NAM: A Network Adaptable Middleware to Enhance Response Time of Web Services*

Shahram Ghandeharizadeh, Christos Papadopoulos, Parikshit Pol, Runfang Zhou
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA

Abstract

Web Services are an emerging software technology that employ XML to share and exchange data. They may serve as wrappers for legacy data sources, integrate multiple remote data sources, filter information by processing queries (function shipping), etc. With those that interact with an end user, a fast response time might be the difference between a frustrated and a satisfied user. A Web Service may employ a loss-less compression technique, e.g., Zip, XMill, etc., to reduce the size of an XML message in order to enhance its transmission time. This saving might be outweighed by the overhead of compressing the output of a Web Service at a server and decompressing it at a client. The primary contribution of this paper is NAM, a middleware that strikes a compromise between these two factors in order to enhance response time. NAM decides when to compress data based on the available client and server processor speeds, and network characteristics. When compared with today's common practice to transmit the output of a Web Service uncompressed always, our experimental results show NAM either provides similar or significantly improved response times (at times more than 90% improvement) with Internet connections that offer bandwidths ranging between 80 to 100 Mbps.

1. Introduction

Many organizations envision Web Services (WSs) as an enabling component of Internet-scale computing. A WS is either a computation or an information service with a published interface. Its essence is a remote procedure call (RPC) that consumes and processes some input data in order to produce output data. It is a concept that renders web applications extensible: By identifying each component of a web application as a WS, an organization may combine these WSs with others to rapidly develop a new web application. The new web application may consist of WSs that span the boundaries of several (if not many) organizations. A final vision of WSs is to realize a dynamic environment that identifies, composes and integrates WSs in response to a query [7]. This is similar to how a relational database management system identifies and composes the appropriate relational algebra operator into a query plan to process a SQL command.

The eXtensible Markup Language (XML) produces human-readable text and is emerging as the standard for data interoperability among WSs and cooperative applications that exchange data. Well-formed XML documents consist of elements, tags, attributes, etc., and satisfy precise grammatical rules. The major commercial vendors, e.g., Microsoft, IBM, etc., employ XML to publish, invoke, and exchange data between WSs. A WS publishes its interface using the Web Service Description Language (WSDL). An Internet application may invoke a remote WS using the Simple Object Access Protocol (SOAP). Typically, an invoked WS produces an XML-formatted response. (See [17] for an overview of WS concepts and terminology in the context of an application.)

Binary encoding is an alternative encoding mechanism that produces compact streams for efficient parsing, which are not human readable. A binary formatted message is typically smaller than its XML formatted counterpart. This is because XML encoding includes repeated tags, labels and attributes. One may employ compression in order to reduce the size of both XML and binary formatted messages. Two popular compression techniques are Zip/GZip and XMill [10]. Both employ techniques based on Lempel-Ziv [18]. The key difference 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 [10]. This column-wise compression is generally better than row-wise compression [9] for large message sizes. With XMill, com-

*This research was supported in part by an unrestricted cash gift from Microsoft research.

pressed XML messages are at times smaller than their Zip compressed binary representation. This typically holds true for those messages that are more than one Megabyte in size [10, 3].

The focus of this paper is on minimizing response time for WSs by deciding when compression would be beneficial. Two popular metrics used to quantify the performance of a computing environment are response time and throughput. Throughput denotes the number of active requests processed by the environment in a given unit of time. Response time is the delay observed from when a client invokes a remote WS until it receives the last byte of the response produced by the service. One may wish 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, but for small messages, the CPU overhead for compression and decompression may increase response time.

The primary contribution of this paper is Network Adaptable Middleware (NAM) designed with the objective to enhance response time[1]. NAM is divided into a client and a server component and is designed to scale in environments consisting of millions of clients that invoke a single web service. If a WS is standalone and does not depend on another web service, denoted WS_S, it is configured with NAM's server component. Otherwise, a WS plays dual roles of being a client of one or more web services and a server for others, denoted WS_{SC}, and is configured with both the client and server components. NAM's components might be included as libraries of a software development environment such as Microsoft's .NET in order to be deployed seamlessly when a WS is deployed.

Our experimental results from both Internet and an intranet deployment of NAM demonstrate its feasibility. NAM readily adapts to its environment to use the appropriate transmission paradigm to minimize response time. Experimental results in Section 3 demonstrate that NAM provides significant savings compared to standard uncompressed transmission with representative current deployments for today's Internet with bandwidths in the order of tens of Mbps.

