Performance of Networked XML-Driven Cooperative Applications*

Shahram Ghandeharizadeh, Christos Papadopoulos, Min Cai, Krishna K. Chintalapudi
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA

Abstract

Web services are an emerging software technology that employ XML, e.g., W3C's SOAP [1], to share and exchange data. They are a building block of cooperative applications that communicate using a network. They may serve as wrappers for legacy data sources, integrate multiple remote data sources, filter information by processing queries (function shipping), etc. Web services are based on the concept of "software and data as a service". With those that interact with an end user, a fast response time is the difference between the following two scenarios: (1) users issuing requests, retrieving their results, and visiting the service repeatedly, and (2) users issuing requests, waiting for response and walking away prior to retrieving their results, with a lower likelihood of issuing future requests for this web service. One may employ a middleware to enhance performance by minimizing the impact of transmission time. This is accomplished by compressing messages. This paper identifies factors that this middleware must consider in order to reduce response time. In particular, it must ensure the overhead of compression (increased CPU time) does not exceed its savings (lower transmission time).

1 Introduction

XML is emerging as the standard for data interoperability among web services and cooperative applications that exchange and share data. Typically, a computer network, either a private (intranet) or a shared one (Internet), is a central component with different cooperative applications residing on different machines. The available network bandwidth of these environments is limited and has a significant impact on the response time of data-intensive applications that exchange a large volume of data. This is specially true for the Internet. As a comparison, a magnetic disk drive supports transfer rates ranging in tens of megabytes per second, e.g., the multi-zoned 180 Gigabyte Seagate Barracuda disk drive (model ST1181677LWV) with an Ultra SCSI 160 interface supports a transfer rate ranging from 20 to 60 Megabytes per second (MBps) depending on the placement of data [8]. The transfer rate of an array of disk drives [3, 5] may exceed gigabytes per second. On the other hand, network connections offering 12.5 MBps (100 Megabits per second, Mbps) are common place at the time of this writing, with the next generation (gigabit networks) supporting hundreds of MBps being deployed. The Internet connections typically range from a few hundred Kilobits (Kbps) to several Mbps. More importantly, the latency involved in network transfers can be substantial. At best, the propagation latency between east and west coast of the United States is approximately 60 milliseconds. Latencies as high as one second are often observed by the Internet users.

The focus of this paper is on transmission of XML data and the role of compression to enhance performance. Two popular metrics used to quantify the performance of a computing environment are: response time and throughput. Throughput denotes the number of simultaneous active transmissions supported by the environment. Response time is the delay observed from when a client issues a request for an XML data source to the time it receives the last byte of the referenced data. Obviously, the objective is to maximize throughput and minimize response time (less wait time). Unfortunately, a higher throughput does not mean a lower response time. For example, one may compress messages in order to increase the throughput of a shared network device (desirable). However, if this is a small message, the CPU overhead of compressing and decompressing messages may increase response time (undesirable).

The focus of this paper is on response time of transmitting XML formatted data and techniques to enhance it. One approach is to minimize the amount of transmitted data. In particular, XML formatted data includes repeated tags and labels that can be compacted using loss-less compression techniques, such as Zip or XMill [11]. In [2], we used TPC-H benchmark to compare the response time and throughput

*This research is supported using an un-restricted cash gift from Microsoft research labs, see http://dblab.usc.edu/WebServices.

of a private network with these two alternative compression techniques. This study focused on a configuration consisting of two PCs connected using a private 100 Mbps switch with no background load. Its main objective was to compare a binary encoding mechanism with an XML-based one using alternative compression techniques. It introduced an analytical model to compute the throughput of the system with these alternatives. It observed that while compression reduces message size significantly, its CPU overhead resulted in a higher response time for an unloaded network.

