ELECTRONIC COMMERCE PERFORMANCE STUDY

Bob Kinicki, David Finkel, Mikhail Mikhailov, Joel Sommers
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609 USA

Sharon Cunningham, Yuriy Elkin, Ranga Gopalan, Mark Quinlivan
Stratus Computer, Inc.
55 Fairbanks Boulevard
Marlboro, MA 01752 USA

KEYWORDS

Electronic Commerce, Performance, Security

ABSTRACT

This paper presents the results of a project designed to explore and evaluate the process of conducting electronic commerce between Web browser clients on the Internet and business servers located on a corporate intranet. The project discusses the development of an electronic commerce system which runs a financial investment transaction application. A key consideration in the development of the system is the security measures needed to assure the success of the electronic commerce.

The report presents and analyzes the results from a series of performance tests designed to understand the tradeoffs between security and performance in an electronic commerce application. The results show that the SSL handshake is expensive and that the effect of other security measures are dependent on the specific architecture of the business application.

INTRODUCTION

The emergence of the World Wide Web as a significant medium for the exchange of information in the business community has occurred very rapidly. The next phase of this Internet revolution currently moving forward is the use of Web technology to conduct electronic commerce (EC). However, complete acceptance and adoption of business transactions over the Internet has been slowed by concerns in the private sector over issues of security, availability and performance of electronic transactions traveling over the Internet and into enterprise networks.

The speed at which the WWW has become part of the commercial world is actually one of the problems in studying whether corporations and individuals can utilize the tools available on the Internet to effectively and efficiently conduct business. Many of the tools and toolkits available for building and maintaining corporate Web pages have been developed at breakneck speed, and these products tend to be fragile because they were not specifically designed to interoperate with a variety of Web applications. Moreover, many of these tools do not address the security and scalability issues which have emerged as significant barriers for Internet electronic commerce.

Another factor in evaluating the viability of electronic commerce is the breadth of expertise needed to interconnect the new software, computer hardware and networking components which have become standard pieces of the Internet interface. In the fall of 1996, Stratus Computer, Inc. joined with researchers from the WPI Computer Science Department to form an Electronic Commerce Research Consortium to conduct research on electronic commerce. The goal of the EC project was to design and build a prototype electronic commerce system to serve as a testbed to study the issues of security, performance, functionality and availability. The objective was to select and develop a small electronic commerce application which included clients accessing a Web server from a remote Web browser to perform a business transaction which accessed both an application database held inside a corporate network and conducted a funds transfer via a credit card company.

This paper discusses the design and implementation of a prototype financial investment transactions application running over an electronic commerce system. A generic model of electronic commerce is introduced in the next section. The following section describes the three configurations of the EC research system with varying levels of security developed at Stratus Computer. The remaining sections discuss testing methodology, the results of the performance tests, and conclusions and future research.

GENERIC ELECTRONIC COMMERCE SYSTEM

System Architecture

When the Electronic Commerce Project began, part of the objective was to explore emerging Internet technologies. The plan was to choose a specific EC application which required the research group to select and evaluate currently available software tools for building Web pages. The EC system envisioned was divided into a front-end and a back-end. The front-end consisted of a Web server accessed by Internet clients through a Web browser. The back-end included an application server receiving transaction requests generated at the Web server and communicating with the appropriate corporate databases to carry out a transaction. The application server issues requests to a database server for specific database information and subsequently interacts with an electronic credit card processing system to handle the funds transfer component of the transaction.

After wrestling with several implementation issues, the generic EC model was modified to the simpler configuration shown in Figure 1. The electronic commerce activity is now divided into those activities conducted on the Internet and those communications carried out on a corporate intranet. The Web server becomes the interface between the Internet and the intranet by sending transaction requests to an application server on the enterprise network. The funds transfer interaction with a credit card checking system was dropped from the project because implementing this piece would require a tedious and time-intensive conversion from one network protocol to another network protocol.

Initially the Internet/intranet interface included a firewall mechanism. Due to time constraints, the firewall implementation was pushed back to the next version of the EC system. The baseline version of the EC system includes no security mechanisms. All messages between the Web browser and the Web server and on the corporate intranet are passed over the networks in the clear. HTTP is the protocol between the browser and the server, and TCP/IP is the protocol running over the corporate intranet. Security features were incrementally added to two other versions of the EC system which are discussed in the next section.