The rest of this paper is organized as follows. In Section 2, we provide an overview of NAM and its components. Section 3 presents an experimental evaluation of NAM. Brief conclusions and future research directions are contained in Section 4.

[1]One may also employ NAM as a stack layer in frameworks such as FALCON [15].

2. NAM

NAM consists of a server and a client component, denoted NAM_S and NAM_C respectively. NAM_C constructs and maintains (a) a profile of messages processed by a client, namely, the time required to decompress a message, and b) a profile of network round-trip-time and loss rate for each contacted server. NAM_S maintains a profile of the time required to compress a message. Profiles gathered by both NAM_S and NAM_C are maintained in a persistent manner to ensure their cumulative growth in the presence of shutdowns, power failures, etc.

The primary advantages of maintaining the profile at the client are: a) the client may customize its estimator based on the characteristics of its hardware and statistical peculiarities of its requested data, b) the server is freed to support millions of clients without incurring the overhead of maintaining a decompression profile for each client. A drawback of this approach is the extra CPU overhead and storage required at a client to maintain the profile, which maybe significant for small, mobile devices. As detailed in Section 2.1, one may use regression in order to maintain a compact representation of these profiles, in the order of tens of bytes. Moreover, the CPU overhead is negligible for clients that are able to perform decompression, and retrieve and process tens of thousands of bytes of data. Experimental results of Section 3 show the benefits of NAM significantly outweigh these overheads.

A WS configured with NAM (acting either as a WS_S or WS_{SC}) continues to inter-operate with legacy clients and Web Services not configured with NAM. This is supported as follows. When a client configured with NAM_C (say a WS_{SC}) invokes a remote WS, its SOAP header includes a flag (along with several other tagged data items required by NAM) denoting the presence of NAM. If the referenced WS is not configured with NAM_S, it ignores this flag and provides an uncompressed output always. If the service is configured with NAM$_S$, it utilizes this flag to transmit its output in either a compressed or an uncompressed manner.

If a NAM enabled WS$_{SC}$ receives a SOAP header without the NAM flag, it assumes the client is not configured with NAM and produces an uncompressed reply. To simplify discussion, and without loss of generality, we assume an environment that consists of two WSs, a WS_S and a WS_{SC} configured with NAM. The WS_{SC} invokes remote methods published by WS_S.

Figure 1 shows the pseudo-code for NAM_S. It consists of a collection of estimation techniques in order to render a decision quickly. When WS_S produces a response M to a request issued by a WS_{SC}, it invokes this pseudo-code with the byte array corresponding to M and WS_{SC}'s SOAP header. NAM_S estimates: a) the size of this message once compressed, b) the time required to compress this message,

```
$NAM_S$ (byte[ ] M, Client $WS_{SC}$) {
  1. $S$ = M.Length();
  2. $S_C$ = Estimate size of M in compressed form;
  3. $T_{Comp}$ = Estimate time to compress M at server;
  4. $T_{Decomp}$ = Estimate time to decompress M at $WS_{SC}$;
  5. Estimate network round-trip time $RTT$ and loss rate $p$
     for network connection between server and $WS_{SC}$;
  6. $RT_U$ = transmission time ($S$, $RTT$, $p$);
  7. $RT_C$ = transmission time ($S_C$, $RTT$, $p$) + $T_{Comp}$ + $T_{Decomp}$;
  8. if ($RT_U < RT_C$) then "transmit uncompressed";
     else "compress and then transmit";
}
```

Figure 1. Pseudo-code of NAM, a network adaptable middleware

Term	Definition
NAM_S	Server component of NAM.
NAM_C	Client component of NAM.
WS_S	A Web Service that is not dependent on other Web Services, configured with NAM_S only.
WS_{SC}	A Web Service that depends on other Web Services, configured with both NAM_S and NAM_C.

Table 1. Terms and their definitions

c) the time to decompress this message at the client, WS_{SC}, and d) the network characteristics. Next, NAM_S employs an analytical model of the underlying network protocol to estimate transmission time with the estimated network characteristics for a given a message size. If the estimated response time using a compression technique is better than an uncompressed transmission then NAM compresses the message and transmits it. Otherwise, the message is transmitted in uncompressed format. If NAM_S includes several compression techniques then it must estimate response time with each and choose the one with the best response time. This trivial extension is not shown in Figure 1.

In Section 2.1, we describe a general purpose technique to estimate compressed message size, compression time, and decompression time as a function of message size. Next, Section 2.2 describes how NAM estimates network transmission time. This model is specific to the TCP protocol [14]. The overall performance of NAM is dependent on the accuracy of these estimation techniques.

