Impact of Network Quality on the Performance of Distributed Applications based on CORBA

T. Andrew Au
Defence Science and Technology Organisation, Australia.
DSTO C3 Research Centre, Fernhill Park,
Department of Defence, Canberra,
ACT 2600, Australia.
Email: andrew.au@dsto.defence.gov.au

Abstract

CORBA is a middleware technology that simplifies or helps in the construction of distributed applications by providing standardised mechanisms that objects can use to communicate over a network. Applying CORBA to C3I problems can provide simple integration of disparate information systems in the military environment. CORBA has primarily been designed for the commercial environment where clients and servers are located in the same local area networks. With the growing number of CORBA-based applications, higher demands are placed on the performance of CORBA operating beyond local area networks. However, wide area networks tend to display a greater variation in operational behaviour that CORBA may not adequately accommodate. Further, a robust middleware infrastructure is a necessary underpinning for effective distributed collaboration across the range of stringent military networks. In order to exploit CORBA in the military environment, we need to understand its system characteristics so as to stretch the boundaries of CORBA-based solutions. This paper reports experimental investigations of the performance of CORBA under typical test environments. We attempt to identify problems that arise from different network conditions in the interaction between client and server. Our experimental results provide guidance in assessing the feasibility of using CORBA in various military computing environments.

1. Introduction

The Experimental C3I Technology Environment (EXC3ITE) is an Australian Defence Project, to develop and leverage the use of distributed object middleware in the Defence Information Environment. Distributed object middleware is an emerging technology leading from current Internet client-server based systems towards next generation Internet multiple component peer-to-peer networks. The benefits of such middleware architectures include evolvability, reuse, scalability, and reliability, thereby offering significant advantages in relation to our multi-billion dollar information infrastructure. Military communications and information systems are often characterised by the development of expensive, purpose-built and non-interoperable systems. The increasing frequency of joint military operations makes interoperability of these systems within and across Services essential. As investment focuses on interoperable information services and components, the legacy of costly,

stove-piped applications will diminish and the vision of scalable information management will be realised.

The systems architecture of EXC3ITE is founded on the concept of using components to encapsulate intellectual property, at a level of granularity that allows ready sharing and re-use, similar in some respects to the Defence Information Infrastructure Common Operating Environment (DII COE). With a suitable set of components, C3I applications can be constructed rapidly and, where appropriate, by end users. Information services (both remote and local) are presented as components, executing as location-independent entities within a distributed system. EXC3ITE currently supports the Common Object Request Broker Architecture (CORBA) [1] and is being extended to accommodate other middleware technologies such as the Distributed Component Object Model (DCOM) and Java™ 2 Enterprise Edition (J2EE).

CORBA is a software layer situated between the application and the operating system. It simplifies construction of distributed applications by providing standardised mechanisms for distributed components to communicate over a network. Typically, a server application stores data that can be accessed by many clients. The clients issue short request messages to the server, which after servicing, eventually sends a response. Applying CORBA technology to C3I problems in the military environment allows simple integration of disparate information systems. Nevertheless, CORBA has primarily been designed for the commercial environment where clients and servers are collocated on the same local area network (LAN), rather than the conditions one would expect in military environments. With the growing number of applications that use CORBA as their infrastructure, it is becoming common that users request distributed services across wide area networks. Higher demands are therefore placed on the performance of CORBA operating beyond LAN environments. Designers of client-server systems have long realised how important it is to performance to minimise the amount of data traffic between client and server. Although the performance achieved through CORBA may not be as good as when using low-level approaches [2], this framework should still need to provide acceptable performance.

The current EXC3ITE network backbone is a high-bandwidth Asynchronous Transfer Mode (ATM) network operating over commercial optical fibre and satellite links using military grade encryption, but EXC3ITE concepts are intended to apply more generally. The performance of distributed applications can become unacceptable because of network limitations such as longer propagation delay, lower bandwidth and poorer reliability of data transfer. In particular, satellite-connected nodes suffer appreciable (1-2 sec) round-trip latency in communicating with other nodes, due to the height of geostationary orbits. In addition, they will typically have less bandwidth available than those connected over terrestrial links. The question arises as to how increased latencies and reduced bandwidths affect the efficiency of CORBA. For example, if there is significant hand shaking between nodes this could have a considerable impact of the performance of time critical applications such as real-time interactive simulation.

From the users' perspective, distributed services must be reliable, flexible, reusable, and capable of providing scalable, quality service to user applications. Many of the system performance characteristics and criteria for assessing the feasibility of CORBA in distributed object computing remain to be established. In order for distributed computing to succeed at a global scale, we need to investigate and clarify various characteristics of this technology. Distributed computing can require

large amounts of data to be communicated over the network. Although available bandwidth is rapidly increasing, we need quantitative measures to judge its sufficiency. Also, since latencies will not improve substantially in the future, it is not clear to what extent high latency will compromise performance.

To investigate the feasibility of using CORBA in military computing environments, this paper reports experimental investigations of the performance of CORBA under expected network conditions. We characterise the performance of CORBA in a range of different network environments, in terms of response time and throughput at the application level. We attempt to identify problems that arise from these conditions in interactions between client and server. In so doing, we aim to gain insights into performance effects due to overheads in the processing hosts and transport over the underlying network. Section 2 describes characteristics of the global Internet and overheads incurred in distributed applications due to CORBA. Section 3 describes the test environment used in our experiments. Section 4 presents results and Section 5 draws conclusions.

