INTRODUCTION

Protocols are strict languages used for communicating reliably between two compatible nodes. Each node must know how to interpret and send signals in accordance with the rules of a particular protocol. Some protocols are very simple but do not offer robust, reliable communication. For a protocol to be reliable it must send data with no loss or duplication. Data must also be delivered in the correct order and within a reasonable timescale [1].

The English language is an example of a type of protocol, used to communicate between people who understand the language. In most circumstances, the language is highly effective, however breakdowns do occur - albeit infrequently. A protocol should not contain any ambiguities and as such they are becoming increasingly complex.

One of the first practical applications of protocols was merely for sending documents reliably to a printer via a cable. This used a basic error checking system known as parity. Parity is successful in detecting errors with a bit error rate of 0.125 (1 in 8) or less, it is said that parity has a Hamming distance of 2. This is effective in theory but in practice errors do not generally occur a single bit at a time. Losses of signal quality mean most errors usually occur in bursts. Parity then only has a 50% success rate [2] in detecting such errors. Protocols were then developed for communication between computers. One of the most basic protocols is the Aloha Protocol. Aloha is the Hawaiian translation for hello. With this simple protocol, computers are involved in half duplex transmission (half duplex is the term for communications that occur in both directions but not at the same time). When data need to be sent, the sending computer sends the data without establishing whether the other computers are ready. Problems are encountered when two computers wish to transmit data at the same time as shown in figure 1.

To allow a more reliable method of transfer, each segment of data contains a Cyclic Redundancy Check. This is a

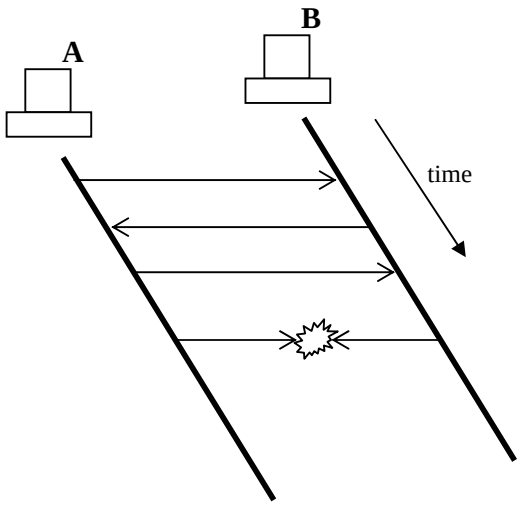

Figure 1: Collision in the Basic Aloha Protocol

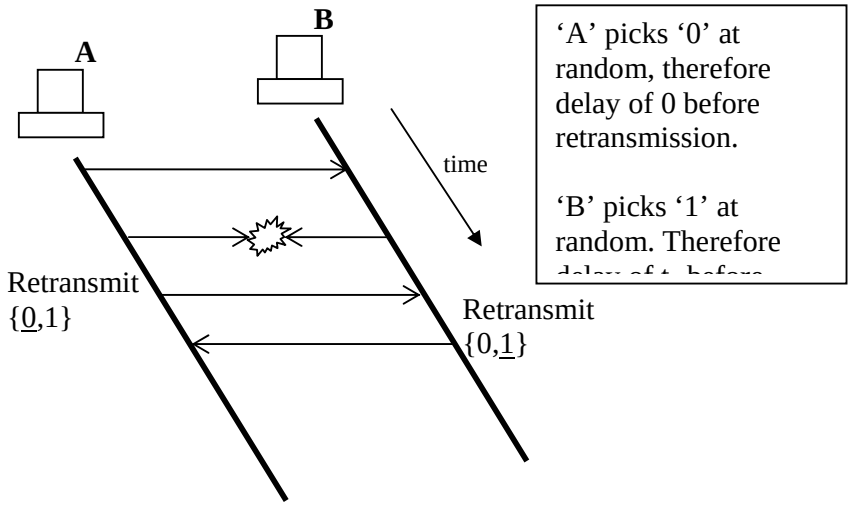

Figure 2: Retransmission in the Basic Aloha Protocol

method of testing data using very simple logic to detect errors. Typically, a CRC algorithm can detect over 99.99% of errors [3]. In the Aloha Protocol, once an error has occurred due to a collision, retransmission is necessary. To stop both nodes immediately retransmitting and causing further collisions, a set of two numbers is created and one of these numbers is chosen at random. This number depicts the delay before retransmission as shown above in figure 2.

Due to the random retransmission, both computers may then again send at the same time. If this occurs, then the set of numbers is doubled and a number chosen at random from this new set. This process is known as Exponential Back-off and is illustrated in figure 3 below.