In this study, we extend our analysis to a more realistic setting: a collection of nodes that communicate using the TCP protocol [6] in an Internet setting. Our objective is to develop a methodology that enables a software middleware to make intelligent decisions when transmitting XML formatted data. This middleware resides between applications and the underlying network protocol. It gathers information about the underlying network and decides when to compress messages (using either Zip or XMill) with the objective to minimize response time. One intelligent decision is to determine whether compressing a message prior to its transmission reduces response time. Using TPC-H benchmark with a realistic network simulation model, we show the following factors impact this decision: 1) compression factor, 2) network loss characteristics, 3) round trip time between source and destination. With TCP, for example, if the original message size is smaller than the maximum segment size (MSS) of the TCP connection then compression does not reduce response time. With larger messages, a higher compression factor provides greater savings. With higher bandwidth networks that incur neither a high latency nor a high loss rate, the overhead of compression might outweigh its benefit. These qualitative and quantitative observations serve as the foundation for future studies that introduce heuristics to improve the performance of network-centric, data-intensive, cooperative computing environments that employ XML. From a commercial perspective, they might be realized in a run-time library for applications developed using Internet development frameworks such as Microsoft's .NET [13], IBM's Web Sphere, etc.

The rest of this paper is organized as follows. Section 2 provides an overview of compression schemes and two variants of the TCP protocol. In Section 3, we report on our network simulation model and experimental platform. Section 4 reports on out obtained results and key observations. We conclude with a summary of learned lessons in Section 5.

2 An Overview of Compression Techniques and TCP

The response time of transmitting the output of a web service (a query) consists of the time to compress the message at the server, transmit it across the network, and decompress it at a client node. With XML, we analyze two compression schemes: Zip/GZip library and XMill [11]. Both employ techniques based on Lempel-Ziv [15]. The key difference between the two is that XMill employs the semantic information provided by XML tags to (a) group data items with related meaning into containers, and (b) compresses each container independently [11]. This column-wise compression is generally better than row-wise compression [10] for large message sizes.

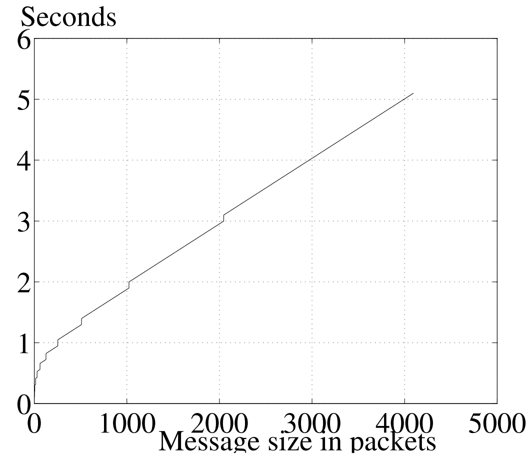

Figure 1. Transmission time as a function of message size using a loss-less TCP connection, b = 10 Mbps, MSS = 1024 bytes, RTT = 100 ms.

A server transmission using TCP comprises of two distinct phases: (1) bandwidth discovery, and (2) steady state. The former occurs at the inception of a connection when TCP discovers the rate at which it can transfer data without causing congestion. When TCP receives the acknowledgment for each successfully transmitted packet, it increases its window size by one packet. Assuming the window size is N and TCP receives N acknowledgments in one Round Trip Time (RTT), this doubles the window size to $2 \times N$ causing TCP to transmit twice as much data before waiting for acknowledgments. During the steady state, TCP transfers data at the maximum transfer rate it discovered during phase 1. Typically, the steady state is achieved when the window size is $b \times RTT$ where b is the bandwidth of the TCP connection.

To illustrate, assume: a) the server is trying to transmit a message with size m, b) the maximum TCP segment size (packet size) is MSS. During the discovery phase, after k RTTs, TCP would have transferred $2^{k+1} - 1$ packets. By the time TCP reaches its steady state, it would have transmitted $\frac{2 \times b \times RTT}{MSS} - 1$ packets. If $m < (2 \times b \times RTT) - MSS$, then the time required to transfer the data would be approximately $RTT \times \lfloor (log_2 \frac{m}{MSS} + 1) \rfloor + \frac{m}{b}$. During the steady state, the window size remains fixed. Figure 1, depicts the variation of time taken to transfer a message assuming a loss-less TCP connection. Figure 1 shows a 3 second transmission time for a message that is 2000 packets in size. If the message size is 4000 packets, one would expect a six second transmission time. However, Figure 1 shows a five second transmission time. This is because the window size has increased, reducing the delays encountered by the server's TCP waiting for acknowledgments.

Network transfer time of a message depends not only on factors like transmission delay and propagation delay, but also on the interaction of TCP with loss. Multiple losses typically throw TCP into long timeouts, significantly increasing transmission time.