2. Distributed Applications over the Internet

The Internet promises worldwide efficient and inexpensive communications. This global infrastructure embraces a mix of communications and computing technologies, as is typical for distributed applications in military computing environments. The Internet of the future will be characterised by additional demands due to mobility. Seamless access to information will need to be available anytime, anywhere. This infrastructure will increasingly be used for distributed computing where rapid and reliable communication is essential to satisfy user requirements. Distributed applications often come with a spectrum of distribution requirements. Users in a fixed location will continue to have almost unlimited access to information by means of high capacity communications, such as satellite, optical fibre and radio relay. Some communication technologies will, however, be denied to truly mobile users.

2.1. Characteristics of the Internet

The quality of the Internet is basically represented by two important network variables: packet loss and network delay. Packet loss measures the reliability of a connection. Measurements of the Internet [3] indicate that 0% to 8% of packets are being lost globally. Packet loss can be attributed to congestion, bit errors, and deliberate discard. Routers discard incoming packets that cannot be transmitted or stored in the buffer. Some bits can be corrupted as data packets are transmitted from one place to another, although error detection techniques are commonly used to check data integrity. Packet discard due to corrupted data is more likely on wireless and satellite links because of poorer link quality. Some networking technologies such as ATM guarantee resource reservation for traffic of higher priority. If there is less bandwidth to transmit all incoming packets immediately, some of the packets of lower priority are either stored in a buffer or simply discarded. Backbone routers may offer packet discard policies so that certain types of traffic are degraded as the router approaches congestion.

The round-trip delay on the Internet is how long it takes for a data packet to travel from one point to another and back. Delays on the global Internet range from 80 ms to over 600 ms. Due to the low

propagation delay in optical fibre, a typical round-trip delay is 80 ms when going between national backbones in North America. Higher latency connections are found on international and satellite communications. It typically takes 300 ms for data to reach its destination on the international Internet and for the response to be returned. Due to the propagation delay over satellite links, it takes 250 ms just for the signal to get from the earth to the satellite and back. A typical round-trip delay is 600 ms when a satellite is in the forward and return paths between client and server. As an indication, Table 1 reveals the round-trip delay of current Internet connections.

Table 1: Typical Internet round-trip delay

local networks	10 ms
national networks	80 ms
international networks	300 ms
satellite links	600 ms

As the Internet becomes the key to future distributed applications, the challenges to the designers are enormous. The network latency is substantial and the packet loss is high. These applications need to be robust and responsive even if the underlying network is slow and unreliable.

2.2. Overheads due to CORBA

CORBA provides an environment for developing and deploying object-based distributed applications. The overall architecture of CORBA is illustrated in Figure 1. At the uppermost layer, the exact connection mechanism between the client and the server is hidden from application programmers, as if the client and the server programs reside on the same machine. CORBA objects are accessed through public interfaces specified in an Interface Definition Language (IDL). An IDL compiler produces stub and skeleton classes, which are used respectively on the client and server hosts to provide presentation layer services for network transmission.

The middle layer provides the necessary infrastructure to hide the details of the underlying Object Request Broker (ORB) layer from applications, making remote object invocation look similar to local invocation. When the client invokes a remote operation it is effectively invoking the operation of that name provided by the client stub. This involves bundling and unbundling the call parameters, known as marshalling and unmarshalling, for transmission across different address spaces. The bottom layer of CORBA specifies the wire protocol for transmission between the client and the server running on different machines, with the ORB acting as an object bus. The ORB is a collection of libraries and

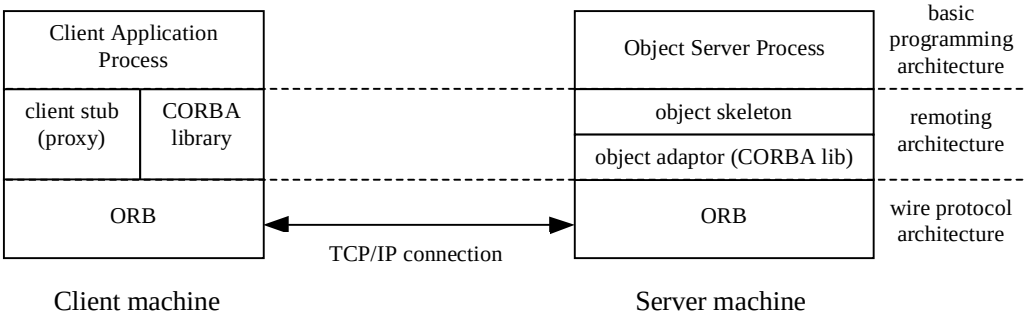

Figure 1: CORBA overall architecture.

network resources that is integrated with end-user applications. It is responsible for locating the remote object and preparing it to receive the request. The ORB then passes to the object the request in the form of a buffer so that the appropriate operation can be executed. The CORBA 2.0 specification defines the Internet Inter-ORB Protocol (IIOP) for interoperability among TCP/IP based ORBs, in which the Internet is effectively used as an ORB communication bus.

