Eng. & Tech . Journal ,Vol.29, No.11, 2011

Development of an Adaptive File Transfer Protocol (AFTP)

Taif Sami Hassan *, Alyaa Hussain Ali **
& Alaa Noori Mazher***
Received on: 4 /5/ 2011
Accepted on: 20/6/ 2011

Abstract

File is one of the main storing units of information. There is a great need to transfer file through computer network especially the internet. The model that is used in Internet is the TCP-IP. The TCP-IP model consists of four layers. The lowest layer is the host network layer, the second layer is the internet layer (IP handling); while the third layer is transport layer which contains TCP and UDP. The last layer is the application layer. This paper is concerned on the User Datagram Protocol (UDP). The UDP is mainly used to transfer messages. It is asynchronous, connectionless and unreliable protocol. The UDP is not suited for transferring files because of the error or loose that could prevent us from reconstruct the original file. In order to overcome this problem new field and configuration is applied in the application layer for guarantee file sending and receiving. This field is the Cyclic Redundant Check (CRC) which is computed for each block of data and all these checks were sent over another port. At the end of receiving data the receiver checks the check matrix in order to determine the error and requesting that block from the sender. The proposed protocol is implemented using the Winsock and Visual Basic programming language.

Keywords: AFTP, TCP, CRC, UDP

تطوير أتفاقية وتكيفة لنقل الملفات

الخلاصة

الملف هو أحد وحدات الخزن الرئيسيه للمعلومات.وهنالك حاجه ماسه لنقل الملفات خلال شبكات الحواسيب وخاصة الانترنت. ان النموذج الخاص المستخدم في الانترنت هو TCP-IP والذي يضم اربع طبقات. الطبقه الاولى الاوطأ هي طبقة الحاسبة الى الانترنيت اما الطبقة الثانية هي طبقة الانترنت وهي مسوؤلة عن العنونة. بينما الطبقة الثالثة هي طبقة النقل وتحتوي على اتفاقتين (اتفاقية TCP واتفاقية UDP). اما الطبقة الرابعة والاخيرة فهي طبقة التطبيقات.

يركز هذا البحث على أتفاقية UDP. ان اتفاقية UDP مسوؤلة عن نقل الرسائل. وهي اتفاقية غير متزامنة بدون ربط غير موثوقة . ان اتفاقيةUDP غير مناسبة لنقل الملفات بسبب الخطا او فقدان البيانات والذي يمنعنا من اعادة تكوين الملفات الاصلية . لغرض تجاوز هذا المشكلة فقد تم اضاقة حقل واعدادات جديدة في طبقة التطبيقات ولضمان ارسال واستلام الملفات . هذا الحقل هو حقل فحص الفائض الدوار والذي يتم حسابه لكل مجموعة من مجاميع بيانات الملف المراد ارساله وهذه المعلومات ترسل على منفذ اخر غير منفذ البيانات. في الجهة المستلمه يتم تدقيق فحص الفائض الدوار لكل مجموعة لغرض معرفة حصول خطا ام لا وعند حصول الخطا في ارسال طلب للمرسل لغرض اعادة ارسال المجموعة الخاطئه.لقد تمت برمجة الاتفاقية المقترحة بواسطة الاداة Winsock و باستخدام لغة Visual Basic.

*** Compute Science Engineering Department, University of Al- Mammon /Baghdad**
**** College of Sciences for Women, University of Baghdad /Baghdad**
***** Computer Science Department, University of Technology/Baghdad**

https://doi.org/10.30684/etj.29.11.12
2412-0758/University of Technology-Iraq, Baghdad, Iraq

1- Introduction

The network play key role in varies application. There are many protocols that are used in varied type of processing such as Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Domain Name Server (DNS), and others. All these applications depend on the TCP-IP protocols. The FTP is used to transfer file. FTP has some drawbacks such as difficulties, slow, and low security. The FTP depends on TCP which is reliable, connection. In order to use UDP protocol which is fast and simple few facilities are introduced in the application layer to overcome the limitation that are exist in the UDP protocol. The main limitations are unreliability, and any error will destroy the whole file [1].