2.1. Regression to Estimate Compression Time and Compressed Message Size

This section describes a generic technique to estimate compression time, decompression time and compressed message size for a given message. Of course, there are many ways to perform this estimation and one may develop and deploy an application specific approach. NAM is envisioned as a collection of different plug-and-play components, enabling an application developer to replace our generic technique with their own specific model. Section 3 shows the tradeoff associated with using our generic approach and how it impacts NAM's decisions.

We utilize a generic polynomial regression technique to detect a curvilinear function between a message size and (a) its compressed message size, (b) compression time, and (c) decompression time. In the following, we provide an overview of polynomial regression and its alternative models. The key advantage of regression is that it represents a large sample set with a finite set of variables. NAM_C computes the coefficients of a regression model for decompression time and transmits it to NAM_S to estimate the decompression time of a message at the client.

Polynomial regression computes the relationship (such as linear, exponential, logarithmic, etc.,) between a dependent variable (say y) and an independent variable (say x). Based on Taylor approximation, if the original functions are difficult or impossible to evaluate directly, the partial sums of the corresponding infinite series is polynomials and can be evaluated as follows:

$$y = f(x) = \sum_{i=0}^{\infty} a_i \times x^i = a_0 + (a_1 \times x) + ... + (a_n \times x^n) + ...$$

Linear, quadratic, cubic regression are special cases of this general equation:

$$Linear : f(x) = a_0 + (a_1 \times x) \quad (1)$$

$$Quadratic : f(x) = a_0 + (a_1 \times x) + (a_2 \times x^2) \quad (2)$$

$$Cubic : f(x) = a_0 + (a_1 \times x) + (a_2 \times x^2) + (a_3 \times x^3) \quad (3)$$

Given n observed samples: $\{x_1, y_1\}$, ..., $\{x_n, y_n\}$, a regression model solves for a_i values with the objective to minimize the sum of difference between the estimated value y_i' and its observed value y_i, i.e., minimize $\sum_{i=1}^{n}(y_i - y_i')^2$. Conceptually, this is accomplished by computing the partial derivatives at every point of x_i and set its result to zero. With cubic regression, this is realized by maintaining three matrices, see Figure 2, where Y = X $\times$ A. One may solve for matrix A to obtain the coefficients by computing the inverse of X, i.e., $A = X^{-1} \times Y$.

At run time, the system may accumulate m new samples by maintaining separate X' and Y' matrices. The system may add these into the existing X, and Y matrices and solve

$$X = \begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \sum_{i=1}^{n} x_i^6 \end{bmatrix}$$

$$A = \begin{bmatrix} \sum_{i=1}^{n} a_0 \\ \sum_{i=1}^{n} a_1 \\ \sum_{i=1}^{n} a_2 \\ \sum_{i=1}^{n} a_3 \end{bmatrix} \qquad Y = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \\ \sum_{i=1}^{n} x_i^3 y_i \end{bmatrix}$$

Figure 2. Matrices maintained in support of Cubic regression.

for a new A matrix. The space complexity of this approach is the size of matrices and independent of the total number of samples. Its time complexity is to solve for matrix A: compute the inverse of matrix X and multiply it by matrix Y.

2.2. Network Models

The network characteristics have a significant impact on the response time associated with transmitting the output of a WS. The response time depends on network bandwidth, transmission and propagation delays, loss rate, and the interaction of the transport protocol with loss. In this paper we focus on the TCP [14] transport protocol because of its wide spread use.

The main challenge for NAM is to devise mechanisms to estimate the response time of a server based on information such as round-trip-time (RTT), bandwidth, output size and loss rate. Of these, only the response size is known, thus the remaining variables must be either measured or estimated. NAM builds on existing work on TCP modeling. Accurately modeling TCP over a wide range of network conditions is a challenging issue, and most existing models are constrained with a strict set of assumptions. TCP modeling is also complicated due to the existence of several TCP variants, but not all of them are widely deployed. Such variants include TCP SACK [11], which speeds up loss recovery with selective re-transmission and TCP Vegas [2], which proposes enhanced techniques for congestion detection. The dominant TCP variant today is TCP Reno [5], and thus is the focus of NAM. As other TCP variants become popular, NAM must be enhanced to accommodate them.