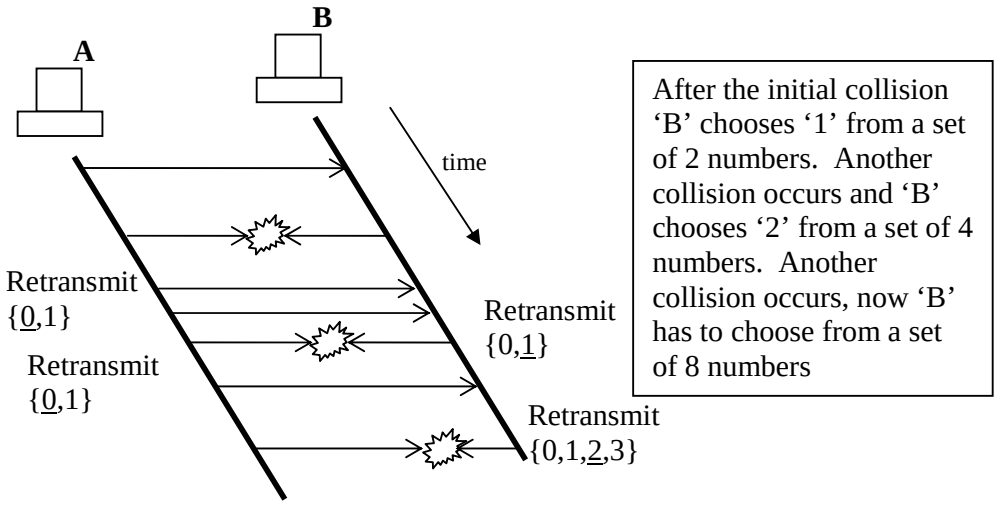

Figure 3: Exponential Back-off in the Basic Aloha Protocol

This creates a big problem for 'B' as if A' has lots of data to send since, if the medium is used by other computers, then further collisions will occur. 'B' has to choose from larger number sets resulting in transmissions from 'B' being delayed. In this case, a more advanced protocol is required.

STANDARD DESIGN OF PROTOCOLS

Modern protocols have been designed through experience and correction of known problems in previous protocols; in fact the process has been described as "design by correction" rather than "correct by design" [4]. Internet protocols are studied by the Internet Engineering Task Force, IETF. The IETF have meetings quarterly and proposals in the form of Request for Comments (RFCs) are discussed. There are three main types of RFC. The first, an Internet Draft, is a proposal for a new standard or modification to an existing one [5]. There are Informational and Experimental RFCs which outline a particular area already standardised or an area undergoing research. If this document is approved, then an official RFC is published. One of the most important RFCs published to date is the "Requirements for Internet Hosts -- Communication Layers" [6]. This defines the standards for protocols at layers 2, 3 and 4.

TCP PROTOCOL

One of the most used protocols for reliable communication in modern computing is the Transmission Control Protocol or TCP. TCP is part of the OSI Reference model, which is an official standard that allows computers to communicate using a number of protocols that have been split into seven layers. Each layer provides a different service for communication. One of the most studied layers is the fourth, known as the transport layer. This:

"provides transparent transfer of data between systems, relieving upper layers from concern with providing reliable and cost effective data transfer; provides end-to-end control and information interchange with quality of service needed by the application program; first true end-to-end layer." [7]

TCP is implemented at this layer.

TCP is a connection-oriented protocol [8] meaning that the two computers communicating must first establish a connection. This will be discussed in more detail later.

TCP HEADER

To allow protocols to be uniquely identified, they require a section before

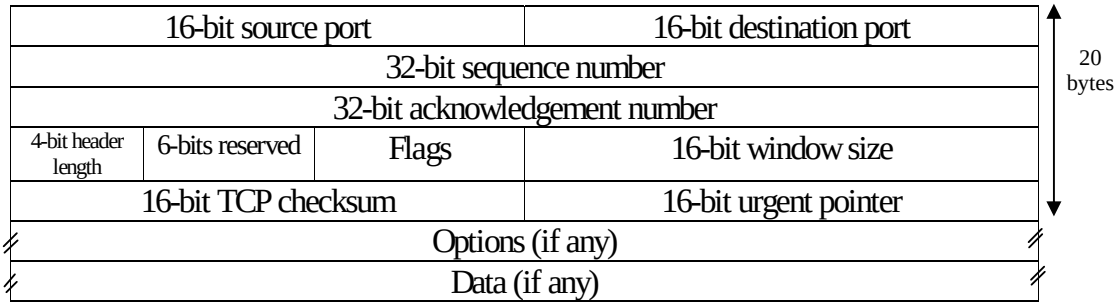

Figure 4: TCP Header [8]

the data known as a header. In this header, protocols also provide information about the subsequent data. A typical TCP segment has a header as detailed in figure 4 below.

