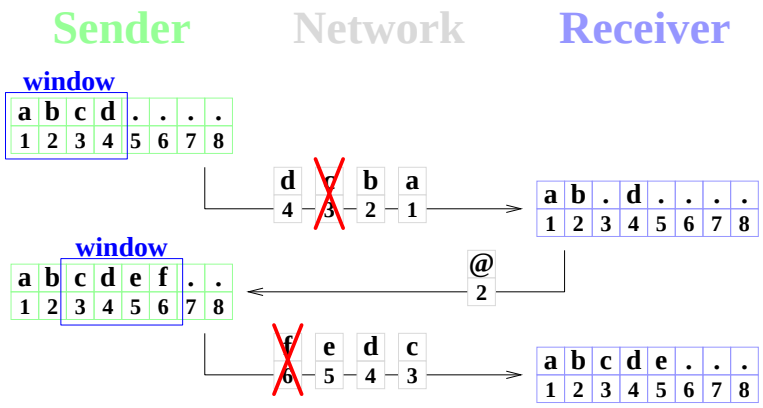

Figure 1: Windowing principle

contains 1 and 2. As a consequence the same window, of size four, now does contain packages 5 and 6, and of course still contains packages 3 and 4. S will now send the series of packages 3,4,5,6...wait...3,4,5,6 until an acknowledgement of one of the packages in the current window is received.

The algorithms that will be discussed in the next section make one important assumption about packages that have been received. This assumption is that every package that has been sent or received is stored in a knowledge base, and kept there *forever*. This is necessary to reason about packages that have been sent or received in the past. The Internet of course doesn't do this, otherwise your computer's memory would be overloaded very quickly. Once the Internet has retrieved the data from the IP-Datagram, the IP- and TCP-header become obsolete. Once the original Application Data has been reconstructed, all the individual packages can also be disposed.

3.1 The Algorithms

We have constructed a knowledge-based algorithm that models the TCP, and thus makes use of the windowing principle. First we will discuss the sender's algorithm, and then the receiver's algorithm.

In order to visualize how the knowledge-based algorithm of the TCP works we have constructed an applet which can be found on [10]. The applet link in the upper menu of this page will take you to an interactive applet that shows the algorithms at work. A small manual is added to explain the applet. We recommend the reader to use the applet to visualize what the rules of the algorithms do.

Before we discuss the sender's and receiver's algorithms we will explain what the variables and functions refer to.

`windowSize`	:	This is the size of the window.
`offset`	:	This is the position on the tape where the window begins.
`i`	:	This is a pointer to a position on the tape. The package with this position will be sent next, but only if it is in the window.

`bestReceived()` : This function returns a position. It checks the set of packages the sender has received so far, and returns the position of the package with the highest consecutive position.

`max` : The highest consecutive position the sender has received so far.

`lastAck` : This keeps track of the position of the highest consecutive package received. If the receiver has received packages 1,2 and 4, lastAck will be 2.

Sender's algorithm:

```
1  windowSize := 4;
   {Set the size of the window.}
2  offset := 0;
   {Set the first position of the window at the beginning of the tape.}
3  while true
         {After the initialization has taken place, start sending the tape.}
4        do i := offset;
         {Set pointer 'i' at the beginning of the window.}
5        while i ≠ offset + windowSize)
         {Check if 'i' is outside the window. If not, send package (α,i) in rule 6/7.}
6        do read(x,i);
            {Read the package with position 'i' from the tape.}
7           send(x,i);
            {Send the package you have just read.}
8           i := i + 1;
            {Set pointer 'i' to the next position on the tape.}
9           max = bestReceived();
            {Set 'max' to 'bestReceived()': the package with the highest
             consecutive position received. }
10          if (offset ≤ max)
            {Check if you have received a package with a position higher than your
             current window offset.}
11          then offset := max + 1;
                  {A package in your window was acknowledged (rule 10). Shift
                   your window.}
12                i := offset;
                  {After shifting your window, set 'i' to the beginning of the
                   window.}
13          end
            {End the window-shifting statements}
14       end
         {End the package sending statements}
15       waitlong
         {The network is busy. Wait a while}
16 end
   {While(true)-loop finished. Start again.}
```