To invoke operations on an object across heterogeneous ORBs, a client must first obtain its interoperable object reference (IOR). Once an IOR is obtained, the client-side ORB connects to the address and port number specified within the IOR. When the TCP/IP connection is successfully established between the ORBs, the client-side ORB creates a proxy object and finally returns the IOR to the client program. The client is able to invoke methods on the proxy object, which in turn, interacts with the server object. In effect, the client's method invocations are translated into operation requests that are sent to the server object.

When the client invokes the remote operation on the proxy object, the stub obtains a buffer and marshals the operation name and necessary parameters into the buffer. This buffer is handed to the underlying ORB, together with the object reference. The client-side ORB then sends the buffer to the target server through the established connection. After receiving the buffer, the server-side ORB unmarshals the object reference to determine the requested object. The remainder of the buffer is then passed to the object skeleton. The skeleton unmarshals the operation name, and the parameters for the operation. It then invokes the operation of the object, passing it these parameters. Once the operation has been executed, the skeleton obtains a buffer and marshals any results that are returned from the operation into the buffer. The buffer is then passed to the ORB to be sent to the client. The proxy object then unmarshals the results, checks for exceptions, and returns them to the client program.

CORBA processing overheads can be attributed to many factors [4]: presentation and session layer conversions and data copying, server demultiplexing, and buffering for network reads and writes. In the process of object activation, a client always binds with an agent or a directory service. This ensures that the request is always routed to an active server, allowing object migration and load balancing. On the other hand, if a client is given a persistent IOR, it needs to contact the implementation repository to ensure that the first request for an object is passed to the correct server. Once the client knows how to contact the object, further requests can bypass the access to the repository. Nevertheless, the location-forward reply from the repository inevitably incurs additional overhead in the retransmission of requests, especially when requests carry large amounts of data.

Depending on the network delays between client, agent, implementation repository and server; significant delay is entailed in the process of object activation. The IIOP consists of simple request-reply interactions. In addition to the TCP and IP headers, IIOP messages typically carry IIOP headers to support ORB interoperability and CORBA transparencies such as presentation layer translation of data into an interoperable format. Our observation of data transfer using the Inprise VisiBroker ORBs indicated that the IIOP overhead is relatively constant at 60 bytes from client to server and 28 bytes return, regardless of the size of IIOP messages. This seemingly insignificant overhead can, however, become a real burden to short IIOP messages for interactive applications especially over high-latency low-bandwidth channels.

The mainstream of CORBA-based solutions is targeted at LAN-based applications and relies on fast, reliable connections. Mobile and satellite links tend to display more variation in operational behaviour that CORBA may not adequately accommodate. Mobile end-user systems often experience changing characteristics of the underlying communications infrastructure — station closures, variation in throughput, and inadequate coverage — leading to a significant number of lost packets. Often distributed applications are not written to deal with performance degradation of the underlying networks. In this environment, objects can fail unexpectedly, and the response time for a method invocation can be unacceptable.

3. The Experimental Environment

The principal aim of our work was to measure the performance of distributed applications over CORBA in a variety of network environments. The test programs were run under different network conditions, each designed to represent different levels of connection performance.

The client is a Pentium III 500 MHz machine running Microsoft Windows 2000 Professional. The server is a Pentium III 866 MHz machine running Microsoft Windows 2000 Server. All tests were performed on these two machines. Various network conditions were simulated using the NIST Net Emulation Package [5]. The client and the server were connected to the emulated WAN via T1 links of 1.544 Mbps, representing a typical business connection to the Internet. Inprise VisiBroker for C++ [6] was used to provide the communication mechanisms. TCP/IP for Windows 2000 has a number of features that optimise TCP performance in various network environments, including support for large TCP windows, selective acknowledgements, and better round-trip time estimation.

The performance measures of interest are the mean client response time and the throughput in terms of bits per second. A simple C++ test routine initiates remote invocation over CORBA so that data flows in both directions alternately in a ping-pong fashion. We measured the response time and throughput between the client and server for a range of message block sizes, under varying network conditions. The response time is the actual time that elapses between the initiation of a method invocation by the client until the result is returned to the client. This simple operation was repeated 1000 times to get a reliable average value and the available throughput as a function of the message size.

We used the same ping-pong test throughout the experiments. The request service time at a particular message size is basically the same for all the tests under different network conditions. By using different combinations of network latency, packet loss, and message size, it is possible to investigate the effects of these variables on the overall performance of the application.

Various network conditions were selected for the collection of experimental results. End-to-end network latencies of 5 ms, 40 ms, 150 ms and 300 ms were simulated to cover the range of round-trip delays expected in typical Internet connections. We simulated packet losses from 0% to 8%, which is the range expected in the Internet. The packet loss is the same in both directions the data blocks are being transferred. Performance results for different data block of size 1 to 65536 bytes were collected.

4. Results

Figure 2 - Figure 5 compare the response times between the client and server across the underlying emulated network for various message sizes. Each graph shows the response time as a function of message size for a particular value of network latency.

Performance degradation is noticed when the client and the server have to communicate under worsening network conditions. It is obvious that the response time increases steadily with the value of network latency (from 5 ms to 300 ms) as well as the message size (from 1 byte to 65536 bytes). However, the effect of network latency on the overall performance is not as detrimental as that of packet loss. A small level of packet loss can greatly increase the response time. Note that high packet loss is not uncommon in long-haul connections where wireless or satellite links are involved.