Selection of a Business Application

The next major decision was selecting whether to develop an application which included business-to-business transactions or rather to focus on business-to-consumer transactions. Once it was concluded that a business-to-consumer model was better suited to our investigative goals, a financial investment service was selected as the application to build on the EC system because it provided a reasonable variety of transaction types and because a prototype Web-based service could be built quickly if the investment functions were supported only for a restricted set of markets. Users gain access to their portfolios by typing a user name and password at the initial Web page, and once they are authenticated an additional Web page presents choices for customer actions.

IMPLEMENTATION OF AN ELECTRONIC COMMERCE SYSTEM

Starting with the EC research model given in Figure 1, the Electronic Commerce Research Consortium began the development an EC test system with Stratus Computer components within the Stratus corporate network. The initial development stage included a deliberate attempt to utilize and compare "off-the-shelf" Web page development products to access an electronic commerce database. The lack of flexibility in these products forced the research team to resort to using C and Perl programming for a large part of the development.

Basic Configuration

Instead of having multiple clients initiating Web server requests from separate PCs, the test system included a client driver process running on a single Pentium PC running Windows NT. The driver process executes a specified number of concurrent threads sending HTTP requests to the Web server. Each thread represents a browser client and adds one unit of offered load to an EC system experiment. The driver includes no display mechanism. Data responses received by the driver from the Web server are counted but not displayed before being discarded. After a brief comparison between a Netscape and a Microsoft Web server showed little performance difference, the Microsoft Internet Information Server (IIS) version 1.0 was selected as the Web server because it was compatible with currently available versions of Stratus system software.