Receiver's algorithm:

```
1  when received(x,0)
   {You can start the algorithm when the first package has been received.}
2  lastAck := 0;
   {You have just received the package (x,0), so set 'lastAck' to 0.}
3  while true
   {After the initialization has taken place, get ready to receive the tape.}
4  do write(x,lastAck);
      {Write the received package with position 'lastAck' to your output tape.}
5     while not(received(x,lastAck+1))
      {This rule determines if you have received the next package on the tape.}
6     do waitshort;
         {Wait a while before sending the next acknowledgement. If you don't,
          the network will overload.}
7        send(ack,lastAck);
         {Send an acknowledgement of the last received package.}
8     end
      {End the statements that send an acknowledgement.}
9     lastAck := lastAck+1;
      {You have received (x,lastAck+1), so 'lastAck' can now be incremented.}
10 end
   {While(true)-loop finished. Start again.}
```

3.2 TCP revisited

Now that we have a clearer picture of how the windowing principle works we can explain how the TCP exploits the bandwidth of a network. It might seem that the TCP uses the bandwidth of a network in a very blunt and naive way. It just dumps a large amount of packages on the network, hoping that some will arrive. The actual process is much subtler. The beautiful thing about the TCP is that it can vary its window size. The receiver determines how much room is left in its buffer to receive IP-datagrams. It determines what the optimal window size would be, and sends this along with its acknowledgements. In order to keep the knowledge-based algorithm transparent we have not treated this feature of the TCP.

Another aspect that is worth mentioning is that if a tape is very long, the position marker will become larger and larger. Your first package might be $(\alpha, 1)$, whilst a latter package might be $(\alpha, 1000)$. The first position requires one bit, the latter ten. Luckily, our algorithm has infinite computing capacity. But the Internet does not. The TCP-Header of the IP-Datagram always reserves 32 bits to indicate the sequence number, our position. This allows more than $4 * 10^9$ sequence numbers to be generated, which is sufficient in practice. On the total scale of an IP-Datagram, these 32 bits are not that substantial.

4 Epistemic analysis of the TCP protocol

We will first describe some choices we made in modeling the sequence transmission problem for the Internet.

An important aspect of the model of sequence transmission given by Aho and others and analyzed in Halpern and Zuck's paper, is that they assume message

transmission to proceed in synchronous clocked rounds. One may think of these rounds as consisting of three consecutive phases: a send phase, a receive phase, and a local computation phase. In their model, messages are received in the same round as they are sent, if they are received at all, so reordering problems do not appear [2]. This model is not adequate for studying TCP, where a global clock is an unlikely assumption and reordering is one of the most important problems to be solved. The model was relaxed by Halpern and Zuck to include asynchronous systems, where S and R perform an action only when they are scheduled. In order to assure liveness, it is assumed that S and R are scheduled infinitely often [5]. We adopt this extension to asynchronous systems.

Like Halpern and Zuck, we also extend the model by Aho and others by allowing messages to come from an alphabet larger than $\{0, 1, \lambda\}$. We even assume that the strings for (α, i) and (ack, i) are distinct for every value of i, so either we need an infinite alphabet or the strings grow longer as i becomes larger. This assumption is necessary to solve the sequence transmission problem in communication media where reordering is possible. For example, suppose that messages $(\text{a}, 10)$ and $(\text{a}, 1000)$ are represented by the same string and R receives $(\text{a}, 1000)$ just after sending its acknowledgment about $(\text{b}, 999)$. Then, due to possible reordering problems, R will not be able to decide whether it is a new message or an overly late version of $(\text{a}, 10)$.

Correctness results

The algorithms implementing the knowledge-based protocol in the applet can solve the sequence transmission problem in communication media where deletion errors and reordering errors, but no other kinds, occur. Formally, this can be proved using a semantics of interpreted systems I consistemt with the knowledge-based protocol.

Theorem 1 *Let I be an interpreted system consistent with the knowledge-based protocol given in section 4. Then every run of I has the safety property and every fair run of I has the liveness property.*

See the introduction for definitions of the notions of safety, fairness, and liveness. Intuitively, safety for the TCP is obvious since R writes a data element only if it knows its value. A formal proof of such a correctness result is still quite long and complicated, however, and we do not give it here (see the journal version of [5] for similar proofs).