In an ideal distributed computing situation, the cost of sending a message between two programs located on different processors can be represented by two parameters: the message start-up time t_s, and the transfer time per data unit t_w [7]. The former is the time required to initiate the communication, whereas the latter is determined by the effective bandwidth of the communication channel linking the source and destination processors. Therefore, the time required to transfer a message of size L units between two processors is:

$$T_{msg} = t_s + t_w L$$

According to this cost equation, we can observe the general trends of data in Figure 2 to Figure 5, each of which is basically linear with some small fluctuations predominantly in the early part of the data series. This can be attributed to the effect of segmentation overheads at the IP layer when the message size L is relatively small but larger than the path maximum transmission unit (MTU). The MTU is a link layer restriction on the maximum number of bytes of data in a single transmission. When a message is too large to be sent across a link as a single unit, a router may fragment the message. This effect is less significant as the message size increases. In our experiment, an MTU of Ethernet (1500 bytes) was used.

Another interesting observation is the spread of data series at various values of packet loss. In local networks at 5 ms latency, small packet loss (0% to 2%) does not greatly affect the response time. It becomes more significant when the network latency increases as in Figure 4 and Figure 5. It should be noted that as the network quality deteriorates, the recovery algorithm at the TCP layer takes longer time to successfully transmit the whole segment across the network, therefore the TCP mechanism increasingly dominates the response time.

In the ideal situation, $1/t_w$ is the maximum effective bandwidth or throughput, which is only achievable as the message size approaches infinity. The throughput in different network settings of our experimental results is shown in Figure 6 - Figure 9 as a function of message size. In general, the throughput decreases as the network latency or packet loss deteriorates. We observe ripples of varying degree in our data series as the throughput increases. As mentioned, the application transfers data as a sequence of smaller blocks (in path MTU packet size), rather than attempting to transmit the entire block as a unit. These results demonstrate that varying the size of data block may have significant effect on the throughput. The throughput is generally poor for small message sizes, but

improves for larger sizes. The performance of CORBA is extremely sensitive to the message size, with small variations in the message size producing different throughput. This behaviour is due to the Nagle algorithm, as used by TCP to buffer and aggregate small packets. The Nagle algorithm is effective in preventing flooding of wide area networks with very small packets. However, when used on networks of lower latency, it can have a disastrous effect on the response time of certain applications. For such applications, the only solution is to disable the Nagle algorithm.

As a note of caution, these results were generated according to our experimental set-up (both hardware and software) and the design of our test programs. The actual values of these results should not be used to estimate the performance of other distributed applications over CORBA even in the same network environments.

5. Concluding Remarks

We have attempted to characterise the performance of CORBA in a range of different network environments, in terms of response time and throughput at the application level. The performance of distributed applications running over CORBA is sensitive to network latency and message size. It is even more sensitive to packet loss. Any unreliability in communication can make a system difficult to use — even a normally insignificant level of packet loss can severely degrade an affected application. As a consequence, reliable transmission is extremely important for delivering high performance CORBA-based distributed applications to end-users.

Performance of distributed applications can be characterised by the quality perceived by the users. With the increasing use of CORBA in mission-critical applications, performance issues become very important. In particular, response time is critical in interactive applications, with a target that is usually less than 2 sec. Beyond a typical threshold around 4-5 sec user complaints increase rapidly. For instance, a threshold for one-way delay for voice applications appears at about 150 ms. Beyond this level, the delay causes difficulty for people trying to have a conversation and frustration grows.

The quality levels for packet loss are found to be even more important. Some applications may be able to tolerate a certain level of packet loss. However, beyond 4-6% packet loss, video conferencing becomes irritating. The occurrence of a long delay of 4 sec or more at a packet loss of 4-5% is also irritating for interactive activities such as telnet and X windows. Packet losses beyond 10-12% are unacceptable: there are extremely long timeouts, connections start to be broken and most applications become unusable.

One critical issue influencing performance of distributed computing is communication costs. We can improve performance by reducing the amount of time spent communicating. Clearly, this performance improvement can be achieved by sending less data. It can also be achieved by using fewer messages, even if we send the same amount of data. This is because each communication incurs not only a cost proportional to the amount of data transferred but also a fixed start-up cost. The granularity of objects is associated with the communication costs in querying remote objects. Regardless of network quality, fine-grained objects have a better response time than coarse-grained objects in handling requests. Nevertheless, use of coarse-grained objects implies fewer objects in the

system, thus less object invocations between client and server. Fine-grained objects imply a large volume of objects, but more object invocations to retrieve the same amount of information.

The cost information can be used in developing an architecture for distributed computing. High start-up costs may suggest further agglomeration so as to increase granularity. Similarly, if data transfer costs are high, we may seek to transfer messages of smaller size if doing so can reduce the total volume of data transferred. As the experimental results have indicated, the cost of distributing remote objects increases dramatically as the extent of connections grows beyond national bounds. The granularity of objects should be carefully designed to minimise the overall communication expense in the servers and the underlying network. Remote objects should be located as close as possible to the requesting clients. Distant remote objects should be located such that the network quality to potential clients is adequate. These objects should be designed using a coarse-grained approach so that remote communication with clients is more efficient for a better overall response time.