There are several variants of TCP deployed in the Internet today. Most of the widely deployed TCP implementations are based on TCP Reno [6]. This TCP variant includes fast recovery and fast re-transmit, which improve performance significantly over vanilla TCP. With fast recovery, when a packet is lost and the sender receives three duplicate acknowledgments for the previous packet, the sender immediately re-transmits the lost packet and cuts its congestion window to half. With fast recovery, subsequent acknowledgments (beyond the three duplicates) inflate the congestion window temporarily. These mechanisms dramatically speed up TCP recovery for a single lost packet. However, when multiple losses occur in a window (e.g., loss bursts), TCP times out. After waking up from a timeout, TCP drops its congestion window to one and re-enters the slow start phase.

TCP timeout values depend on the RTT of the connection and its variance. Timeout is typically a small multiple of the current RTT. There are two exceptions: (a) when the variance is high, in which case it can dominate the RTT calculation, and (b) when the same packet is lost multiple times, which doubles the previous timeout value. Thus, a timeout almost always incurs a high performance penalty for TCP.

A recent TCP variant, TCP SACK (TCP with selective re-transmission) attempts to address the above limitation. TCP SACK acknowledges packets selectively, and thus allows the sender to re-transmit multiple lost packets in a single window. With TCP SACK, endpoints negotiate additional parameters at the beginning of the connection. The use of TCP SACK is not widespread as yet, but is expected to quickly increase as more TCP stacks become SACK-capable.

Another factor contributing to the network's transmission time (particularly for small transfers) is the TCP slow start mechanism. At the beginning of every connection, TCP attempts to quickly discover the available bandwidth in the path by doubling its window during every RTT. Thus, a small message consisting of a few packets may still take several RTTs to be transmitted even though plenty of network bandwidth is available.

In sum, the following network factors contribute to the response time of delivering a query output:

- output data set size
- network bandwidth
- connection RTT and RTT variance
- network loss characteristics (rate, burstiness)
- the employed TCP implementation (Reno versus SACK)

3 Experimental Platform

We analyze the impact of network parameters using the 1 gigabyte TPC-H benchmark [14], a decision support benchmark with documented queries and data sets. TPC-H includes both retrieval and refresh queries. The refresh commands generate large requests and small responses. The retrieval queries offer a mix of commands that generate either (a) large requests and small responses, and (b) large requests and large responses. This motivated us to focus on retrieval queries and ignore refresh queries from further consideration. We report on 21 out of 22 queries because we could not implement query 15 in a timely manner. With the reported response times, we focus only on compression, decompression, and transmission times. We eliminate the query execution time all together.

For our simulation experiments we used ns [7], which is a discrete event simulator that provides several TCP variants. Two types of environments for network simulations were used. Their main difference lies in the topology used, which in turn impacts the way losses occur. The loss is uniform in the first environment, whereas the background traffic determines the losses in the second. The two environments are shown in Figure 2.

Environment 1: Uniform Loss

This simple experimental environment offers a high degree of control, enabling us to manipulate each system parameter individually to study its impact. The experiment is as follows: two nodes are created, one is a TCP source and the other is a TCP destination. We vary the link delay, loss and bandwidth and measure transfer time for different message sizes, as follows:

(a) pick values for delay, bandwidth and loss for the link

(b) decide which TCP variant to use (Reno or SACK)

(c) compute the time required to transfer messages of various sizes, representing uncompressed, Zip-compressed and XMill-compressed messages. We start the timer when the first SYN packet is sent and stop the timer when the last packet is acknowledged. The difference between these two is the network transmission time.

Environment 2: Loss Based on Background Load

The previous experiment forms a good baseline to observe response time as a function of link delay, loss and bandwidth. However, it is not realistic because most Internet loss is caused by congestion. More specifically, Environment 1 fails to capture:

(a) Loss attributed to bursts: When a drop-tail queue (the most commonly used type of queue) overflows at a router, all future arriving packets are discarded. If these packets happen to belong to the same flow (e.g., if they belong to the packet train forming a single window), then a burst of packets will be discarded. In contrast, uniform loss spreads losses more evenly.

(b) Variable delay: as congestion builds up, the queues at the routers fill up increasing the delay between the two endpoints.