Comparison with protocol B

As mentioned in the introduction, TCP could inexactly be viewed as a generalization of Halpern and Zuck's protocol B, by setting the window-size to 1. As a reminder, knowledge-based protocol B is given here in a presentation similar to the one in [7]:

Protocol B Sender's algorithm:

```
1  i := 0;
2  while true do
3        begin read(x_i);
              {Read the package with position 'i' from the tape.}
4             send(x_i); ''K_S K_R(x_{i-1})'' until K_S K_R(x_i);
              {Send a combined message of the data element you have just read
               and an acknowledgement of R's last acknowledgment (none if i=0).}
5             i := i + 1;
              {Set pointer 'i' to the next position on the tape.}
6        end
```

Protocol B Receiver's algorithm:

```
1  when K_R(x_0) set i:=0;
   {You can start the algorithm when the first data element has been received.}
2  while true do
   {After the initialization has taken place, get ready to receive the tape.}
3        begin write(x_i);
              {Write the received data element with position i to your output tape.}
4             send ''K_R(x_i)'' until K_R(x_{i+1});
              {Send an acknowledgement of the last received package.}
5             i := i + 1;
6        end
```

In this special case where TCP operates with window-size 1, it will send one package at a time and the window may only be shifted if the acknowledgement for this package has been received, exactly as in protocol B.

However, there is an important difference between the two algorithms as well. When implementing protocol B, a finite message alphabet is used so that e.g. the "$K_S K_R(x_{i-1})$" messages are not distinct for all values of i. For protocol B, which is meant to work in environments where reordering problems do not occur, this does not present a problem. Afek and others have shown, however, that protocols using only a finite message alphabet can never solve the sequence transmission problem in environments where both reordering and deletion errors may occur, see [1]. Thus it is essential that our knowledge-based protocol for TCP uses an infinite message alphabet (or messages growing in size).

Accumulating knowledge

The theorem below holds in communication media where there may be deletion and reordering errors, but no other kinds. In particular, there should be no undetected mutation and insertion errors. The theorem says essentially that, using the TCP, the sender and receiver accumulate more and more knowledge about messages sent a long time ago. For example, after the window has moved two complete window sizes, the sender knows that the receiver knows that the sender knows that the receiver knows that the sender knows that the receiver knows that the sender knows the first data element, or formally $K_S(K_R K_S)^3(\alpha, 1)$. This result is different from Halpern and Zuck's analysis of the protocols A and B, where only a depth 4 knowledge of the form $(K_S K_R)^2(i)$ was shown to hold (see the conference version of [5]).

Definition 1 *We use the following abbreviations:*
$K_R(\alpha, n)$ *stands for "R knows that the n-th data element is* α*";*
similarly for $K_S(\alpha, n)$.
$K_R(n)$ *stands for "R knows the value of the n-th data element";*
similarly for $K_S(n)$.
$\Box\varphi$ *stands for "*φ *holds now and at all moments in future", i.e.* $\Box\varphi \Leftrightarrow \varphi \wedge G(\varphi)$.

Theorem 2 *Let R be any set of runs where:*

- *The safety property holds (so that at any moment the sequence Y of data elements received by R is a prefix of X, the infinite sequence X of data elements on S's input tape);*
- *S's state records all data elements it has read and acknowledgements it received;*
- *R's state records all the data elements it has written.*

Let w be the window-size. Then for all runs in R and all $n \geq 0, i \geq 0$ *the following hold:*

R *R writes* $(\alpha, i + n * w) \rightarrow \Box(K_R K_S)^{n+1}(\alpha, i)$.

S *S receives* $(\text{ack}, i + n * w) \rightarrow \Box K_S(K_R K_S)^{n+1}(i)$.

Before we give a formal proof, we will try to give an intuitive feel of the knowledge the sender and the receiver can accumulate about each other with respect to the packages. First of all S will know a package (α, i) as soon as it has been read from the tape $(K_S(i))$. R knows a package (α, i) when it is received $(K_R(i))$. A package received by R must have been read by S, so R also knows that S knows the package $(K_R K_S(i))$. When S receives an acknowledgement of a package, S knows that R has received it. The sender can deduce $K_R K_S(i)$ from this (thus $K_S K_R K_S(i)$).