Each section of the header is briefly discussed below:

- **16-bit source port number.** This is the IP address of the sending computer (note, this is in conformance to the IPv4 standard. IPv6 requires a 32-bit IP address)
- **16-bit destination port number.** This is the IP address of the intended recipient (again this is in conformance to IPv4, IPv6 requires 32-bits)
- **32-bit sequence number.** This is a number, which is incremented by the number of data bytes sent. This ensures that all data is received in order and without duplication.
- **32-bit acknowledgement number.** This contains the number of the next expected sequence number to be received from the other client in communication.
- **4-bit header length.** This is total length of the TCP header in 32-bit words. This allows the use of options, but is usually set to 5 (20 Bytes).
- **Flags.** 6 bits indicating type of segment. These are discussed in more detail below.
- **16-bit window size.** This informs the other client how much data can be received in the next segment. This decreases if data is not processed and remains in a buffer. One of the most commonly used options is window scaling, which allows much larger window sizes to be advertised. Typically, a modern operating system has a default of 16 or 32 Kbytes for its maximum window size.
- **16-bit TCP checksum.** This is an error detection field that must be calculated and set for every segment. This is used for checking any errors in both the header and the data.

TCP FLAGS

The flags are used for defining the type of segment to be transferred. Types of segment are used either for establishing or terminating connections, or, sending or acknowledging data. As mentioned earlier, TCP requires connections to be set up. To do this the inclusion of a SYN flag has been implemented. This allows two computers to synchronise to each others sequence numbers. The ACK flag acknowledges receipt of data and allows a connection to be established. This must be done in both directions to establish a link.

Data can be sent using the PSH flag, although this is not required. The correct use of the PSH flag is for passing data to the OS as soon as possible, but most implementations of TCP use this for all data. Again, an ACK flag is used for acknowledging receipt of data. If data is received corrupted or out of order then another ACK is sent, not acknowledging this data but as a label for warning. “the data was not reliable, please resend”. An URG flag is used for urgent data, the operating system then looks at the urgent pointer and adds this to the sequence number to obtain the last byte of data that is flagged as urgent.

Once the connection is established and all data has been transferred, it is required to close the connection to free resources. This can be done in two ways. The standard way is to issue a FIN flag, which requests the connection to be closed. On receipt of

an ACK, the connection is closed. As with the connection-establishment, the termination must be done in both directions to completely close the connection. An abnormal method of terminating a connection is to send an RST flag. This causes a reset, and does not require both directions to close. This was implemented in the case of computer crashes but is misused by the new HTTP/1.1 protocol [9].

TCP TIMERS

In order for the protocol to be reliable, it must also deliver the data in a reasonable timescale. TCP includes three timers to ensure this occurs. These are the retransmission, persist, and keepalive timers.

The retransmission timer is used as a basic time-out counter. If no acknowledgement of data is received within this counter then data is retransmitted as it may be undelivered. Usually, the first timeout is set to 1.5 seconds [8], and then uses exponential back-off until 64 seconds is the delay between retransmissions. The default number of retransmissions is usually set to 12 before the connection is reset. To avoid data loss through network congestion, the protocol initiates a slow-start phase which gradually increases throughput until an optimal value is achieved.

The persist timer is used to find out the window size of another client. Suppose computer A wishes to send data to B, but B advertises a window size of 0, indicating its buffer is full and cannot receive any more data temporarily. A continues to send segments with no data to B to establish when it is ready to receive more data again. A persists in trying to send data to B.

Finally, a keepalive timer is used for connection that remain open for long times. If a connection remains open and idle for a period, usually two hours, then a keepalive probe is sent. There are four possible outcomes from this scenario, either the computer is still there, is unreachable, has crashed, or has crashed and rebooted. The protocol takes necessary actions as to what to do in each case.

PROBLEMS WITH TCP

The design of the TCP protocol was not achieved overnight, it has had many revisions since its original specification [10]. These revisions have been ‘designed’ by the RFC updates through the IETF. Until recently, this was the preferred method of protocol design; getting a group of experts in the field and formulating a design with their combined knowledge and past experience. It has been stated that if 5% of the protocol is used 95% of the time then concentrate on this part of the design. Some engineers think that if the protocol crashes and a re-connection is required, this is fine. What if the computer requires a reboot due to the protocol failing? Or in a more extreme circumstance, what if a server needs rebooting? Where is the line drawn as to what is a reasonable failure within a protocol? Different applications require a varying amount of reliability. The complex nature of the TCP protocol leads to design problems. It is not possible to test every set of data, every flag, every time-out so how does one verify the design?