(c) Variable bandwidth: the actual bandwidth captured by a TCP flow is inversely proportional to its RTT. With the Internet, a TCP flow competes with other TCP flows which may have drastically different RTTs, and in turn, modulate the bandwidth available for a particular flow.

(d) Heterogeneous flows: TCP long flows facilitate transmission of most of the Internet data. Most traffic are Web flows which behave more aggressively than long TCP flows, impacting network characteristics significantly.

Environment 2 captures the above characteristics, see Figure 2. It employs the familiar dumbbell topology with approximately 100 TCP flows to generate a heterogeneous mix of background traffic. The traffic mix includes both web and FTP flows, with a wide range of RTTs. To select RTTs, we analyzed the distance between each node in a large map of the Internet (56000 nodes) [9] to create a probability density function (PDF). The link delays for each background flow, was then selected from this PDF.

Each experiment was carried out similar to Environment 1, described above. The simulation parameters and their sample values are shown in table 1 The major differences are (a) no loss rate was specified, and (b) the simulation was allowed to "warm up" (i.e., background load to stabilize) before measuring the reported transfer times.

Parameter	value
Number of nodes in source pool	100
Number of nodes in destination pool	100
Delay of bottleneck link	1 ms
Bandwidth of bottleneck link	10 Mbps
Bandwidth of links connecting nodes in source pool to node 1	100 KBps
Bandwidth of links connecting nodes in destination pool to node 2	100 KBps
Path MTU for all links	1000 bytes
Delay between node 2 and a destination node d	1 ms

Table 1. Parameters of Environment 2.

We vary the bandwidth, round trip time and the size of the file to determine their impact on the time to transfer the file. In all our experiments we generate the background traffic based on the ns web traffic model, which simulates users browsing the Internet. A page comprises several objects. Clicking on each object leads to a file transfer whose size is drawn from a heavy tail distribution (Pareto). The time interval between initiating the transfer (clicking objects) is exponentially distributed. The time interval between the user changing over to another page is also exponentially distributed. Hence flows are generated randomly from nodes in the source pool to nodes on the destination pool.

To capture the heterogeneity in delay between nodes in the Internet we adopted the following technique. We obtained router level topology maps containing about 5600 router level nodes from the Mercator Internet mapping project (http://www.isi.edu/scan/mercator/mercator.html). The router level topology is represented as an undirected graph, where nodes represent routers in the Internet and the links are bidirectional. Let the graph $G = (V, E)$, where V is a set of nodes and E is a set of edges. From this topology we generated a model for node distance in the form of a probability density function (PDF) in the following manner. First, we performed a breath first search at every node $v \in V$ and determined the shortest distance in hops for every pair of nodes (v_1,v_2), $v_1, v_2 \in V$. Let the total number of node pairs be P and let the number of node pairs with k hops between them be p_k. We obtain the probability density function (PDF) $f(k) = \frac{p_k}{P}$. Figure 3 shows the PDF obtained after analyzing the topology in this manner. To create our simulation topologies, each time we add a new flow we draw a sample from the PDF to determine the distance between the new source and destination. The procedure is as follows:

Let, N_{src} be the set of all nodes in the source pool and N_{dst} the set of all nodes in the destination pool. Let $h_{x,y}$ represent the number of hops between nodes x and y. The delay $D(x, y)$ between nodes x and y are set as $dh(x, y)$,

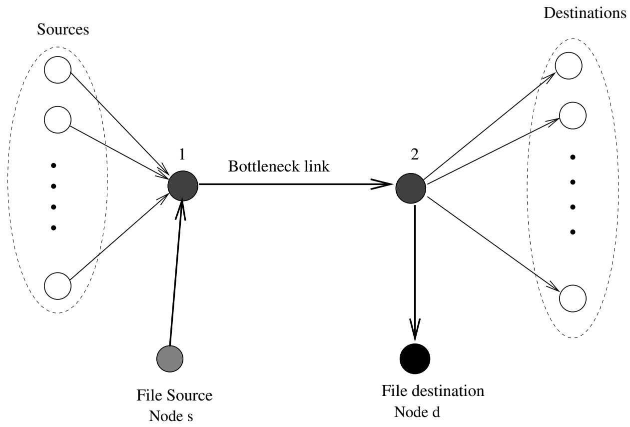

Figure 2. The bottleneck topology used for Environment 2, the ns simulations.