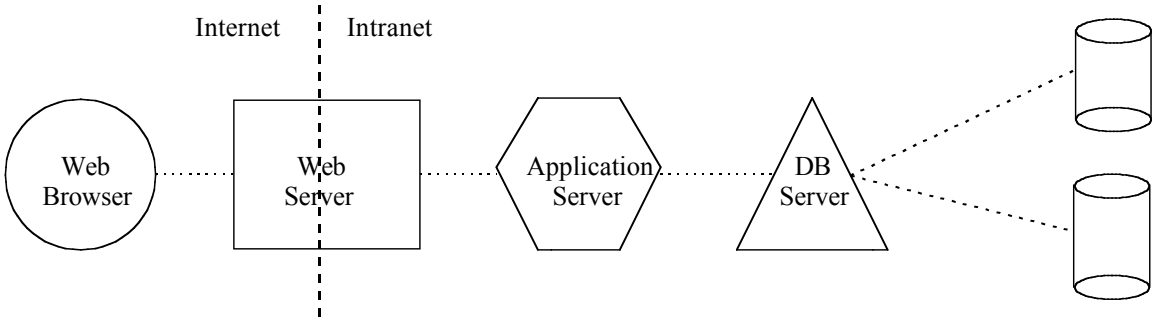


Figure 1: Generic Electronic Commerce System

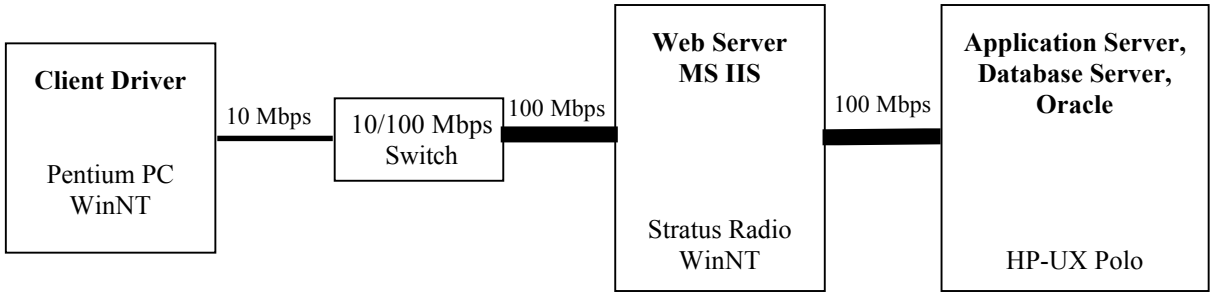


Figure 2: Stratus Electronic Commerce Research System

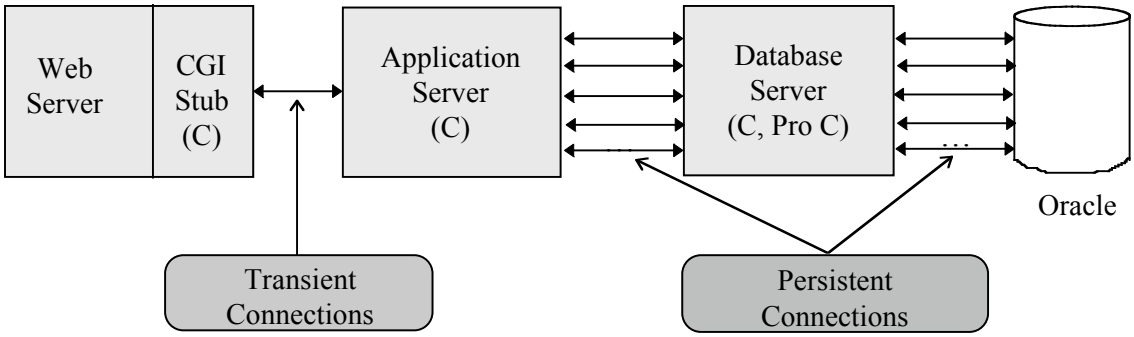


Figure 3: Server Connections for EC System

The PC hosting the client driver was connected to an Ethernet LAN within Stratus and the path between the driver and server included a 10/100 Mbps Ethernet switch. Figure 2 presents the details of the experimental electronic commerce system built. The Web server runs on a Stratus Radio computer system running Windows NT. The Web server uses CGI stubs written in C (see Figure 3) to send transaction requests over the Stratus network to an application server running HPUX on a Stratus Polo computer system. The application server interacts with a database server residing on the same Polo machine. The database server converts the application requests into specific requests to an Oracle database on the Polo machine.

Figure 3 gives the connection details between the three servers. The CGI Stub in the Web server communicates with the application server via transient connections which are disconnected when the application server responds to the Web server request. The connections between the application server and the database server and the

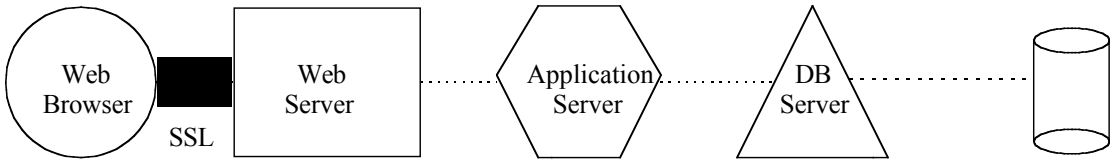


Figure 4: EC System with SSL

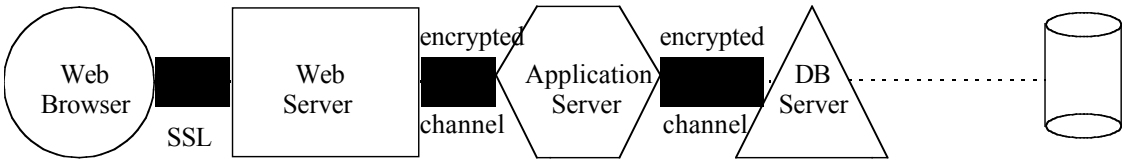


Figure 5: EC System with SSL and Back End Encryption

connections between the database server and the Oracle database were implemented as persistent connections to avoid significant performance degradation when accessing the database.

Configuration with Security between the Browser and the Web Server

The basic configuration provides no security protection for clients conducting electronic commerce via an Internet browser. Figure 4 shows the Secure Socket Layer (SSL) protocol [1] added to the base configuration to provide encrypted communications between the browser and the server. Since SSL negotiates the use of different ciphers when establishing a connection, the RC4 cipher in exportable (40 bit) mode was used for all SSL connections.

The SSLEay [2] implementation of SSL and the RSA BSAFE [3] encryption library for the RC4 cipher were used to build this configuration. To establish a secure channel, a TCP connection is opened to port 443 (a well-known port) on the Web server and then the SSL handshake protocol is executed to set the session parameters which include the cipher algorithm and the secret key used for the session. No client authentication process is implemented because the user is authenticated by the login/password sequence upon initial access to the financial services system.