Distributed object middleware is modelled on calling remote methods such that caller waits for call message to get to remote server and for response message to come back. However, such type of communication is too sensitive to variations in network quality to support distributed applications. Running CORBA-based distributed applications over tenuous networks can easily lead to unacceptable fragility. Hence, use of distributed object middleware outside LAN environments needs careful attention to the granularity of objects and the number of logical transactions between client and server if acceptable performance is to be achieved.

Our findings are based on network statistics collected from the Internet, whose network quality is probably better than that of many military networks, especially in the tactical domain. A robust middleware infrastructure is a necessary underpinning for effective distributed collaboration across the range of military communication environments. While distributed object middleware is basically designed for closely coupled objects using a synchronous style of communication, message oriented middleware is more suitable for loosely coupled objects with no end-to-end transactions. Messaging technology uses asynchronous communications between processes to provide reliable delivery of messages despite network or object failures. Current work is based on assessing the relative tradeoffs between connection-oriented and connectionless (messaging) technologies. Our aim is to examine (i) how distributed applications can be implemented using a loosely coupled asynchronous style of communication, (ii) the improvements in robustness gained through use of current business-quality messaging technologies to communicate between distributed elements, and (iii) any ensuing disadvantages in terms of overall complexity and speed of response.

6. References

[1] CORBA: The Common Object Request Broker: Architecture and Specification, Revision 2.3: Object Modelling Group, 1998.

[2] D. Miron, and S. Taylor, “Performance Characteristics of a Java Object Request Broker,” DSTO Report DSTO-TR-0696, June 1998.

[3] Internet Traffic Report, Andover News Network, http://www.internettrafficreport.com

[4] A.S. Gokhale, and D.C. Schmidt, “Evaluating CORBA Latency and Scalability over High-Speed ATM Networks,” IEEE Transactions on Computers, April 1998.

[5] NIST Net Home Page, http://snad.ncsl.nist.gov/itg/nistnet/

[6] VisiBroker for C++ Programmer’s Guide, Version 4.0, Inprise Corporation, 2000.

[7] Ian T. Foster, “Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering,” Addison-Wesley, Feb 1995.

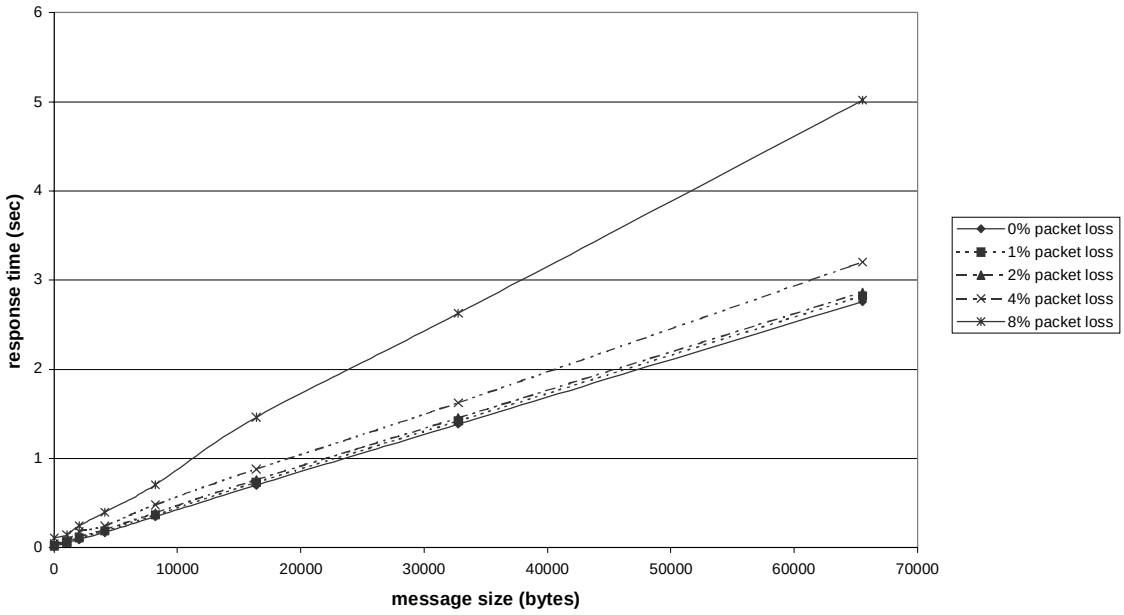

Figure 2: Response time in typical local networks (network latency 5 ms)