2-Transmission Control Protocol-Internet Protocol (TCP-IP)

The name TCP/IP is derived from the most widely used protocols in the suite: TCP is a byte stream protocol that provides reliable end-to-end communication between two processes running on the same or different host systems; IP is a simple best-effort packet switching protocol that allows many different interconnected networks to share the same virtual address space and form a single internet work. From the host's point of view, IP makes the underlying internet appear as a single virtual network [1].

The world's largest computer network, the Internet, uses TCP/IP as its dominant protocol suite. The Internet today is a worldwide internet. It connects over 1.7 million hosts and has a user community estimated at approximately 30 million people [1]. TCP/IP is also used on many corporate networks and is particularly popular with UNIX workstations. However, TCP/IP implementations can be found for almost any class of machine from PCs to supercomputers [1].

3- User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as data grams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP is sometimes called the Universal Datagram Protocol. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.[1]

UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and data grams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP)

or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games[2].

UDP applications use datagram sockets to establish host-to-host communications. Sockets bind the application to service ports that function as the endpoints of data transmission. A port is a software structure that is identified by the port number, a 16 bit integer value, allowing for port numbers between 0 and 65,535. Port 0 is reserved, but is a permissible source port value if the sending process does not expect messages in response [2].

4- Packet structure.

UDP is a minimal message-oriented Transport Layer protocol that is documented in IETF RFC 768.

UDP provides no guarantees to the upper layer protocol for message delivery and the UDP protocol layer retains no state of UDP messages once sent. For this reason, UDP is sometimes referred to as Unreliable Datagram Protocol [3].

UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload. If transmission reliability is desired, it must be implemented in the user's application.

The UDP header consists of 4 fields as shown in fig. (1). the use of two of those is optional in IPv4. In IPv6 only the source port is optional.

4-1 Source port

This field identifies the sending port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero [3].

4-2 Destination port

This field identifies the destination port and is required.

4-3 Length

A 16-bit field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that's the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65527 bytes of data) for a UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes.

4-4 Checksum

The 16-bit checksum field is used for error-checking of the header and data. The algorithm for computing the checksum is different for transport over IPv4 and IPv6. If the checksum is omitted in IPv4, the field uses the value all-zeros. This field is not optional for IPv6[4].

5- Cyclic Redundancy Check (CRC)

CRCs are based on the theory of cyclic error-correcting codes. The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks was first proposed by W. Wesley Peterson in 1961[4+]. Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors, contiguous sequences of erroneous data symbols in messages. This is important because

burst errors are common transmission errors in many communication channels, including magnetic and optical storage devices. Typically, an n-bit CRC, applied to a data block of arbitrary length, will detect any single error burst not longer than n bits, and will detect a fraction 1−2−n of all longer error bursts [5].

Specification of a CRC code requires definition of a so-called generator polynomial. This polynomial resembles the divisor in a polynomial long division, which takes the message as the dividend and in which the quotient is discarded and the remainder becomes the result, with the important distinction that the polynomial coefficients are calculated according to the carry-less arithmetic of a finite field. The length of the remainder is always less than the length of the generator polynomial, which therefore determines how long the result can be.

In practice, all commonly used CRCs employ the finite field GF (2). This is the field of two elements, usually called 0 and 1, comfortably matching computer architecture. The rest of this article will discuss only these binary CRCs, but the principles are more general.

The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC: it uses the generator polynomial x+1.

CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of messages delivered. However, they are not suitable for protecting against intentional alteration of data. Firstly, as there is no authentication, an attacker can edit a message and recalculate the CRC without the substitution being detected. This is even the case when the CRC is encrypted, leading to one of the design flaws of the WEP protocol [3]. Secondly, the linear properties of CRC codes even allow an attacker to modify a message in such a way as to leave the check value unchanged[4][5], and otherwise permit efficient recalculation of the CRC for compact changes. Nonetheless, it is still often falsely assumed that when a message and its correct check value are received from an open channel then the message cannot have been altered in transit.

Cryptographic hash functions, while still not providing security against intentional alteration, can provide stronger error checking in that they do not rely on specific error pattern assumptions. However, they are much slower than CRCs, and are therefore commonly used to protect off-line data, such as files on servers or databases [6].