Several analytical models exist to estimate response time with TCP Reno. These include models that consider both network loss [12, 16] and no loss [16], termed "LM" (Loss-estimation Model) and "NLM" (No-Loss estimation Model), respectively. We evaluated these two models with both simulation studies using ns2 [6] and experimentation using in a testbed with NIST Net [4]. Some of these results are presented below. In summary, each model has its own strengths and weaknesses: LM is accurate with long flows but inaccurate with short flows (such as those frequently produced by WSs). NLM is accurate at modeling both short and long flows but inaccurate when flows experience loss. Finally, both models become less accurate when response time is dominated by CPU time to transfer data from the user to the kernel space. The two models along with a more in-depth discussion of obtained results are detailed below.

In order to understand TCP's contribution to response time, we first summarize the behavior of TCP. A TCP connection begins with a three-way handshake, which takes up one round-trip time (RTT). For data larger than a single packet, TCP enters the slow start phase, where it increases its transmission window by one for each received acknowledgment (ACK). In the absence of delayed ACKs, the receiver sends an ACK for each new packet. If there is no loss, the sender's window doubles every Round-Trip-Time (RTT) until it reaches a pre-specified limit, termed maximum window size W_{max}, typically bounded by the receiver's socket buffer size. At this point, the sender enters a steady state and continues to transmit W_{max} packets every RTT until the entire message is transmitted. Therefore in the absence of loss we can use the model described by NLM [16] to estimate the transfer time of a message consisting of N packets as follows:

$$T_{trans} = \begin{cases} RTT \times \lfloor log_2 N \rfloor, & if\ N \leq N_{exp} \\ RTT \times (\lfloor log_2 W_{max} \rfloor + \lceil \frac{N - N_{exp}}{W_{max}} \rceil + 2), & else \end{cases}$$

where T_{trans} is the network transmission time, N_{exp} is the number of packets transmitted during slow start phase, N_{exp}=$2^{\lceil log_2 W_{max} \rceil} + W_{max} - 1$.

NAM_C maintains a history of measurements with all the recently contacted servers. This history is kept in a local database, which is consulted before a request is sent to a server. If past experience has shown the network path to be relatively free of loss, or if the response size is small (a few packets) then using the above model is reasonable. In our framework, NAM_C maintains an estimate of RTT for each of the corresponding servers. When communicating with a specific server, NAM_C provides its RTT estimate to NAM_S as a part of its request (e.g., in a SOAP header). Note that having NAM_C communicate this information to NAM_S is more scalable because it frees the server from maintaining a potentially large database for all its clients. NAM_C might estimate the RTT to NAM_S either off-line by using PING, or by monitoring the RTT on recent transactions with the server, or a combination of these. With PING and low bandwidth connections, e.g., DSL, Cable Modem, etc., where transmission time of data dominates RTT, we must ensure that the transmitted ECHO request is padded to the size of a full packet, otherwise, the RTT estimates will be inaccurate.

NAM needs additional parameters to estimate response time, such as the maximum TCP window size and the loss rate. Both are available from the TCP protocol control block that reside in the kernel (a control block maintains the state the protocol requires for each connection). Currently, there is no interface that provides these parameters to the application. Work such as Congestion Manager [1], however, aims to address this limitation by allowing congestion information to be shared among all protocols residing on a machine. For our NAM prototype we modified the Linux kernel to return such information to the application via the getsockopt() system call. This was a simple modification requiring just a few lines of code.

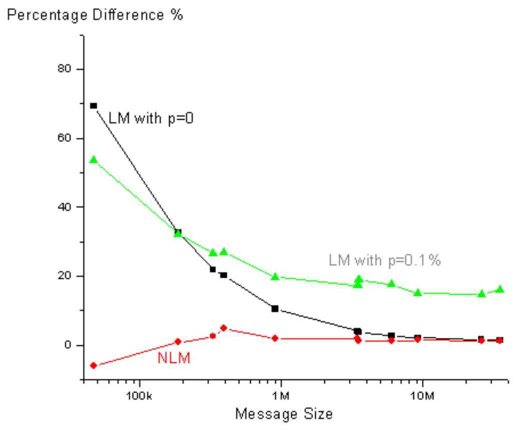
NLM works well if there is no packet loss. Packet loss in TCP may lead to a timeout, and the impact on response time is dramatic because it adds idle time and causes the protocol to enter slow start again. When history indicates that loss is likely during the transmission of a particular large message, we estimate the network transmission time of message M as follows: $\frac{M.Length()}{\beta(p)}$, where $M.length()$ is message size, and $\beta(p)$ is an estimation of network bandwidth. Using the models of [12], the network bandwidth is estimated as:

$$\beta(p) \approx min(\frac{W_{max}}{RTT}, RTT\sqrt{\frac{2p}{3}} + T_0\, min(1, 3\sqrt{\frac{3p}{8}})p(1+32p^2) \quad (4)$$

