e-Learning for policing, intelligence and counter-terrorism: Performance constraints for confidentiality

Paul A. Watters
Postgraduate Professional Development Program, Division of Information and Communication Sciences,, Macquarie University NSW 2109, AUSTRALIA
<pwatters@ics.mq.edu.au>

Abstract

Online education for professionals in the policing, intelligence and counter-terrorism fields is problematic for several reasons: personnel are deployed in frequently changing locations; the confidentiality of the material being presented is critical; and the identity of and interactions between students may need to be restricted in various ways. In this paper, I examine whether ever-increasing wireless bandwidths (rather than CPU speed) can assist in the deployment of secure, mobile e-learning solutions for these students, by examining the effective bandwidth utilisation of a 1Gbps wireless network.

Using Java and .NET prototypes, I determined that the mean effective bandwidth utilisation for simulated streamed media was 3.10 Mbps for Java, some 330 times smaller than the physical bandwidth available. The mean effective bandwidth utilisation was 12.39 Mbps for Visual C++ .NET, some 82.64 times smaller than the physical bandwidth available. I conclude that network bandwidth is not a limiting factor in deploying these solutions – even existing wireless networks of lesser bandwidth could be used effectively.

1. Introduction

Distance learning has come a long way since bland yellow packages containing photocopied course notes and assignments were dispatched to students every semester. E-learning systems have facilitated student interaction through virtual classrooms, bulletin boards and wikis, while sophisticated multimedia solutions allow real-time streaming of audio, video and text content at high rates. These technologies are very effective when deployed in local areas, but questions remain about their effectiveness over wide areas, and mobile hosts.

Why do we need e-learning systems that are mobile? One student cohort that experiences a significant mobility requirement in their occupations are continuing education students from the areas of policing, intelligence and counter-terrorism. These students may be deployed at short notice to locations which are not known in advance, and where they may not be able to disclose public IP addresses for the end-to-end connectivity required by a virtual private network, for example. So, the requirement for these students is not just mobility, but a very high-level of security. Web-based e-learning systems may simply use Secure Socket Layer (SSL) technology here, or they may encrypt course material prior to transmission with a symmetric key distributed out of band.

Attempting to provide a secure, mobile e-learning solution is problematic, especially with respect to performance. Problems with the availability of network bandwidth are often commented on in e-learning circles, but what I hope to demonstrate in this paper is that the bandwidth of the wireless network is the least important factor in solving this problem – indeed, I am going to suggest that neither near-future 4G 1Gbps wireless or existing lower bandwidth technologies have the slightest impact on the deployment of secure, mobile e-learning systems. The major bottleneck, I suggest, is a CPU limitation introduced by the requirements of ciphers, particularly in the distribution of real-time media over a secure communications channel.

Technological advances in hardware bring about continuous improvements in speed and bandwidth. Several laws have been formulated to predict these changes, and these have generally shown to be true. For example, Moore's Law, formulated in the 1970's by Gordon Moore from Intel, predicts that CPU speeds will double every 18 months. In the networking world, Gilder's Law predicts that bandwidth triples every 12 months. The resulting trends indicate a rapid divergence of network bandwidth from CPU speed. While this difference may motivate CPU manufacturers to strive to outdo Moore's Law, in all likelihood, Moore's Law will hold. This leaves software developers will the task of developing applications that use relatively less CPU time and relatively more network bandwidth – otherwise, the network will lay idle for most of the time. For example, if a task requires 10 minutes to process 1Gb of data, that then takes only 1 second to transmit, then the network is being used for only one second out of six hundred. This represents only 0.16% of its total capacity. Idleness of 99.84% is not effective utilisation of a resource.

Technology is subject to the Law of Supply and Demand – if there is no demand for increased bandwidth, because applications cannot make use of it, then supply will have to be cut, and the implications for the network hardware industry could be quite serious. The key requirement for increasing demand for high bandwidth networking is to effectively close the gap between CPU speed and network bandwidth by developing applications that make better use of network bandwidth and hardware.

As I've indicated above, one area in which applications are increasingly placing demands on CPU speed is securing and encrypting data in secure, mobile e-learning environments. The development of Internet-based e-learning systems has accelerated the demand for secure exchange of data across wide area networks, especially for professionals in the policing, intelligence and counter-terrorism fields. For example, every time a student using a Tablet PC makes a connection to an e-learning website, all Hypertext Transfer Protocol (HTTP) requests and responses must be encrypted and decrypted using secure HTTP (HTTPS). When a client sends a request to the server, the request data is encrypted on the client side, and must then be decrypted on the server side. Conversely, when a server sends a response to a client, the response data is encrypted on the server side, and must then be decrypted on the client side. The security goal of HTTPS is to ensure that data exchanges are authentic and secure from interpretation, even if a packet sniffer intercepts them. The identification goal of HTTPS is to ensure that data being sent from a client to a server, or from a server to a client, is from a correctly identified source, by using public key cryptography.