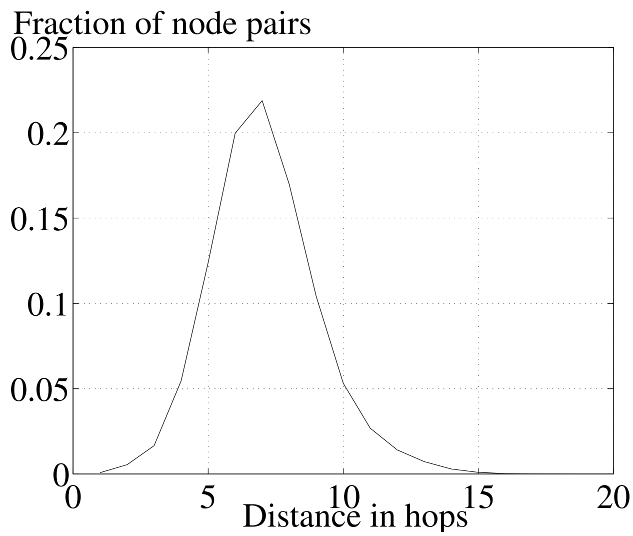

Figure 3. Example pdf of hop distance between any two nodes in the Internet

where d is a fixed delay per hop.

1. for each pair $< n1, n2 >$, $n1 \in N_{src}$ and $n2 \in N_{dst}$.
2. draw $h(n1, n2)$ randomly using the density function in Figure 3.
3. draw $h(n1, 1)$, a uniformly distributed random natural number from $1, \cdots, h(n1, n2) - 1$.
4. set $h(2, n2) = h(n1, n2) - h(n1, 1)$.
5. calculate $D(n1, 1)$ and $D(2, n2)$ as $dh(n1, 1)$ and $dh(2, n2)$ respectively. We used $d = 20$ ms.

The parameters varied in Environment 2 are the following:

1. The bandwidth between node s to node 1.
2. Round trip time (rtt_f) between the source (node s) and destination (node d) node. Since the delays of the bottleneck link and the delay between node 2 and node d are already fixed to 1 ms, the only variable is the delay $D(s, 1)$. W chose $D(s, 1) = \frac{rtt}{2} - 2$ ms.
3. The size of data to be transfered, S. This is determined using TPC-H queries.

In our experiments we explored 9 bandwidth-delay combinations. We used the values (100Kbps, 1Mbps and 10Mbps) for bandwidth and (1000ms, 200ms and 20ms) for the round trip time. For each of the 9 combinations we ran simulations for message sizes S based on those shown in Table 2. These message sizes correspond to the amount of data produced by each TPC-H query.

4 Performance Results

By manipulating parameters of Environments 1 and 2, we were able to analyze response time as a function of different system parameters. The obtained results showed three key observations. In the following, we describe each and present selective experimental results that demonstrate each observation.

Observation 1: In order to decide between an uncompressed transmission and a compressed transmission of a message, one must consider the overhead of compression and decompression (CPU time) and the network transmission time savings offered by the reduced message size (after compressing the message). If the overhead is less than savings, see Table 2, then compression should be employed. Table 2 shows the message size with no compression, Zip and XMill compression techniques. The first column is the TPC-H query id. The rows are sorted in ascending order of uncompressed message size. With Zip and XMill, Table 2 shows the compression and decompression times. The percentage improvement column compares the response time of Environment 1 with and without a compression technique, termed RT and RT_{Comp}, respectively. The percentage improvement is computed as follows: $\frac{RT - RT_{Comp}}{RT_{Comp}}$. The network configuration is TCP-Sack with the following parameter values: 100 Kbps bandwidth, zero loss rate, and a 1 millisecond latency.