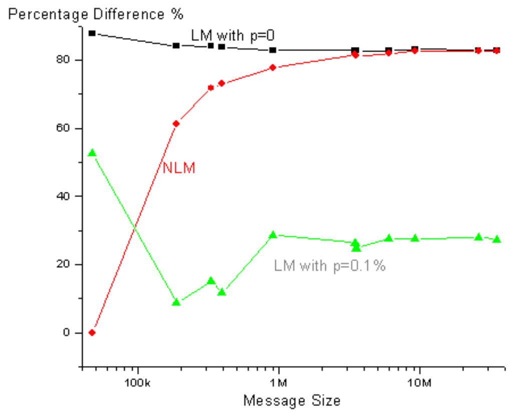
where p is the loss probability and T_0 is the duration of timeout. Equation 4 is appropriate for bulk transmissions which send a large amount of data.

Figure 3 shows the accuracy of both models with different latencies. These experiments were conducted using three machines: a client, a server, and a dedicated NIST Net router. These machines were connected using a 100 Mbps Ethernet switch. The x-axis shows the message size. The y-axis shows the percentage difference between the observed and estimated network transmission times, $100 \times \frac{Observed - Estimated}{Observed}$. With a high network latency and no loss, see Figure 3.a, NLM performs very well. In the presence of loss, this model becomes inaccurate (and eliminated from this presentation). LM does well for large messages (long flows) with and without loss. With short messages (less than 100 Kilobytes), LM exhibits a high percentage of error [12].

Figure 3.b shows the accuracy of each model when network latency is low. With no packet loss, the observed error was high. This is because we estimate RTT using PING. This estimation is inaccurate because it ignores the time required to transfer data from user space to kernel space, see Figure 4. This time becomes significant with large messages sizes ($>$ 100 Kilobyte), resulting in a high percentage of error. One may enhance the accuracy of these models by requiring NAM_C to maintain the observed RTT for the



3.a Latency = 100 msec



3.b Latency = 1 msec

Figure 3. A comparison of analytical models with a 100 Mbps Ethernet switch using NIST Net.

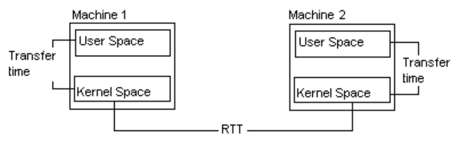


Figure 4. PING does not include the transfer time from user space to kernel space.

Term	Range
Low Latency($\downarrow L$)	$<$ 20 ms
Moderate Latency($\leftrightarrow L$)	20-100 ms
High Latency($\uparrow L$)	$>$ 100 ms
Low Bandwidth($\downarrow B$)	$<$ 10 Mbps
Moderate Bandwidth($\leftrightarrow B$)	10-100 Mbps
High Bandwidth($\uparrow B$)	$>$ 100 Mbps

Table 2. Bandwidth/Latency Terms and their ranges

previous connections with a server. It is important to note that this correction in RTT will enhance the quality of decisions rendered by NAM, but it does not impact it greatly, because it is almost always appropriate to transmit messages uncompressed for environments where RTT is so low that the CPU time or memory transfers become significant. When the RTT is underestimated, NAM underestimates the total transmission time, motivating it to transmit data uncompressed.

In light of these results, NAM employs a hybrid model, building on the strengths of both models. This hybrid model works as follows. For connections were the expected loss is zero (based on previous statistics), NAM employs NLM. If previous statistics reveal a high likelihood of loss then NAM employs LM.

3. Performance Results

We conducted numerous experiments to evaluate the effectiveness of NAM in both a controlled laboratory setting and an Internet deployment. The controlled laboratory setting was configured with a variety of network switches and processor speeds. We used NIST Net [4] to study NAM with a variety of packet loss and bandwidths. This experimental setup enabled us to study NAM with speeds up to 1 Gbps. Such bandwidths are expected to become common in the near future.

The Internet experiments were conducted using connections between USC and clients at different academic institutions as well as home clients connected using either DSL/Cable modems or 56 Kbps dial-up modems. These experiments offered a variety of network bandwidths and latencies. A connection might offer either a low ($\downarrow L$), moderate ($\leftrightarrow L$), or high ($\uparrow L$) latency. With a given latency, a connection may offer either a low ($\downarrow B$), moderate ($\leftrightarrow B$), or high ($\uparrow B$) bandwidth, see Table 2. This yields nine combinations. This paper describes our observations with five of these, which were selected to show that NAM is a general purpose technique that adapts to its target environment and application. NAM's overall behavior is determined by how well regression model of Section 2.1 and analytical models of Section 2.2 perform their estimations.