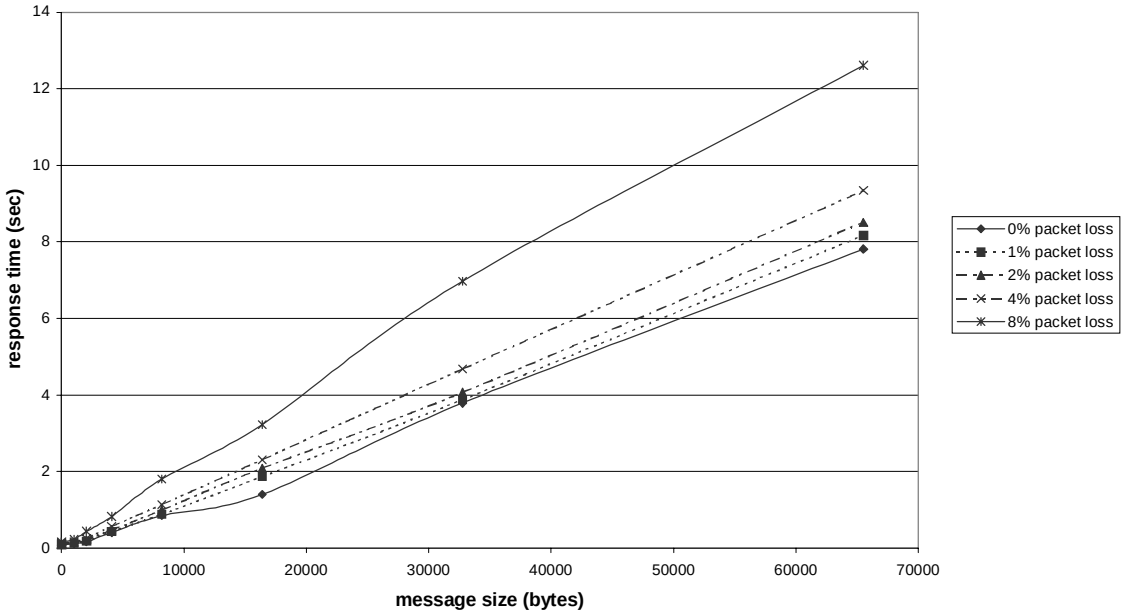

Figure 3: Response time in typical national networks (network latency 40 ms)

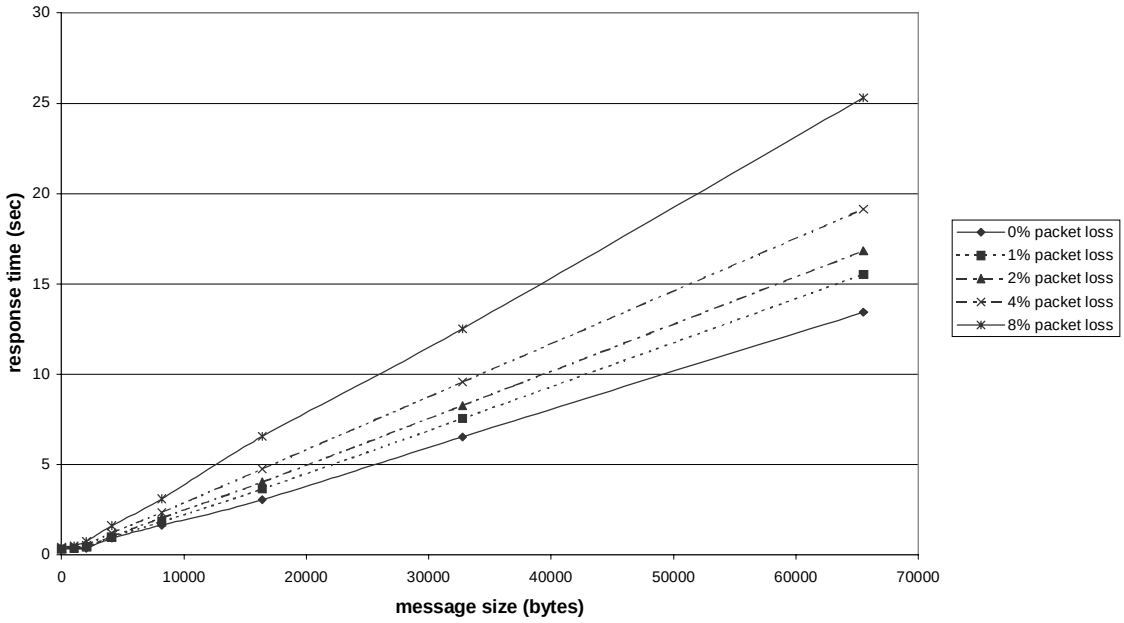

Figure 4: Response time in typical international networks (network latency 150 ms)