When HTTPS was first introduced, key sizes of 40 bits were commonplace. Ciphers using small key sizes are now known to be easily breakable [1]. For example, the RSA Corporation recently challenged an invitation to crack a string encrypted using the Data Encryption Standard (DES) [2]. In less than 24 hours, the source string "See you in Rome (second AES Conference, March 22-23, 1999)" had been decrypted [3].

As cryptographic attacks on intercepted data transmitted through the Internet experience greater success, the key size for ciphers must continually increase. Many key sizes are now 128 or 168 bits in size, placing an increased load on client and server CPUs. As more and more Internet applications require data security provided through encryption, the situation will only grow worse, in terms of the effective bandwidth utilisation of individual applications.

Applications that require all data to be encrypted before network transmissions, such as e-learning systems, have not been able to effectively utilise greater bandwidths. This is because a bottleneck occurs at the source machine for data encryption, and at the target machine for decryption, that is significantly greater than the time taken for transmission. For example, encrypting and decrypting 1 Gb of data, that only takes 1s to transmit across the network, is likely to reduce the effective bandwidth utilisation of the applications significantly.

In some e-learning applications, the encryption and decryption overhead could be quite significant. For example, some architectures are designed to synchronously distribute multimedia to a large number of hosts effectively creating a virtual classroom. Ideally these systems have been implemented on private networks, so that data shared between the nodes would not be encrypted. However, given that policing, intelligence and counter-terrorism can be deployed anywhere in the field, there is a strong push towards using the public Internet for distributing e-learning material, requiring this data to be encrypted before transmission and decrypted after transmission. If the students share a group key (i.e., using a symmetric cipher), then this overhead is only incurred once on the server, and then once on every student's system. However, if every student has their own key, then the overhead would be incurred once on the server for every student, as well as once on every student's system.

Anecdotes concerning the CPU-intensive nature of encryption operations are easily found. However, there is little solid evidence to confirm the impact that data security operations have on e-learning applications that use a mix of text and media. Some papers [4] only evaluate the performance of specific CPU types, without relating them to the underlying bandwidth now available to tablet PCs or other mobile devices. The aim of this paper is to quantify the effective bandwidth utilisation of a simple application that requires encryption on the client side, and decryption on the server side, using a symmetric cipher. The goal is to determine, on a simulated mobile network, how effectively bandwidth is being used. Two platforms are used for comparison: a Visual C++ .NET and Java. These two languages are commonly used to build distributed e-learning applications such as those that operate over the Internet. By examining more than platform, the validity of the results of this study can be enhanced.

2. Methods

Two applications were written in Visual C++ .NET and Java to encrypt a 1Gb array of integers using the Data Encryption Standard (DES), to simulate a 1Gb multimedia e-learning file. The source code is contained in Appendix A and Appendix B respectively. Both were executed on the same Tablet PC computer system running Windows XP, with 512M RAM and a Pentium III CPU. No other user applications were executed during the testing period. The Java program was executed first using the Forte 4 for Java Integrated Development Environment (IDE), using the Java Runtime Environment (JRE) version 1.3.1_01 (HotSpot). The Java application was executed ten times. The Visual C++ .NET program was executed first using the Visual Studio .NET IDE, using the "Release" application solution configuration. The Visual C++ .NET application was also executed ten times.

3. Results

The mean effective bandwidth utilisation for the Java applications was computed by taking the quotient of 1 Gb data, and the time required to encrypt that data, for all ten observations. The mean effective bandwidth utilisation was 3.10 Mbps for Java, some 330 times smaller than the physical bandwidth available. The results for the Java program are shown in Table 1, and the code is shown in Appendix A.

Table 1. Effective bandwidth utilisation for encrypted data using Java.