Now we reach a more interesting case. We will show what R can deduce once it receives a package $(\alpha, i+w)$, in which w is the window-size. R will reason like this. If S has sent $(\alpha, i+w)$ then its window must contain $(\alpha, i+w)$. A window with size w that contains $(\alpha, i + w)$ cannot also contain (α, i). Apparently S has already shifted its window past (α, i). S would only have done this if it has received (ack, i) from R. Since R now knows that S has received (ack, i), and that (as we showed above) S would deduce $K_S K_R K_S(i)$ from this, R can deduce $K_R K_S K_R K_S(i)$ It also works the other way around. Once S receives $(\text{ack}, i + w)$ it knows that R has received $(\alpha, i + w)$. S knows that R will deduce $K_R K_S K_R K_S(i)$ from this package (as above), thus $K_S K_R K_S K_R K_S(i)$. Thus, the further the tape-transmission progresses, the more information can be deduced about all parts of the currently read tape.

Proof We prove the theorem by induction on n. In the proof, we freely use two general principles. First, $P(\Box\varphi) \rightarrow \Box\varphi$ (from tense logic); Second, R and S are assumed to store all relevant information from their message history. So, if R knows a positive modality (like $K_S K_R K_S$) about data elements now, it will know it always in future, i.e. $K_R(\varphi) \rightarrow \Box K_R(\varphi)$ for appropriate φ, and similarly for S.

n=0 It is clear that in general, we have the following, because S's state records all data elements it has read:

$$S \text{ sends } (\alpha, i) \rightarrow \Box K_S(\alpha, i)$$

R knows the above fact. Now if R receives a data element from S it knows that S has sent it sometime in the past (for there are no undetected mutation and insertion errors), which implies that $K_R(P\Box K_S(\alpha, i))$, which in turn implies by our two general principles that $\Box K_R K_S(\alpha, i)$. Thus, we have

$$R \text{ writes } (\alpha, i) \rightarrow \Box K_R K_S(\alpha, i),$$

which is the **R**-part of the theorem for n=0.

Because R sends (ack, i) only if it received the i-th data element sometime in the past and because of the first general principle, the following holds:

$$R \text{ sends } (\text{ack}, i) \rightarrow \Box K_R K_S(i).$$

If S receives an acknowledgement, it knows that R has sent it in the past, thus because S knows the above fact, we derive the following by the two general principles:

$$S \text{ receives } (\text{ack}, i) \rightarrow \Box K_S K_R K_S(i),$$

which is exactly the **S**-part of the theorem for n=0.

induction step Suppose as induction hypothesis that **R** and **S** hold for $k-1$, where $k \geq 1$. We will prove that **R** and **S** hold for k itself.

Because S only moves its window forward after it has received acknowledgements about all data elements in the window, we have the following:

$$S \text{ sends } (\alpha, i + k * w) \rightarrow P(S \text{ receives } (\text{ack}, i + (k-1) * w)).$$

We may combine this fact with the **S**-part of the induction hypothesis and the first general principle to derive:

$$S \text{ sends } (\alpha, i + k * w) \rightarrow \Box K_S (K_R K_S)^k(i).$$

R knows the above fact. Now if R receives a data element with position marker $i + k * w$ from S, it knows that S has sent it sometime in the past which implies by the above fact and our two general principles that $\Box K_R K_S (K_R K_S)^k(i)$. Thus, we have

$$R \text{ writes } (\alpha, i + k * w) \rightarrow \Box (K_R K_S)^{k+1}(i),$$

which is exactly the **R**-part of the theorem for n=k.

As in the base case, R sends an acknowledgement about the $i + k * w$-th data element only if it received that element in the past, so we derive by our first general principle:

$$R \text{ sends } (\text{ack}, i + k * w) \rightarrow \Box(K_R K_S)^{k+1}(i).$$

S knows the above fact, so if it receives an acknowledgement about the $i + k * w$-th data element, it knows that R has sent this in the past, so by the two general principles we conclude:

$$S \text{ receives } (\text{ack}, i + k * w) \rightarrow \Box K_S(K_R K_S)^{k+1}(i),$$

which is exactly the **R** part of the theorem.

We assumed from the start that the input tape is infinite and that infinitely many messages from R to S and from S to R are delivered. Thus, the above theorem shows that for any n and any message, depth n knowledge of that message will eventually be reached. The next subsection shows that common knowledge of the message remains nevertheless out of reach.