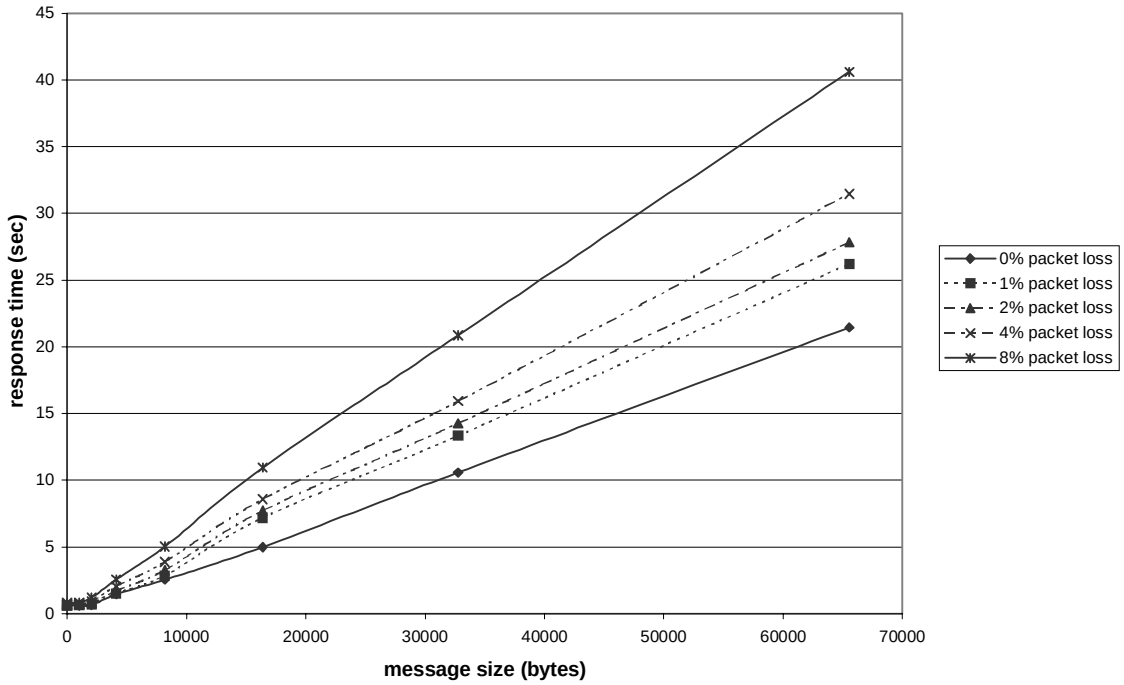

Figure 5: Response time in typical satellite connections (network latency 300 ms)

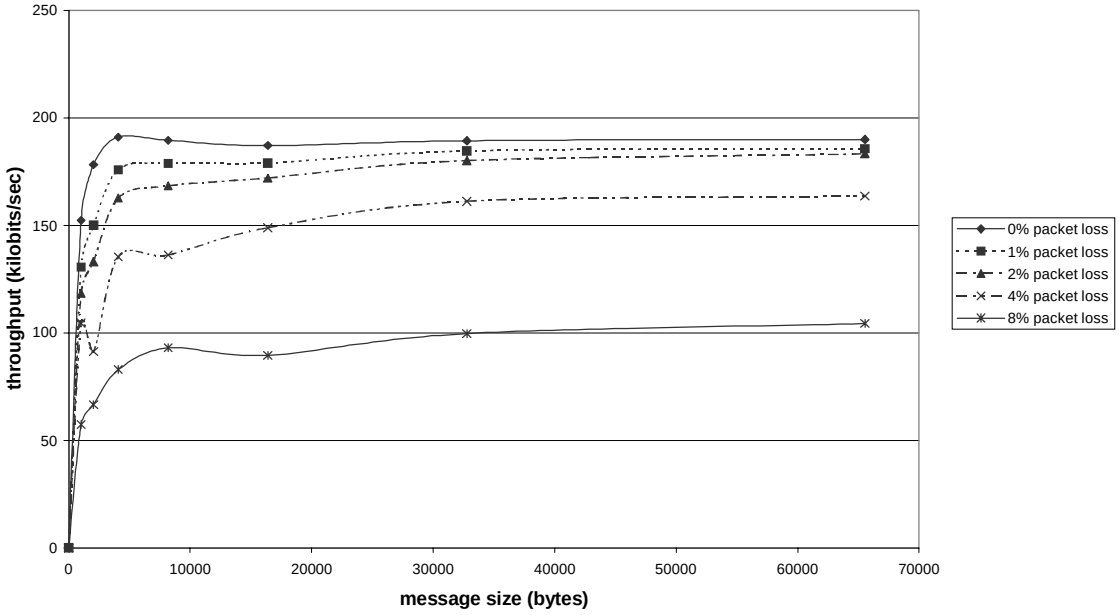

Figure 6: Throughput in typical local networks (network latency 5 ms)

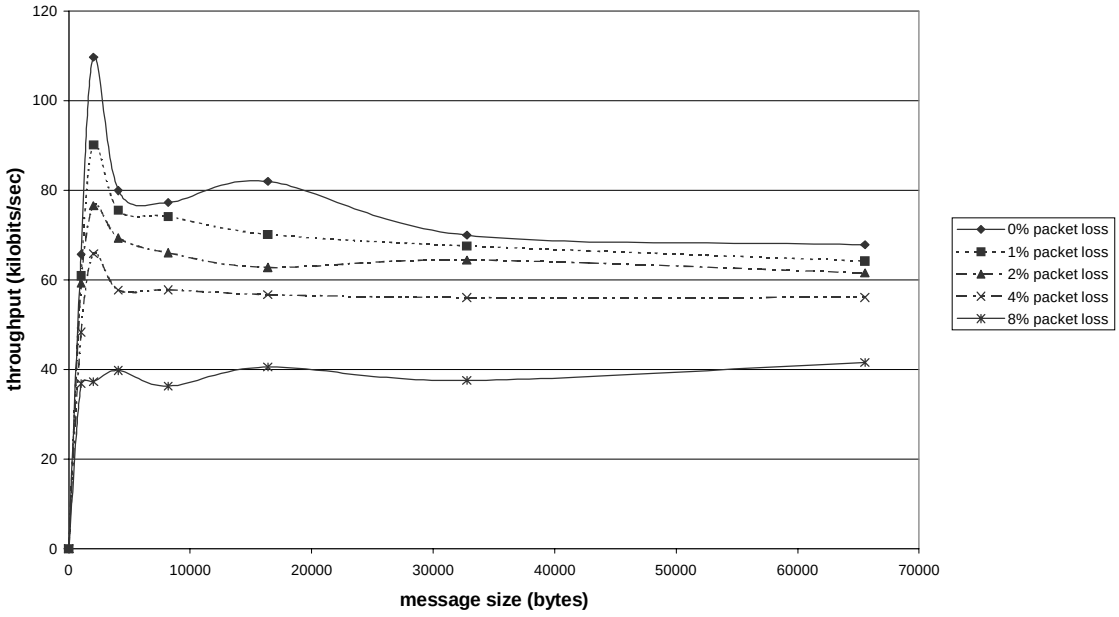

Figure 7: Throughput in typical national networks (network latency 40 ms)