The five reported experiments are:

- Southern California, SC$_{\downarrow L,\leftrightarrow B}$: An Internet deployment with a Linux server at USC and a client at University of California, San Diego. This connection offers typical bandwidths ranging from 90 to 96 Mbps and latency of 3.5 ms (RTT = 7 ms). This is a low latency, moderate bandwidth experiment.

- US$_{\leftrightarrow L,\downarrow B}$: An Internet deployment with a Linux server at USC (west coast) and ISI at Washington DC (east coast). The ISI connection is a T1 with typical bandwidth of 1-1.2 Mbps and 45 ms latency (RTT = 90 ms). This experiment represents a moderate latency, low bandwidth connection.

- US$_{\leftrightarrow L,\leftrightarrow B}$: A coast-to-coast Internet deployment consisting of one machine at University of Massachusetts at Amherst, and another at USC in Los Angeles. The bandwidth between USC and UMass is approximately 87 to 96 Mbps with 45 ms latency (RTT = 90 ms). This experiment represents a moderate latency, moderate bandwidth connection.

- Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$: An Internet deployment with a Linux server at USC and a Sun OS 5.8 client at the University of Saarlandes in Germany. This Internet connection observes 90 to 97 Mbps bandwidths with 89 ms latency (RTT=178 ms). This represents a high latency, moderate bandwidth connection.

- 1-Gbps$_{\downarrow L,\uparrow B}$: An intranet configuration consisting of two machines connected using a Gigabit Ethernet switch. The bandwidth is limited by the processor speed and varies from 300 to 500 Mbps. Latency is low, 0.15 ms (RTT = 0.3 ms), because the machines are physically located in the same room (SAL 102) at the USC database laboratory. This represents a high bandwidth, low latency connection.

We analyzed NAM's behavior using the TPC-H benchmark [13], a decision support benchmark with documented queries and data sets. This benchmark can be configured

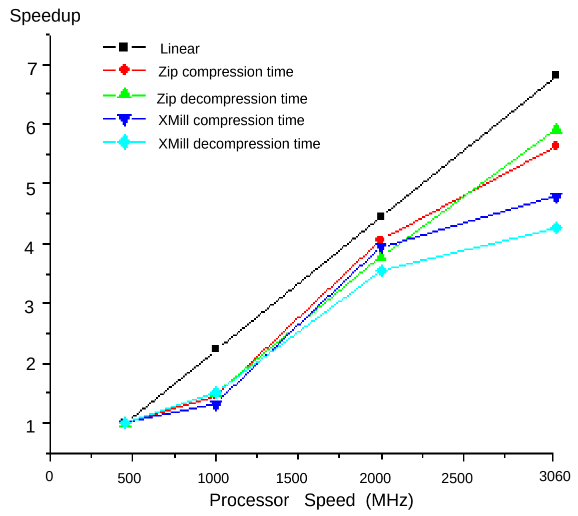


Figure 5. Speedup of compression and decompression times for TPC-H Query 13 as a function of processor speed.

with different database sizes. We analyzed two: 1 and 10 Gigabyte database sizes. 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. Since NAM's focus is on network transmission times, we focus on retrieval queries and ignore refresh commands from further consideration. We categorized the 22 retrieval queries into four categories: a) Small queries, denoted Q_S, with XML formatted result set sizes equal to or smaller than 1 Kilobyte, b) moderate queries, Q_M, with result set sizes greater than or equal to 1 Kilobyte and smaller than 1 Megabyte, c) large queries, Q_L, with result set sizes greater than or equal to 1 Megabyte and smaller than 10 Megabytes, and d) huge queries, Q_H, with results set sizes greater than 10 Megabytes and smaller than 50 Megabytes.

We analyzed the performance of Zip and XMill with these query classes as a function of different processor speeds: 450 MHz, 1 GHz, 2 GHz, and 3.06 GHz. Table 3 shows these numbers for four TPC-H queries (1 Gigabyte database size) with a 3.06 GHz PC. Each query is a member of different query class and shown for illustration purposes. (We refer the interested reader to [3, 8] for tables showing these number for all queries and different processor speeds.) Table 3 shows three important observations. First, the compression factor does not necessarily increase as a function of message size. Second, XMill yields more compact messages when compared with Zip for messages larger than 1 KB. Third, XMill is more time consuming than Zip.