The SSL protocol has a standard mechanism for resuming sessions with an SSL server which is designed to simplify the handshake process. However, Stratus computer systems feature failover mechanisms for improved availability. Thus the Stratus Electronic Commerce Research System includes an SSL handshake for each HTTP request and SSL version 2.0 was used in implementing the EC system.

System with SSL and Back End Encryption

The SSL configuration includes mechanisms to deal with security attacks carried out on the open Internet, but security techniques are also needed at the Internet/intranet interface and within the intranet itself. Firewalls are the currently accepted mechanism for dealing with security protection at the Internet/intranet interface. As previously mentioned, the firewall implementation has been designated for the next phase of this research study. This section discusses the security measures implemented to protect the electronic commerce data traveling over the corporate intranet.

The SSL and Back End Encryption configuration shown in Figure 5 includes encryption of all data transferred between the CGI program on the Web server and the application server and between the application server and the database server using the IDEA cipher [4]. IDEA is a symmetric cipher where a private key used to encrypt is also used to decrypt messages. This implies the private key must be securely stored at both servers.

The IDEA cipher is used instead of SSL between the CGI program and the application server because the connections between these two processes are transient. If SSL were used, each client transaction would require two SSL handshakes.

Table 1

Transaction Types

Identifier	Transaction	Description
1	Web server only	Simple HTTP GET request to gauge performance of non-CGI operations.
2	Customer validation	If user name and password match in database, return main option page (look at account information, display open orders, display transaction history, etc.)
3	Open orders display	Display a table of the customer's open orders
4	Transaction history display	Display a table of the customer's completed orders.
5	Load order entry page	Get HTML form for customer to create an order. Loading this page involves getting the full list of securities from the database (250 symbols and names). This transaction returns the most data, so it is the most network and encryption intense.
6	Create order	Use information entered in the form generated from transaction 5 and create the customer order.
7	Transaction mix	Mix consists of 8 atomic transactions: customer validation, load order entry page, create order, display transaction history, customer validation, load order entry page, create order, display open orders.

PERFORMANCE TEST METHODOLOGY

This section discusses the details of the performance tests and includes the specific set of transaction types implemented for testing.

Test Configuration Issues

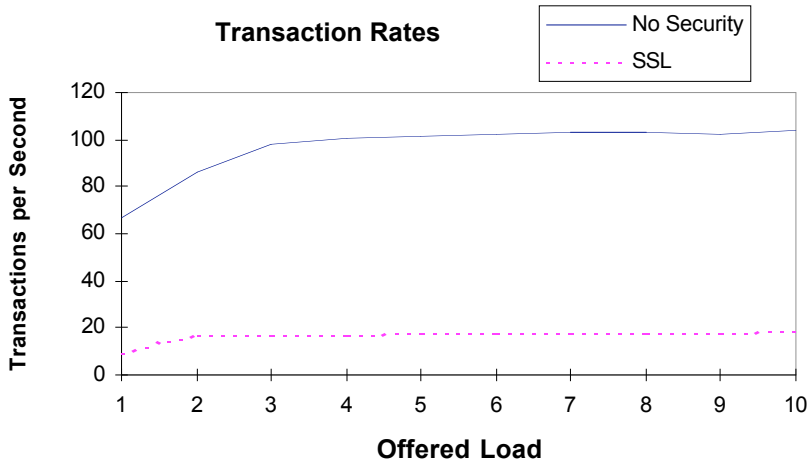
The Stratus Electronic Commerce Research Consortium conducted a series of experiments on the three configurations described above. The tests are controlled through the client driver program running on the PC. Throughput and round-trip response times are measured by time-stamping activities at the PC. The response time includes the time to open a TCP socket to the Web server, make the HTTP request, receive the HTTP reply, and close the socket. Reported response times are averaged over all concurrent threads.

The driver program has a command line interface for specifying the number of concurrent client threads to spawn for a given test. At driver startup, each spawned thread loads a text file for a specified transaction type. The text files contain the sequence of HTTP requests to make to the server for a given transaction type. If HTTP POST queries are requested an additional file containing POST form data is loaded. Each test run consists of each of the clients looping through the set of HTTP requests for a 30 second warm-up period (no statistics collected) followed by a 1 minute test with performance monitoring.