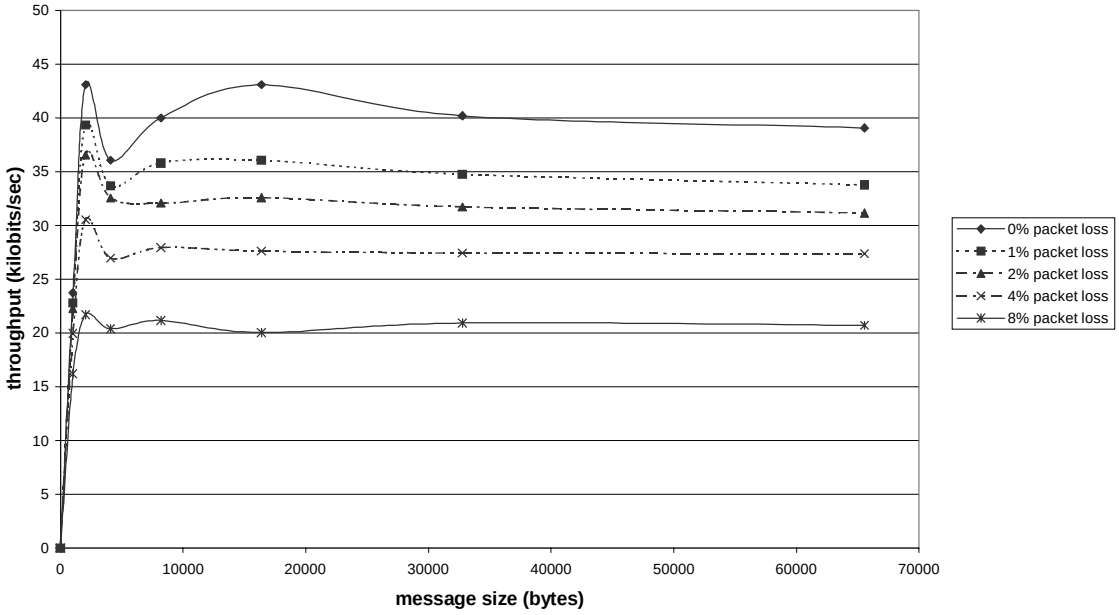

Figure 8: Throughput in typical international networks (network latency 150 ms)

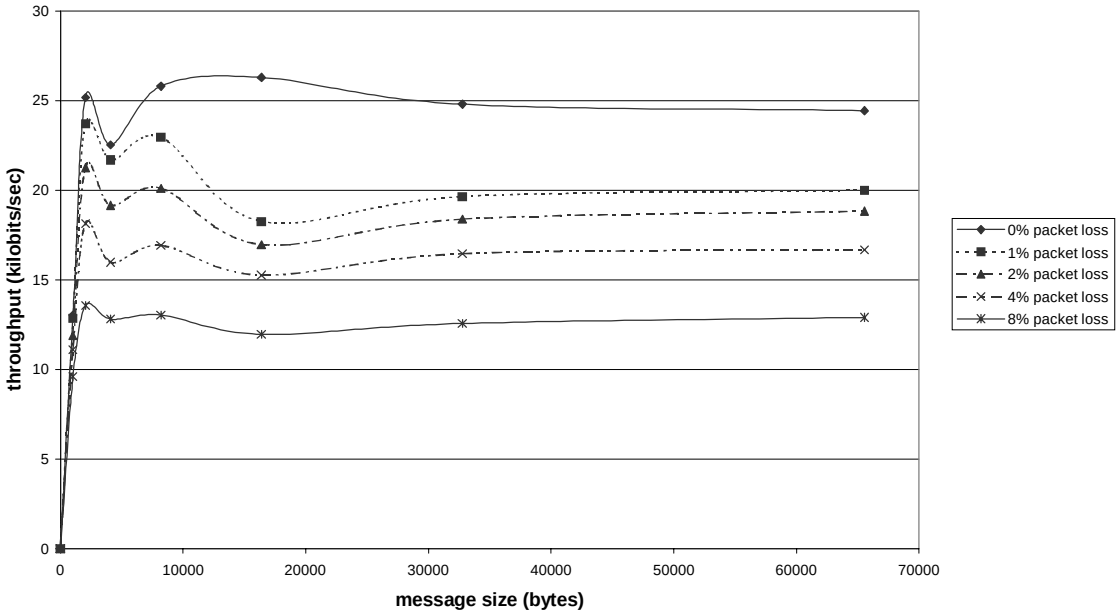

Figure 9: Throughput in typical satellite connections (network latency 300 ms)