6- Computation of CRC Code.

To compute an n-bit binary CRC, line the bits represent the input in a row, and position the (n+1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row.

Start with the message to be encoded:

11010011101100

This is first padded with zeroes corresponding to the bit length n of the CRC. Here is the first calculation for computing a 3-bit CRC:

11010011101100 000 <--- inputs left shifted by 3 bits

1011 <-- divisor (4 bits)

01100011101100 000 <--- results

If the input bit above the leftmost divisor bit is 0, do nothing and move the divisor to the right by one bit. If the input bit above the leftmost divisor bit is 1, the divisor is

XORed into the input (in other words, the input bit above each 1-bit in the divisor is toggled). The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

```
11010011101100 000 <--- inputs left shifted by 3 bits
       1011        <--- divisor
01100011101100 000 <--- results
 1011              <--- divisor...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000
     1011
00000000110100 000
      1011
00000000011000 000
      1011
00000000001110 000
       1011
00000000000101 000
       101 1
       -----------------
00000000000000  100  <---remainder (3 bits)
```

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These n bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some post processing) [7].

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- inputs with check value
1011       <--- divisor
01100011101100 100 <-- results
 1011             <--- divisor...
       00111011101100 100
       And so on until:
00000000001110 100
          1011
00000000000101 100
          101 1
       ------------------
             0
```

7-Application Layer

In the application layer there is a lot of work. At first file would be input and the file will be divided into block. The adopted block is 5000 Byte. The CRC code will be computed for each block. The block must be sending through the UDP internet protocol and using WinSock. After sending all the blocks into specific port. The CRC array also must be sent through another port. The receiver check the CRC code for each receiving block and the receiving CRC for the sender; if there is error with specific block the receiver request the sender in order to resent the error blocks as illustrated in figure.(2).

7-1 Socket

The Internet Transfer Control (ITC) is a handy control for Internet programming, but there is another control that is even more robust and helps programmers to create more flexible applications. The Winsock control comes with Visual Basic 6.0 (VB6) and is used to create applications that access the low-level functions of the Transmission Control Protocol/Internet Protocol (TCP/IP) [7].