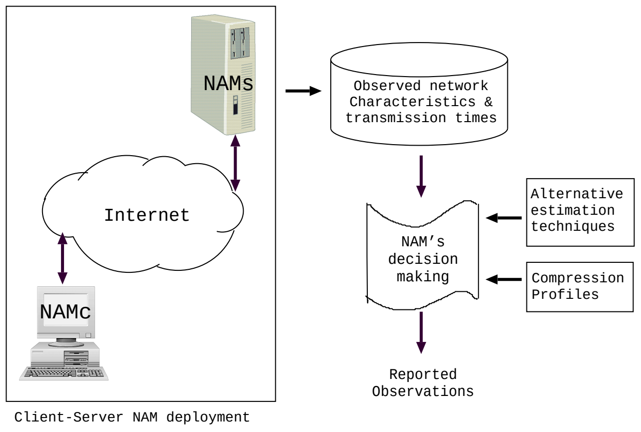

NAMs
Internet
NAMc
Client-Server NAM deployment
Observed network Characteristics & transmission times
NAM's decision making
Alternative estimation techniques
Compression Profiles
Reported Observations

Figure 6. Experimental framework.

It is also important to note that compression and decompression times do not scale linearly as a function of processor speeds. In Figure 5, we show the speedup observed with one TPC-H query (Q_{13}) as a function of different processor speeds. The break-down is applied to both compression and decompression times of Zip and XMill. Similar trends hold for other TPC-H queries. In particular, no query observes a linear speedup as a function of processor speed.

We used the controlled environment of Figure 6 to analyze the impact of alternative estimation techniques on NAM. The details of this environment are as follows. We stored the result of each TPC-H queries in a file and registered the compression and decompression time of each data set with alternative processor speeds. These times exclude the time to read the file into memory. These correspond to the "compression profiles" box of Figure 6. Next, we performed an experiment consisting of an Internet application configured with a NAM_C invoking a remote server with the identity of a query. The query id uniquely identifies the data set size that must be transmitted from the server to NAM_C. The server employs NAM to determine if the referenced data set should be transmitted compressed or uncompressed and logs this information. Next, it transmits the data set in uncompressed, Zip compressed and XMill compressed to NAM_C and registers the observed response time and network characteristics in a log file. An off-line program, "NAM's decision making" box of Figure 6, processes these logs and emulates different processor speeds. In addition, it computes the percentage improvement provided by NAM when compared with: a) uncompressed transmission always, corresponding to how Web Services are deployed today, termed Uncompressed, b) Zip transmission always, a simple improvement on today's status that would employ Zip always, c) XMill transmission always, an environment that employs XMill at all times.

Figure 7.a shows the percentage improvement with

		Zip Compression				XMill Compression			
Query	MSG size	MSG size	Comp Factor	Comp Time (ms)	Decomp Time (ms)	Msg size	Comp Factor	Comp Time (ms)	Decomp Time (ms)
Q_S (Q_{12})	929	303	3.1	0.68	0.07	358	2.6	1.78	0.70
Q_M (Q_{20})	46,219	6,766	6.8	2.8	0.85	5,962	7.75	9.54	3.04
Q_L (Q_3)	3,513,484	243,912	14.4	146.40	55.32	199,671	17.60	356.54	173.1
Q_H (Q_{10})	25,927,167	4,166,404	6.2	1,493.00	569.00	3,259,501	7.95	2,916.6	273.3

Table 3. Sample compression and decompression times using a 3.06 GHz processor. The query id in parentheses denotes the TPC-H query used as a representative of a category. The granularity of reported message size and times are in bytes and ms, respectively.