FORMAL SPECIFICATION

When an engineer is initially presented with a design, there are often many problems in the interpretation of the requirements. Usually, a formal specification will be proposed before the design commences. This document will dispose of all platitudes,

ambiguities, inconsistencies and omissions [11]. Platitudes are vague statements where no real information can be deduced. Ambiguities occur when statements have multiple meanings and it is unclear as to which is intended. Inconsistencies are contradictory statements which cannot be met due to other requirements. With a formal specification constructed, it is possible for the design to commence and checked to ensure the requirements are met.

VALIDATION AND VERIFICATION

Validation and Verification is the process of comparing a design to the formal specification. This is carried out at regular stages of the design to ensure problems are not encountered. Verification is often thought of as, "Are we building the system right?" in conformance with the formal specification. This stage is often completed using module and integration testing, allowing the design to generally follow the requirements. Suppose all modules are tested individually and they work as intended, they are then tested when integrated together to ensure correct operation is still achieved. Validation is often referred to by, "Are we building the right system?" This is often carried out by using acceptance tests, usually a mathematical means of testing areas of design. This may be by exhaustive methods if the size of design allows it, but generally this is not the case.

It is this process of validation and verification throughout the design that produces a working design. Methods of incorporating this and analysing each stage with formal verification can be used to prove a design is exactly as specified in the formal specification. The design would then be 100% correct at each stage of the process, and would lead to a completed design being provably correct, hence correct-by-construction.

CORRECT-BY-CONSTRUCTION

Correct-by-construction is the term given to a design which is provably correct by using certain methods during the whole of the design process. Formal methods of verification are used at every possible step to ensure the formal specification of the design is adhered to. There are many different techniques of formal verification available for use. Such methods use exhaustive approaches, highly mathematical techniques or complex software and logic depending on the type of design in question. Formal methods have been used for many years in different fields of engineering. Digital Very Large Scale Integration, VLSI, has been using these techniques for over 10 years and one major CAD company [12] in the field has implemented this into their latest release of software. Techniques for specification allowing easier verification use formal ordinary mathematics and temporal logic. Temporal Logic of Actions, TLA [14], was developed as a variation of these techniques to enhance the design stages. These techniques have recently been used in many software designs to achieve better results. There are also some languages which assemble designs on previously proven algorithms [13]. The most used method however is temporal logic verification which is extremely thorough in its verification. Some designers think this process offers too robust a design, which does not make efficient use of resources. In the application of internet protocols, this is undesirable. The internet is slow and utilisation of communication media should be maximised, however, in a

safety-critical application this may not be the case. Such a safety-critical system may be a fly-by-wire system used in military aircraft. This system is used to estimate what a pilot desires when turning his flight control left. The system has been very successful and is soon to be introduced into the automobile market. Fly-by-wire and other safety-critical systems often have large bandwidths in comparison to the amount of utilisation they require and with memory and processing power being extremely cheap, mean that there is an application for correct-by-construction protocols. It is also just a matter of time until a high-level language is developed for implementation of such protocols; one upcoming validation language for protocols is PROMELA [15]. When good tools are available, it will be possible to produce more efficient protocols that are correct-by-construction, but there are additional problems with this method of design.

One of the major disadvantages of correct-by-construction design is cost. Formal methods of validation and verification are very laborious and occupy a large percentage of the design timescale. Correct-by-construction requires these formal methods to be applied at frequent stages throughout the design process. This equates to a very high consumption of resources in the design. To combat this disadvantage, one area of engineering that is easily validated using formal methods is the design of finite state machines. Protocols, even complex ones, can be thought of as a finite state machine, FSM. This type of design requires a much more in-depth formal specification but once this document is complete, the design should be straightforward and validation easily applied.

Correct-by-construction, however, does have one major flaw that may never be eliminated. This problem lies in the proof of a design being validated 100% at any stage. If this validation process uses any software (which is probable as a degree of automation is required for sufficiently large designs) then the software may contain bugs. There has been an instance [16] when a research company had produced high-level protocol development software. This software was designed using correct-by-construction techniques to prove that it would be provably correct. When designing a protocol with this software, the protocol was found to have errors and did not work. In this case, it was found that the C compiler being used contained bugs.