In the driver implementation the standard TCP *close* sequence caused severe degradation to the transaction execution rate. This problem was bypassed by having the driver use the *shutdown* system call instead of the *close* system call. Another problem encountered when running experiments was the behavior of the Microsoft IIS 1.0 server when it was forced to handle a high number of CGI requests. After a sustained period with many CGI connections, the behavior of the server system became erratic. Our test times were restricted because of this problem, but rebooting the system between runs helped to avoid this problem.

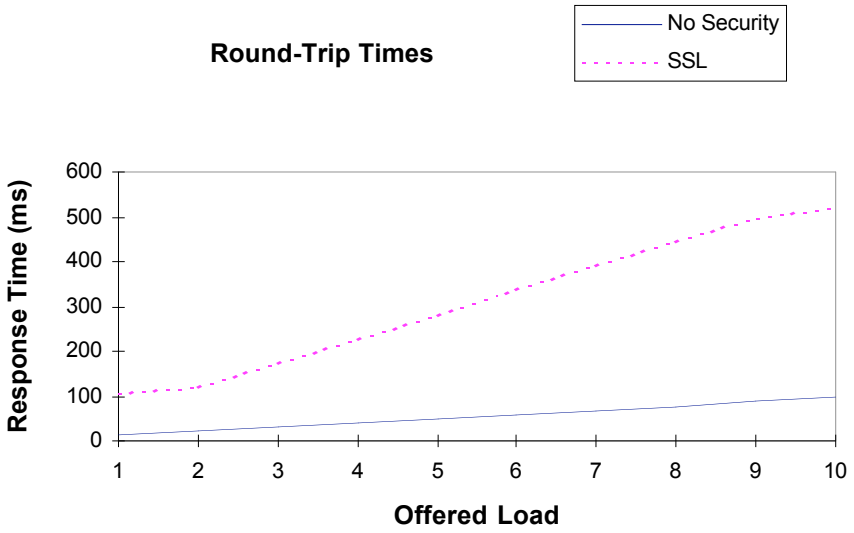
Financial Investment Service Transactions

After reviewing expected customer actions within the framework of a financial services application, the research group created seven transaction mixes to characterize the behavior of electronic commerce clients (see Table 1). The types were chosen to isolate some of the expected performance effects due to specific transactions. For example, a type 1 transaction does not access the Oracle database and consequently has a shorter response time than the other transaction types.



Transaction Rates for Transaction 1

Figure 6a



Response Times for Transaction 1

Figure 6b

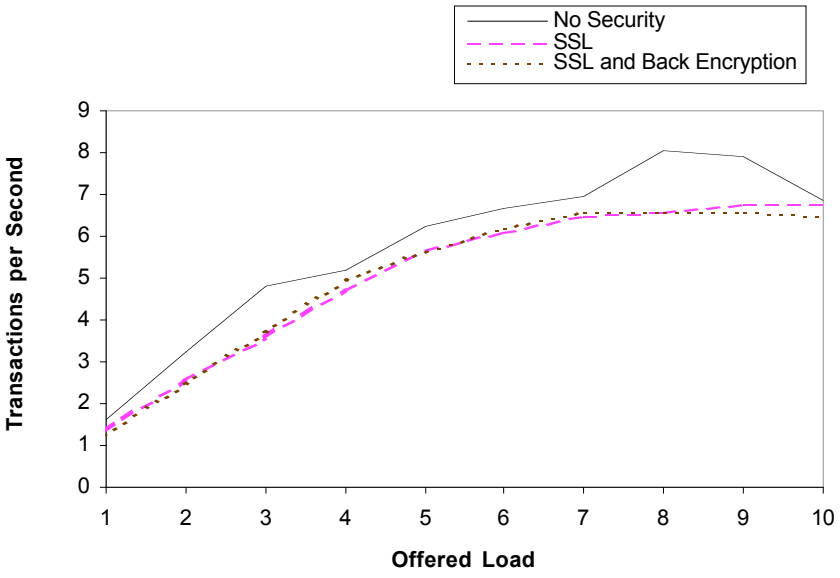
PERFORMANCE RESULTS

This section presents the results from a series of tests run on the three configurations discussed above. The tests were devised to examine the performance of the electronic commerce system by increasingly incorporating security components into the system. By varying the transaction mixes, the experiments provide insight into the relative cost of security mechanisms for a variety of client activities.