Obs	**Size**	**Task**	**Total**	**Bandwidth (Mbps)**
1	1Gb	DES (Encrypt)	0:01:06	1.89
2	1Gb	DES (Encrypt)	0:00:42	2.98
3	1Gb	DES (Encrypt)	0:00:38	3.29
4	1Gb	DES (Encrypt)	0:00:38	3.29
5	1Gb	DES (Encrypt)	0:00:38	3.29
6	1Gb	DES (Encrypt)	0:00:38	3.29
7	1Gb	DES (Encrypt)	0:00:39	3.21
8	1Gb	DES (Encrypt)	0:00:39	3.21
9	1Gb	DES (Encrypt)	0:00:38	3.29
10	1Gb	DES (Encrypt)	0:00:38	3.29
		Mean	0:00:41	3.10
		SD		0.44

The mean effective bandwidth utilisation for the Java applications was computed by taking the quotient of 1 Gb data, and the time required to encrypt that data, for all ten observations. The mean effective bandwidth utilisation was 12.39 Mbps for Visual C++ .NET, some 82.64 times smaller than the physical bandwidth available. The results for the Visual C++ .NET program are shown in Table 2, and the code is shown in Appendix B.

Table 2. Effective bandwidth utilisation for encrypted data using C++.

Obs	**Size**	**Task**	**Total**	**Bandwidth (Mbps)**
1	1Gb	DES (Encrypt)	0:00:10	12.50
2	1Gb	DES (Encrypt)	0:00:11	11.36
3	1Gb	DES (Encrypt)	0:00:10	12.50
4	1Gb	DES (Encrypt)	0:00:10	12.50
5	1Gb	DES (Encrypt)	0:00:10	12.50
6	1Gb	DES (Encrypt)	0:00:10	12.50
7	1Gb	DES (Encrypt)	0:00:10	12.50
8	1Gb	DES (Encrypt)	0:00:10	12.50
9	1Gb	DES (Encrypt)	0:00:10	12.50
10	1Gb	DES (Encrypt)	0:00:10	12.50
		Mean	0:00:10	12.39
		SD		0.36

4. Discussion

The results of this study demonstrate that, using a standard computer system with either of two leading platforms for developing distributed systems, the effective bandwidth utilisation on a simulated 1Gbps link would be between 3.10-12.39 Mbps, some 82.64-330 times smaller than the available bandwidth. These figures only account for encryption on the client or server side: decryption on the client or server side would effectively halve these performance figures, suggesting that the effective bandwidth utilisation for the Java application would be 1.55 Mbps, and that the effective bandwidth utilisation for the Visual C++ .NET application would be 6.195 Mbps. This implies that the bandwidth utilisation is between 165.28-660 times smaller than the available bandwidth. Although it may seem fanciful to compute performance for 1Gbps wireless links, these are likely to become available on 4G networks in the near future [5]. In any case, the scale of difference between existing and future bandwidth availability is dwarfed by the gap between available actual bandwidth utilisation.

This study used DES encryption, which was one of the earliest forms of encryption made available. Most commercial-grade applications requiring data security would no doubt use Triple-DES encryption which requires three iterations of a standard DES algorithm combined with some data management operations. By extrapolation, using Triple-DES would push the bandwidth utilisation of the applications to be between 495.84-1,980 times smaller than the available bandwidth.

These results must be interpreted in the context of more expensive, specialised computer systems that are typically available as servers. For example, UNIX systems available from Sun Microsystems can host up to 106 CPUs on board. A base system of this type with only 72 CPUs costs US$3,235,430 according to Sun Microsystems' catalogue. Running the Java application on this hardware platform should significantly improve performance. However, even this system would not be able to encrypt and decrypt data using DES to bring it on par with the network's bandwidth. This reinforces the point that computer systems, even at the top end, cannot match the performance of the network.

If the network is so good, and if CPU technology is bound by Moore's Law, then methods, models, architectures and protocols need to be developed to move ubiquitous CPU-intensive tasks, such as encryption and decryption, to the network level rather than the system level. It isn't immediately obvious how to achieve this goal. However, the development of streaming ciphers, such as RC4 [6], suggests that better use can be made of the network capabilities in the future.

One barrier to hardware implementation is that these ciphers are not invulnerable to attack [7]. In addition, HTTPS is a strictly point-to-point technology between clients and servers – how can peer-to-peer and systems that bind multiple service providers into a single "virtual enterprise" be secured against attack, and yet use network bandwidth effectively? One possibility is for clients to share an IPSec "tunnel", to provide an authenticity and confidentiality guarantees for packets. However, implementing encryption at the hardware level is still prohibitively expensive – for example, nCipher's nFast 800 PCI card costs US$3,495, while Rainbow Technologies' CryptoSwift 400 costs US$3,849. Both cards provide hardware acceleration of encryption at a cost level that can be justified for servers, but not for clients. However, they are not yet available for mobile devices such as Tablet PCs.