From this, it can be seen that there are many other aspects to take into consideration when declaring some design ‘provably correct’. What if the C compiler in the above example was designed using correct-by-construction techniques and did not contain any bugs? The processor may then contain bugs, but can the processor use this design process? Processor design is carried out using synthesis software which accepts a high-level language such as VHDL or Verilog as a specification. A compiler will then produce the layout for transistors to be embedded into silicon. As mentioned above, one major company in this field now supports correct-by-construction techniques in their software. This could be used to produce a processor that is provably correct, and along with a software compiler which uses these techniques, surely a design could then be manufactured that is provably correct and hold? There is scepticism about this, as for the design to be provably correct, it must be correctly specified in the formal specification. Unfortunately, this is not an easy task, an example of an existing protocol

which suffers from this is the Internetworking Protocol, IP. If a packet arrives at a router with the 'don't fragment' (DF) flag enabled then ….

As mentioned earlier, finite state machines require an in-depth formal specification but it is easy to check for completeness, this is not the case for designs that cannot be specified as an FSM. With this problem, many engineers feel that a provably correct design may never be exactly 100%, but is it not good to get better designs?

CONCLUSION

Existing protocol designs have been found to contain many problems which cause connections or, more seriously, computers to crash. This is a problem in the application of safety-critical systems where a protocol is required that will not crash, even under extreme conditions. It would also be desirable to many engineers to have a protocol free from errors.

Correct-by-construction is a method of design based upon formal methods of verification and validation of a formal specification document, constructed thoroughly before design commences. This method is thought to provide a design which is provably correct and would be ideal to apply this technique in the design of protocols.

Drawbacks in this process include the amount of time and therefore cost during the validation stages which are carried out at regular intervals during the design. A degree of automation can be used in the form of software systems but these may contain bugs unless similar techniques are used in their production. This gives a trade-off which can be chosen appropriately to the design in question. Further disadvantages include a design which is too robust and as a result does not have good utilisation for a given communications media.

For internet applications, it is desirable to have a protocol that is freely available and uses the media optimally. In this application, it would not be desirable to use correct-by-construction techniques. In safety-critical applications however, the media does not have to huge utilisation. This is due to the low cost of high throughput media and very low cost of memory and processor power available today. In this case, correct-by-construction techniques would be much more useful to reduce risk.

The specification of a design is an area which provides difficulty. Finite state machines are easy to specify although lengthy, and as such are easily validated. Protocols can be designed as large, complex finite state machines, therefore using correct-by-construction may be a valid design method.

REFERENCES

[1] Godred Fairhurst, http://www.erg.abdn.ac.uk/users/gorry/eg3561/arq-pages/reliability.html, Lecture Notes, Retrieved Nov 2001

[2] "Parallel Cyclic Redundancy Check (CRC) For Hotlink™", Cypress Semiconductors Corporation, March 1999

[3] Anand R, Ramchandran K, Kozintsev IV (Sep 2001), "Continuous error detection (CED) for reliable communication", *IEEE Transactions on Communications*, **49** (9): 1540-1549

[4] Vinod Dham, "CAD Tools: Wish List of An IC Designer", VP Broadcom Corp.

[5] S. Bradner (Oct 1996), "The Internet Standards Process -- Revision 3", *RFC 2026*

[6] R. Braden (Oct 1989), "Requirements for Internet Hosts -- Communication Layers", *RFC 1122*

[7] Godred Fairhurst, http://www.erg.abdn.ac.uk/users/gorry/eg3561/intro-pages/osi.html, Lecture Notes, Retrieved Dec 2001

[8] W. R. Stevens, "TCP/IP Illustrated, Volume 1 – The Protocols", Addison Wesley, Oct 2000

[9] Allan Bruce (Aug 2001), "Notes on Setup of Hosts and Dummynet Machine for Simulation of Satellite Transfers and World-Wide-Web Experiments", Electronics Research Group – University of Aberdeen

[10] M. Rey (Sep 1981), "Transmission Control Protocol, DARPA Internet Program, Protocol Specification", *RFC 793*

[11] M. Player (Feb 2000), EG3563 Lecture Notes, University of Aberdeen

[12] Cadence Design Systems, http://www.cadence.com/datasheets/baseline.html, Retrieved Dec 2001

[13] D. W. Loveland (Nov 2000), "Automated deduction: achievements and future directions", *Communications of the ACM*, **43** (11es): 257-263

[14] L. Lamport (Nov 2001), "Specifying Systems – Preliminary Draft", Copyright Leslie Lamport

[15] G. J. Holzmann, "Design and Validation of Computer Protocols", Prentice-Hall, 1991

[16]