Retrieving Static Web Pages

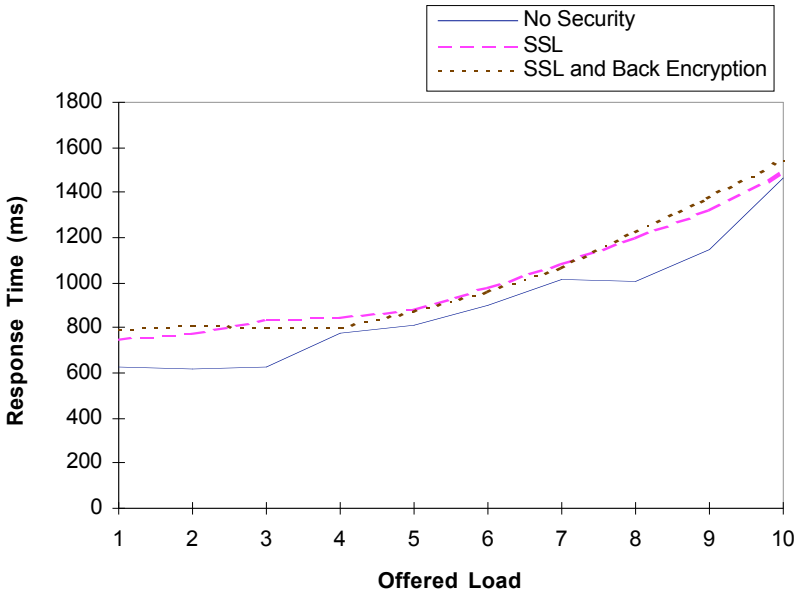
The first set of tests conducted isolate the effects of adding SSL between the Web browser and the Web server. This is done by running a test where the clients repeatedly issue transaction 1 requests (see Table 1). These simple HTTP GET requests require the server to access only static Web pages. The server issues no CGI operations and the back-end of the system is not involved in the transaction.

Figure 6 shows the results from two series of tests run using only transaction 1 requests where the number of concurrent clients (offered load) at the



Transaction Rates for Transaction 2

Figure 7a

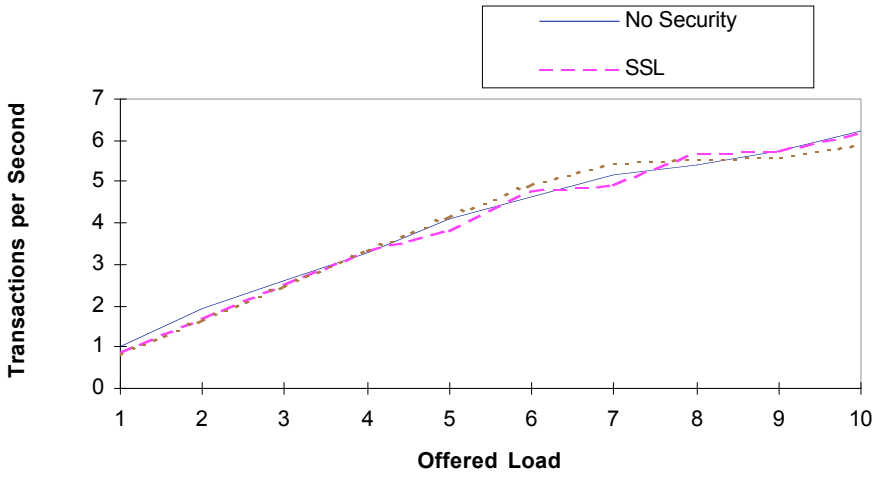


Response Times for Transaction 2
Figure 7b

driver is varied from 1 to 10. The specific transaction is a request for the Web server root document */index.html* .