TCP/IP is a specification that defines a series of protocols used to standardize how computers exchange information with each other. TCP/IP provides communication across interconnected networks that use diverse hardware architectures and

various operating systems. The protocols in TCP/IP are arranged in a series of layers known as a protocol stack. Each layer has its own functionality.

Winsock is a standard that is maintained by Microsoft. This standard is basically a set of routines that describe communications to and from the TCP/IP stack. These routines reside in a dynamic link library (DLL) that runs under Windows. The Winsock DLL is interfaced with TCP/IP and from there through the Internet.

This article will show how to use the Winsock in a client server environment. We will create two separate applications, one of which will be a server and the other will be a client. Both client and server will interact with each other to exchange data. Client will send a request to the server, and the server, which will be connected to a database, will retrieve the information requested by the client from the database and will return the requested information back to the client. You will find a database with this article. The database contains item numbers and their prices. In real-life situations, the database might be located on a machine different from the one that hosts the client application[7].

7-2Ports

let's talk about the ports before we proceed any further. A port is a special memory location that exists when two computers are in communication via TCP/IP. Applications use a port number as an identifier to other computers. Both the sending and receiving computers use this port to exchange data [7].

To make the job of communication easier, some port numbers have been standardized. These standard port numbers have no inherent value other than those users have agreed to use them with certain applications. The table below lists a number of popular and publicly accepted port numbers and their corresponding applications.

7-3 Using the Winsock Control

Winsock is above the TCP/IP protocol stack in the ISO/OSI model. TCP/IP is an industry standard communication protocol that defines methods for packaging data into packets for transmission between computing devices on a heterogeneous network. TCP/IP is the standard for data transmission over networks, including the Internet. TCP establishes a connection for data transmission and IP defines the method for sending data packets.

The Microsoft Winsock control makes using the TCP/IP a breeze. Microsoft has wrapped up the Winsock and NET API calls into a neat package that you can easily incorporate into your Visual Basic applications [7].

7-4 Winsock Operating Modes The Transport layer (also known as the Host-to-Host Transport layer) is responsible for providing the Application layer with session and datagram communication services. The core protocols of the Transport layer are TCP and User Datagram Protocol (UDP). The Winsock control supports the following two operating modes: [7].