		Zip Compression				XMill Compression			
Query ID	MSG size (S)	MSG size (S)	Comp Time	Decamp Time	% Improv-ement	Msg size (S)	Comp Time	Decomp Time	% Improv-ement
17	779	330	0.82	0.98	-18.36	407	1.56	1.56	-28.09
14	780	331	0.98	1.42	-23.05	407	1.56	1.56	-28.09
19	782	333	1.56	0.98	-24.09	415	1.56	1.56	-28.09
6	785	336	1.30	1.04	-22.66	415	1.56	1.42	-27.16
8	1,007	359	1.20	0.87	-20.56	443	1.56	1.56	-28.09
12	1,118	385	1.56	0.98	791.92	459	1.56	1.56	744.94
4	1,700	459	1.56	1.56	744.95	514	1.56	1.42	755.87
5	1,810	486	1.04	1.12	825.40	543	1.56	1.20	773.25
7	2,040	474	1.20	0.87	833.47	539	1.56	1.56	774.94
22	2,364	513	1.20	0.92	1619.20	577	3.13	0.87	1350.84
1	3,053	684	6.25	1.04	1037.87	738	17.19	1.56	550.47
18	4,802	783	1.56	1.56	2902.25	843	1.56	1.56	2902.25
13	8,260	861	1.56	1.12	6024.41	810	3.13	1.42	5113.04
20	33,811	5,115	3.13	1.56	683.61	4,593	10.94	1.56	665.95
9	49,405	4,298	4.69	1.56	1,056.21	3,532	12.50	1.56	1,367.57
21	86,505	5,304	6.25	3.13	1,528.34	4,127	18.75	1.56	1,845.74
11	246,958	21,398	21.88	4.69	1,044.63	16,096	42.19	3.13	1,370.46
2	341,462	48,058	35.94	7.81	600.25	37,656	57.81	6.25	798.40
3	3,489,037	242,348	270.31	37.50	1,318.23	198,249	509.38	23.44	1,603.41
16	5,983,442	302,593	445.31	46.88	1,840.53	225,648	912.50	32.81	2,424.64
10	25,565,892	4,119,445	2532.81	615.63	514.76	3,205,939	4220.31	410.94	301.81

Table 2. Response time improvement using compression techniques; compression and decompression times are in milliseconds using a 2 GHz processor. Network bandwidth = 100 Kbps, latency = 1 ms, loss rate = 0.

		Zip Compression				XMill Compression			
Query ID	MSG size (S)	MSG size (S)	Comp Time	Decamp Time	% Improv-ement	Msg size (S)	Comp Time	Decomp Time	% Improv-ement
17	779	330	3.88	0.43	-68.304276	407	7.55	2.28	-83.09383
14	780	331	3.88	0.43	-68.304276	407	7.55	2.27	-83.07952
19	782	333	3.88	0.44	-68.35443	415	7.55	2.28	-83.09383
6	785	336	3.89	0.43	-68.35443	415	7.56	2.27	-83.09383
8	1,007	359	3.93	0.48	-68.798744	443	7.70	2.34	-83.388695
12	1,118	385	3.97	0.50	-22.720253	459	7.79	2.38	-58.915367
4	1,700	459	4.14	0.55	-25.26159	514	8.13	2.51	-60.443043
5	1,810	486	4.06	0.54	-24.242426	543	8.15	2.53	-60.567825
7	2,040	474	4.09	0.59	-25.14971	539	8.30	2.63	-61.330242
22	2,364	513	4.17	0.61	-11.50443	577	8.59	2.72	-54.921112
1	3,053	684	4.39	0.67	-15.01416	738	9.07	2.94	-57.17345
18	4,802	783	4.71	0.85	19.047617	843	10.09	3.36	-41.747578
13	8,260	861	5.29	1.12	42.687275	810	12.11	4.28	-34.747147
20	3,3811	5,115	12.70	3.80	25.490206	4,593	35.25	12.06	-43.171726
9	49,405	4,298	15.66	4.28	55.494133	3,532	52.37	16.47	-41.436745
21	86,505	5,304	22.49	7.24	91.06636	4,127	69.56	28.67	-30.989456
11	246,958	21,398	66.02	19.70	87.52321	16,096	178.63	77.93	-26.427736
2	341,462	48,058	113.02	35.53	44.37809	37,656	254.53	100.34	-28.69418
3	3,489,037	242,348	885.53	270.67	106.39019	198,249	2371.41	995.85	-20.793142
16	5,983,442	302,593	1428.23	432.98	127.30181	225,648	4258.62	1793.28	-23.166965
10	25,565,892	4,119,445	8985.40	2774.65	35.827183	3,205,939	19235.02	7267.31	-29.631702

Table 3. Response time improvement using compression techniques; compression and decompression times are in milliseconds using a 450 MHz processor. Network bandwidth = 10 Mbps, latency = 1 ms, loss rate = 0.