Negative results: no common knowledge

Even though for any n, an n-fold depth of knowledge about messages among R and S is eventually realized, the two processors will never attain common knowledge about anything that they did not mutually know from the outset. In fact, this follows from some general results by Halpern and Vardi [6], using a representation for traces by Van der Meyden [8]. First let us adapt a classical definition. A trace (or run) $s_1, s_2, \ldots$ of global states is called *non-simultaneous* if for every transition from s_i to s_{i+1}, there exists a processor (from R, S) for which the two global states are indistinguishable.

Lemma 1 (Halpern and Moses 1990) *In non-simultaneous runs, common knowledge is constant.*

Lemma 2 (Halpern and Moses 1990, Van der Meyden 98) *In an environment where there is no common knowledge about the maximum time delay with which messages may arrive, there cannot be a gain in common knowledge.*

Theorem 3 *In an asynchronous environment, no common knowledge about the values of messages can be attained by any protocol.*

Proof In our case, assuming that there is no global clock, states are best represented by the history of messages read and sent by the two processors, with consecutive repetitions deleted. It is easy to see that two consecutive states in such a trace are always indistinguishable for R or for S: if one processor has just sent a message or an acknowledgement, it does not know whether the other one has received it yet. Thus, the relevant runs are non-simultaneous. Now the theorem follows immediately from the previous two lemmas, assuming the natural condition that there is no common knowledge about the maximum time delay with which messages may arrive.

5 Conclusions

In this article we have shown that a real-life protocol such as TCP can be modelled by knowledge-based algorithms. These algorithms can be analyzed to determine the robustness of the protocol. Can mutation, deletion or insertion errors be handled by this protocol? If not, what are the practical consequences? The User Datagram Protocol, mentioned briefly in section 2, might yield very different results, as it has not been built to guarantee a perfect delivery.

In the future this approach could perhaps be used in designing and implementing new protocols, for the Internet, or for any new technology requiring electronic data transmission. Using these logical tools, correctness and robustness of real-life protocols can be verified.

As to further research, it would be interesting to investigate whether the correctness results of section 4 may be extended. We conjecture, for example, that an environment with any two kinds of deletion, mutation, and insertion errors can be handled by implementing the knowledge-based algorithm for TCP using encodings similar to [5].

Acknowledgements

We would like to thank Egon Baars and Jeroen Meijer who (together with author Freek Stulp) were contributors to the student project which inspired the writing of this article. Also thanks to Bill Bitner who answered some of our questions about TCP as it works in practice.

References

[1] Y. Afek, H. Attiya, A. Fekete, M. J. Fischer, N. Lynch, Y. Mansour, D. Wang and L. D. Zuck, Reliable communication over unreliable channels, *Journal of the ACM*, Vol. 41, No. 6 (1994), pp. 1267–1297.

[2] A.V. Aho, J.D. Ullman, A.D. Wyner, and M. Yannakakis, Bounds on the size and transmission rate of communication protocols. *Computers and Mathematics with Applications*, vol. 8, nr. 3 (1982), pp. 205-214.

[3] B. Bitner and R. White, *Care and Feeding for better VM TCP/IP Performance*, IBM Endicott, 1992.
http://vmdev.gpl.ibm.com/dEVPAGES/Bitner/presentations/tcpip/ipcare.html

[4] R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi, *Reasoning about Knowledge*, Cambridge (MA), MIT Press, 1995.

[5] J.Y. Halpern and L.D. Zuck, A little knowledge goes a long way: simple knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM* 39, nr. 3 (1992), pp. 449-478. Earlier version appeared in: *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, 1987, pp. 269-280.

[6] J.Y. Halpern and Y. Moses, Knowledge and common knowledge in a distributed environment, *Journal of the ACM* 37, nr. 3 (1990), pp. 549-587.

[7] J.-J. Ch. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science*, Cambridge, Cambridge University Press, 1995.

[8] R. van der Meyden, Common knowledge and update in finite environments, *Information and Computation* 140, No. 2 (1998), pp. 115-157.

[9] M. Miller, *Troubleshooting TCP/IP: Analyzing the Protocols of the Internet*, San Mateo (CA), Prentice-Hall, 1992.

[10] F. Stulp, Visualisation of the Transmission Control Protocol, http://tcw2.ppsw.rug.nl/~freeks/tcp/