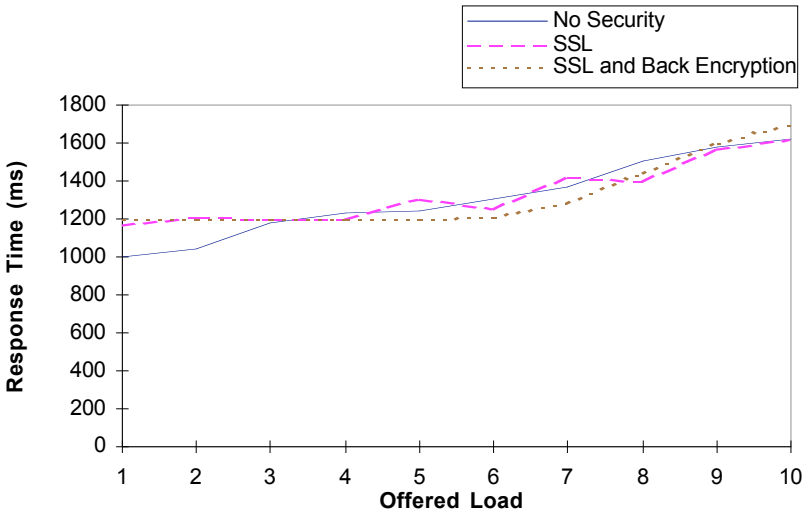
The **No Security** curve corresponds to tests run on the basic configuration while the **SSL** curve represents tests run on the SSL configuration represented in Figure 4.

Figure 6a shows dramatically the performance cost associated with adding SSL. With only one client the difference in transactions per second (tps) is 50. With no security measures, the tps rises linearly to 100 tps at 3 clients and stays at that level for loads of 100 clients (not shown in figure). With SSL the transaction rates quickly levels off at 17 tps.



Transaction Rates for Transaction 5

Figure 8a



Response Times for Transaction 5

Figure 8b

The response time results in Figure 6b are consistent. SSL yields slower response times even for a single customer, and at a load of 10 the SSL response time is 5 times slower than the system without SSL. A further refinement of the measurements show that the SSL handshake is responsible for about 75% of the total response time in Figure 6b.

Encrypted CGI Requests

The next set of tests study the effects of adding encryption to the messages between the three servers (Web, application and database). Note all transaction mixes except type 1 issue HTTP POST requests. This means that for transactions 2 through 7 the CGI program at the Web server is invoked and the application and database server are involved.

Figure 7 presents the results for all three configurations when the clients are issuing only type 2 transactions, customer validations. The increased security of encrypted messages has little effect on the transaction rates and response times shown in Figure 7. Comparing the insecure type 1 transaction (static Web page fetching) rates in Figure 6a with the insecure type 2 transaction (dynamically-generated Web pages) rates in Figure 7a, one sees a drop from 100 tps to 6tps at a load of 5 concurrent clients. These results imply that the cost associated with accessing the backend of the process (the application and database server) is significantly higher then the overhead attributed to secure communications.

The back-end performance cost reported for dynamically-generated Web pages can partially be attributed to using CGI instead of using native server API at the Web server. The overhead of using native API to generate Web pages and issue requests to the application server is much less than CGI overhead. However, a CGI implementation is portable not only across Web servers but also across operating systems and hardware platforms.

Figure 8 presents results from a set of tests where only transaction type 5 requests are issued. Type 5 transactions send much more data then type 2 transactions. Since the largest component of SSL is the handshake, Figure 8 shows that the relative impact of the SSL overhead is reduced when more data is sent per transaction.

CONCLUSIONS AND FUTURE RESEARCH

The Stratus Electronic Commerce Research Consortium implemented a prototype financial services application on an EC research system within the Stratus corporate network. Three distinct configurations (Figures 2, 4 and 5) were employed to study the impact of secure sockets and message encryption on end-to-end electronic transactions. The results from a series of experiments show that an SSL handshake severely reduces the transaction processing rate.

The cost of encrypting messages is small relative to the cost of using CGI stubs and the high overhead inherent in an Oracle database access. The impact of the SSL handshake is shown to be reduced when the volume of data transmitted per transaction is high.

The use of several different transaction types for the financial services system produce significant performance differences. This suggests that the design and performance of an electronic commerce system will depend on the nature of the business transactions.

The next phase of this research will include building a firewall between the client driver and the Web server. The Stratus EC Research Consortium hopes to expand the financial services applications to include live data and interface the system to a credit card processing system through an internal product which would allow the transactions to go over protocols other than TCP/IP. The EC research system was designed such that multiple servers could be implemented at the Web server, the application server or the database server. Multiple server tests are needed to bring the transaction rate for the secure configurations up to an acceptable level.

REFERENCES

1. http://home.netscape.com/eng/ssl3/index.html
2. http://www.psy.uq.edu.au:8080/~ftp/Crypto
3. http://ftp.rsa.com/rsaref/
4. Schneier, Bruce, *Applied Cryptography*, 2nd Ed., Wiley, New York, 1996.