Table 2 shows a negative response time improvement with both Zip and XMill when the uncompressed message size is less than 1 kilobyte, i.e., less than or equal to 1007. Compression provides no benefit because, during the discovery phase, TCP decided a window size of one packet whose maximum size of 1024 bytes. This means that messages smaller than 1024 bytes are transmitted as one packet and require approximately the same transmission time. Once the uncompressed message size exceed 1 packet size, compression starts to improve response time by reducing the number of transmitted packets. Note that compression time is typically greater than decompression time with both Zip and XMill. Moreover, both of these values depend on the characteristics of the input data.

In Table 2, the percentage improvement in response time is not necessarily a function of message size, e.g., query 13 produces less data than query 10 and observes a higher percentage of improvement. Instead, it is a function of compression factor, the time attributed to compressing and decompressing a message, and the network characteristics (bandwidth, loss rate, latency). A higher compression factor increases savings in network transmission time. In our example, the compression factor with query 13 is ten while that of query 10 is five. As we increase the bandwidth of the underlying network to 1 Mbps and 100 Mbps, the overhead of compression becomes significant. With a 10 Mbps network connection, the highest percentage improvement observed is with Zip with query 16 (548% improvement). Moreover, with XMill compression, query 1 observes the highest negative percentage improvement in response time (-71%) because its relatively high compression overhead dominates response time. With a 100 Mbps network bandwidth, the percentage improvement observed is negative for all queries.

We also analyzed system response time with different processor speeds (2 GHz, 1 GHz, and 450 MHz). A slower processor speed (say 450 MHz) increases the compression and decompression times. If the network provides a poor performance (e.g., low bandwidth, high loss rate, high latency, or a combination of these) then compression continues to improve response time significantly (similar to those shown in Table 2). With a high bandwidth network connection that provides a zero loss rate and 1 ms latency (a local area network), the benefits of compression diminishes, see Table 3. For example, with a 450 MHz processor configuration (for both the server and client), XMill compression provide no response time benefits. The overhead of compression also reduces the percentage improvements offered by Zip.

Observation 2: Compression improves response time by a wider margin with higher network loss rates, and higher network latencies because it minimizes the impact of these factors by transmitting fewer network packets. Once again, both Environment 1 and 2 were supportive of this observation. This section draws upon Environment 2 to substantiate this observation. It is important to note that the quantitative results can be presented in a qualitative manner. This is because even though a compression technique may reduce message size significantly, the simulated background load may increase transmission time significantly (as compared to an uncompressed transmission that encounters a light background load). With a given processor speed (say 450 MHz) and an ideal network characteristic (high bandwidth, e.g., 100 Mbps, low loss rate and latency), both Zip and XMill compression technique degrade response time. Compression starts to improve performance for data intensive queries when one increases either the loss rate, latency, or both. For those queries whose produced data is less than one network packet (1024 bytes), compression provides no benefit, see the discussions in Observation 1. The factor of improvement for a query depends on the compression factor, encountered background load, and characteristics of the link between N_s and N_d. With higher processor speeds (1 and 2 GHz), the percentage improvement in response time observed with compression techniques increases.

Observation 3: When compared with TCP-Reno, TCP-SACK improves performance by approximately 10% when either the loss rate is high, the network latency is high, or both. With a high network latency, TCP SACK employs large windows, not available in Reno, to prevent many of the time outs. With a high network loss rate, TCP SACK minimizes many of the timeouts that would have occurred with Reno. With selective acknowledgments, TCP SACK can precisely convey information about multiple lost packets to the sender, which in turn allows the sender to quickly send retransmissions to fill holes (if any). With TCP Reno, the sender must receive at least three duplicate acknowledgments before initiating a retransmission. It has been observed that TCP-Reno will go into timeout whenever two or more packets are lost within a window [6].

5 Conclusion and Future Research Directions

A computer network is a central component of many cooperative, data intensive applications. Binary and XML are two popular encoding mechanisms for formatting data transmitted between these applications. XML has emerged as a popular encoding mechanism for those applications that strive to inter-operate with either the existing or future applications.