sckTCPProtocol
sckUDPProtocol

8- The AFTP System

Our system was programmed using visual basic programming language. The system consists of two forms the first form is used to transfer data through the WinSock control, while the second form is used to receive data. Figures. (3 and 4) describe sending and receiving sides of AFTP.

Also in order to check the CRC for the file blocks before sending and after sending the CRC codes is written into TXT file as shown in fig (5).

9- Discussion and Conclusion

The objective of this paper is to design and build a simple and flexible protocol for sending and receiving big files. Our this follows a very well known fast and simple internet protocol User Datagram Protocol UDP and trying to get benefit from its facilities. The UDP is fast but it does not gaurntee to send the file with out error. There are a very well known error detection method Cyclic Redundancy Check CRC which is used with many application and protocol. This paper suggests dividing the file data into block in order to overcome the delay that could occur during error with specific data, the block will help us to resending only the error block not all the file. The CRC code will be compute for each block. The CRC code array must be send at last. The design give use great flexibility in sending all data and then if error occur only the error block must be resend, and the sending process will not get delay by waiting each block to be receive (Stop and Wait Protocol), and also the advance Sliding Window Protocol.

References

[1]Tanenbaum, A. S.; "Computer Network"; 3rd edit; Prentice Hall International Inc; 2010.

[2]Buchanan, W. J.; "Mastering Network"; 1st edit; MacMillan Press LTD; 1999.

[3]Hauden, M.; "Tech Yourself Networking in 24 Hour"; 1st edit; Sams Pub; 1998.

[4]Siebold, D.; "Visual Basic Developer's Guide to SQL Server"; SYBEX; 2000.

[5]Jerke, N.; "Visual Basic Developer's Guide to E-Commerce with ASP and SQL Server"; 1st edit; SYBEX; 2000.

[6]Williams, A.; Barber, K.; Newkirk, P.; "Active Server Pages Solutions"; 2000.

[7] Parab, H. A.; "A Case Study: Reverse Engineering a Multi-Tired Application with rational unified process, rational rose and UML"; M.Sc. thesis, 2001.

Table (1) public ports

Service	Port
HTTP	80
FTP	20, 21
Gopher	70
SMTP	25
POP3	110
Telnet	23
Finger	79
Local	loops/callbacks 0

bits	0 - 15	16 – 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Figure (1) UDP Header

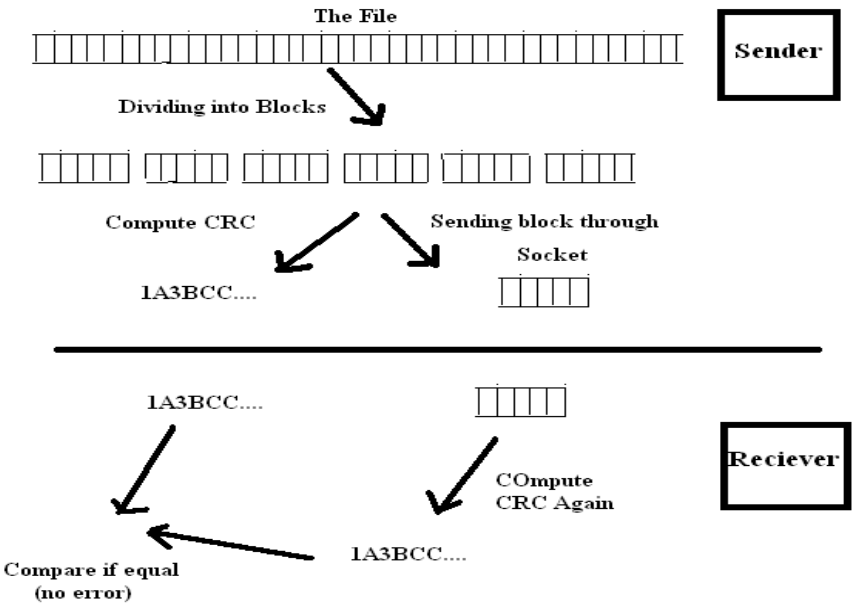

Figure (2): The AFTP System

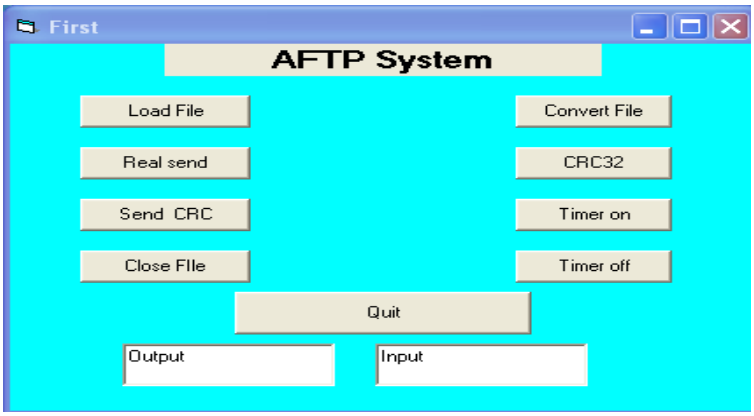

Figure (3): The Sending Side Program.

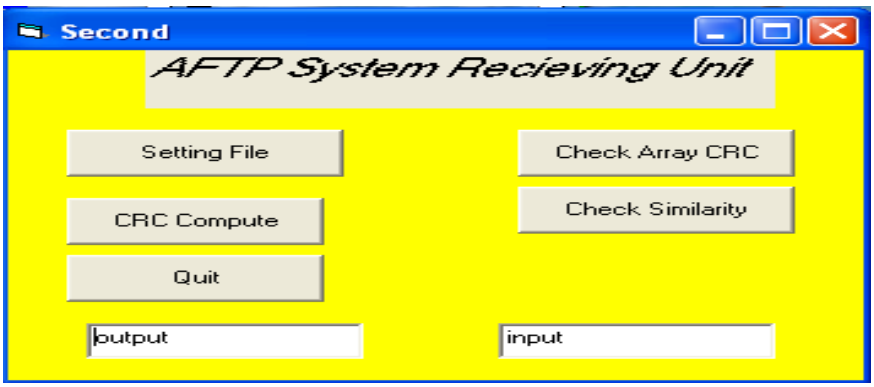

Figure (4): The Receiving Side Program

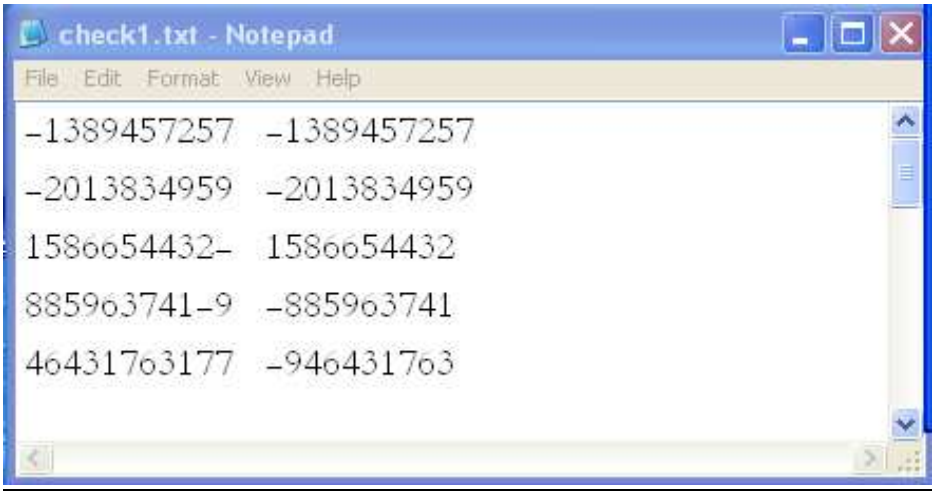

Figure (5): The Check CRC File