In summary, the different growth rates for CPU and network capacity require that software developers create applications that use relatively less CPU time and relatively more network bandwidth – otherwise, the network will lay idle for most of the time. Many Internet applications require encryption as part of their normal operation, which uses relatively more CPU time, and makes even less efficient use of the network – although there are some significant differences between latencies depending on the platform and language chosen for development. While moving some of these operations to the hardware level may improve efficiency, this strategy is prohibitively expensive for clients.

In terms of e-learning, more computationally ciphers, coupled with hardware implementations designed for Tablet PCs and other mobile systems, would significantly improve the availability of secure learning solutions for students from the policing, intelligence and counter-terrorism fields – but switching to a 4G network is likely to make little difference.

5. References

[1] Netcraft (1997). Secure Web Server Survey. Available at http://www.netcraft.com/surveys/analysis/https/1997/Mar/CMatch/strongsv.html

[2] RSA Security (1999). DES Challenge III Broken in Record 22 Hours. Press Release available at http://www.rsasecurity.com/news/pr/990119-1.html.

[3] Electronic Frontier Foundation (1999). RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF). Press Release available at http://www.eff.org//Privacy/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html.

[4] Bui, T. (2002). *Performance Characteristics of the Intel® Itanium™ Processor on Data Encryption Algorithms*. Intel White Paper. Available at cedar.intel.com/media/pdf/security/perfcharipfencrypalg_final.pdf.
[5] IDG News Service. (2005). NTT DoCoMo hits 1Gbps in 4G field trials. Article available at http://wireless.itworld.com/4268/0506244g/page_1.html.
[6] Rivest, R. (1992). The RC4 encryption algorithm. RSA Data Security.
[7] Golic, J. (1998). Linear statistical weakness of alleged RC4 keystream generator. Lecture Notes in Computer Science 1403, pp. 226-238. New York: Springer.

Appendix A: Java Implementation

```
import java.util.*;
import java.security.*;
import javax.crypto.*;
import java.io.*;
import sun.misc.*;
import javax.crypto.spec.IvParameterSpec;

public class Encrypt
{
  public Encrypt()
  {
  }

  public static void main (String args[])
  {
    String cipherType="DES/ECB/PKCS5Padding";
    String keyType="SHA-1";
    Date d=new Date();
    byte bin[]=new byte[131072000];
    for (int i=0; i<131072000; i++)
    {
      bin[i]=1;
    }
    Security.addProvider(new
com.sun.crypto.provider.SunJCE());
    Key key;
    try
    {
      ObjectInputStream          in          =
new           ObjectInputStream           (new
FileInputStream("C:/SecretKey.ser"));
      key=(Key)in.readObject();
      in.close();
      System.out.println("Start...");
      System.out.println(d.getHours()  +  ":"  +
      d.getMinutes() + ":" + d.getSeconds());
      Cipher              cipher               =
      Cipher.getInstance(cipherType);
      cipher.init(Cipher.ENCRYPT_MODE, key);
      d=new Date();
      byte[] raw=cipher.doFinal(bin);
      System.out.println("End crypto...");
      d=new Date();
      System.out.println(d.getHours()  +  ":"  +
      d.getMinutes() + ":" + d.getSeconds());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

Appendix B: C++ Implementation

```
#include "stdafx.h"
#include <windows.h>
#include <time.h>
#include <Wincrypt.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
  printf("Starting...\n");
  char tmpbuf[128];
  _strtime( tmpbuf );
  printf( "Start time:\t\t%s\n", tmpbuf );
  HCRYPTPROV hProv   = 0;
  if(!CryptAcquireContext(&hProv,        NULL,
    MS_DEF_PROV, PROV_RSA_FULL, 0))
  {
    printf("Error          %x          during
    CryptAcquireContext!\n", GetLastError());
    return -1;
  }
  HCRYPTKEY hKey = 0;
  if(!CryptGenKey(hProv,             CALG_DES,
    CRYPT_NO_SALT, &hKey))
  {
    printf("Error  %x  during  CryptGenKey!\n",
GetLastError());
    return -2;
  }
  int srclen = 131072000;
  int bufferlen = srclen + 8;
  unsigned  char  *bin  =  (unsigned  char  *)new
char[bufferlen];
  for (int i=0; i<srclen; i++)
  {
    bin[i]=1;
  }
  DWORD destlen = srclen;
  BOOL ret = CryptEncrypt( hKey, 0, TRUE , 0,
bin, & destlen, bufferlen);
  _strtime( tmpbuf );
  printf( "End time:\t\t%s\n", tmpbuf );
  delete [] bin;
  return 0;
}
```