We envision a middleware that serves as an intermediary between the underlying network protocol, e.g., TCP, and the cooperative applications that generates data. This middleware analyzes the characteristics of the underlying network and the transmitted data in order to enhance response time by compressing data. Our experimental results demonstrate that a right decision improves transmission time significantly (several orders of magnitude). At the same time, we observe that compressing data does not improve response time at all times. There is a trade off between (a) compression overhead in the form of CPU time to compress and uncompress a message, and (b) savings with a reduced number of network packets. If the network connection between two components is less than optimal, e.g., low bandwidth, high latency, high loss rate, etc., and the amount of transmitted data is significant (more than 1 TCP packet) then compression enhances performance. The exact amount of improvement is a function of compression factor, network characteristics, and the processor speeds. If the server and client processors are not fast then compression and decompression times dominate transmission times. These observations are in accord with those in [11].

In order for a "middleware" to make the right decision, it must estimate: 1) the amount of data to be transmitted, 2) compression time at a server (overhead), 3) decompression time at a target client (overhead), 4) available network bandwidth, 5) network loss rate, and 6) network latency. Items 4 to 6 enable the middleware to estimate the benefits of compression. If the benefits outweigh overheads then the middleware may utilize either Zip or XMill. Item 1 can be estimated by building a profile of submitted queries and their produced result sets. Items 2 is trivial while item 3 is addressed by the target client disclosing its processor speed when establishing a connection. Items 4 to 6 are challenging. In particular, there are no reliable methods to estimate bandwidth [12]. There are cases, however, when the available bandwidth is known. One example is an environment with restricted access bandwidth, such as modems (telephone, DSL or cable). Network latency and loss characteristics are also difficult to estimate. Numerous studies have been carried out, however, the networking community does not yet have a good model to estimate these parameters [4]. Heuristics to estimate these parameters shape our immediate research directions. These heuristics will almost certainly improve the average transmission time over a period of time (and not the transmission time of every request). This is because they must detect and adapt to the changing characteristics of the network. We intend to explore a preliminary design, implementation, and evaluation of our middleware with Environments 1 and 2.

References

[1] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsoh, H. F. Nielsen, S. Thatte, and D. Winder. Simple Object Access Protocal (SOAP). Technical Report 8, World Wide Web Consortium (W3C), May 2000.

[2] M. Cai, S. Ghandeharizadeh, R. Schmidt, and S. Song. A Compression of Alternative Encoding Mechanism for Web Services. In *Submitted for consideration to DEXA Conference*, 2002.

[3] P. M. Chen and D. A. Patterson. Maximizing Performance in a Striped Disk Array. In *Proc. 17th Annual Int'l Symp. on Computer Architecture, ACM SIGARCH Computer Architecture News*, page 322, 1990.

[4] A. Downey. Using Pathchar to Estimate Internet Link Characteristics. *Proceedings of ACM Sigcomm*, September 1999.

[5] A. L. Drapeau, K. W. Shirrif, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. H. Chen, and G. A. Gibson. RAID-II: A high-bandwidth network file server. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 234–244, 1994.

[6] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3):5–21, July 1996.

[7] K. Fall and K. Varadhan. *The ns Manual*. The VINT Project, April 2002. http://www.isi.edu/nsnam/ns/index.html.

[8] S. Ghandeharizadeh, D. Ierardi, and D. Kim. Placement of Data in Multi Zone Disk Drives. In *Proceedings of the Second International Baltic Workshop on Databases and Information Systems*, 1996.

[9] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. *IEEE Infocom*, 2000.

[10] B. R. Iyer and D. Wilhite. Data Compression Support in Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994.

[11] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. Technical Report MSCIS-99-26, University of Pennsylvania, 1999.

[12] V. Paxson. End-to-End Internet Packet Dynamics. *Proceedings of ACM Sigcomm*, September 1997.

[13] D. S. Platt. *Introducing Microsoft .NET*. Microsoft Press, 2001.

[14] M. Poess and C. Floyd. New TPC Benchmarks for Decision Support and Web Commerce. *ACM SIGMOD Record*, 29(4), December 2000.

[15] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.