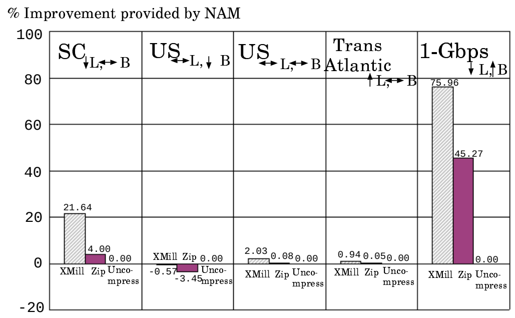
NAM when configured with a 3.06 GHz processor speed for five different deployments. For each, the y-axis shows the percentage improvement of NAM when compared with XMill compression always, Zip compression always, and uncompressed always. We observed a zero loss rate in these experiments. NAM employs an uncompressed transmission for Q_S always because the message is smaller than a TCP packet (a packet is 1480 bytes between two Linux machines and 1460 bytes between a Linux machine and a Sun OS 5.8 machine). Its model assumes TCP must transmit at least one packet worth of data and attributes zero benefits to using either Zip or XMill. The results shows the superiority of this decision with the low latency connections that offer moderate to high bandwidths (1-Gbps, SC). With a low bandwidth, moderate latency connection, US$_{\leftrightarrow L,\downarrow B}$, compression provides marginal benefits (1 to 3%) with a 3.06 GHz processor because the model's assumption is violated (transmitting fewer than one packet worth of data does provide savings in network transmission times). With a 2 GHz processor speed, this marginal benefit disappears. With both 1 GHz and 450 MHz processor speeds, use of either Zip or XMill is inferior to an uncompressed transmission.

Figure 7.b shows the percentage savings for Q_M and a 2 GHz processor speed. With the 1-Gbps$_{\downarrow L,\uparrow B}$ environment, NAM continues to transmit data uncompressed, providing substantial savings when compared with environments configured to use either Zip or XMill always. With Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$ and US$_{\leftrightarrow L,\leftrightarrow B}$, NAM employs XMill for 77% of queries, Zip for 14% of queries, and uncompressed transmission for the remaining 9% of queries. An environment that would employ XMill compression always outperforms NAM by approximately 19% for Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$. This drops to 2% with a 1 GHz processor. With a 450 MHz processor, NAM is superior to XMill always. Note that NAM provides substantial savings when compared with today's common practice to transmit uncompressed always.

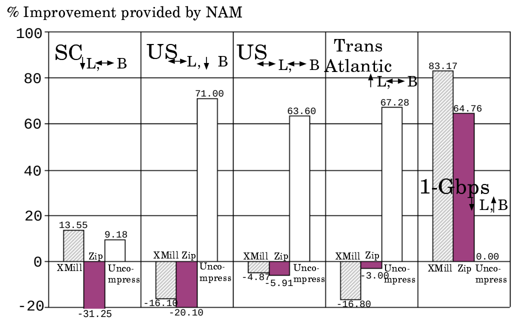
The results observed with both US$_{\leftrightarrow L,\downarrow B}$ and SC$_{\downarrow L,\leftrightarrow B}$ demonstrate the importance of estimating network characteristic and data compression times accurately. Consider each experiment in turn. With US$_{\leftrightarrow L,\downarrow B}$, NAM employs XMill for 68% of queries, Zip for 22% of queries, and uncompressed transmission for the remaining 10%. This is partly because cubic regression cannot estimate the compressed message size, compression and decompression times accurately with samples from both the 1 and 10 Gigabyte databases. If perfect estimates were provided, NAM would employ Zip and XMill 81% and 19% of the time respectively, providing savings when compared with XMill. The same would hold true if NAM was provided with samples from 1 Gigabyte database. The US$_{\leftrightarrow L,\downarrow B}$ results demonstrate the importance of estimating compression and decompression times accurately.

With SC$_{\downarrow L,\leftrightarrow B}$, NAM employs XMill for more queries than desired, resulting in inferior performance when compared with an environment that employs Zip always. This is partly due to inaccurate estimates provided by regression. However, even with perfect knowledge of compressed message size, compression and decompression times, an environment that employs Zip always will continue to outperform NAM by 7%. This is because NAM observes a loss from a prior transmission and uses this to estimate response times with different compression techniques. However, the observed transmission does not encounter the expected loss rate, causing the Zip compressed response times to appear superior.

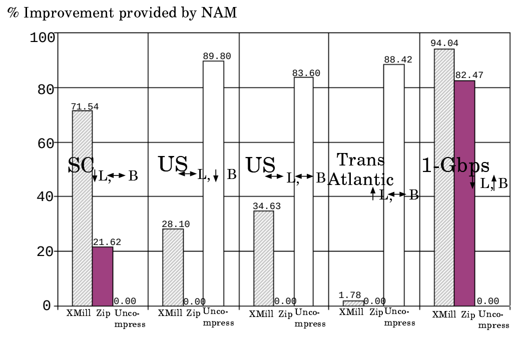
Figure 7.c shows the percentage savings for the Q_L and a 1 GHz processor speed. With low latency connections that offer moderate to high bandwidths (1-Gbps$_{\downarrow L,\uparrow B}$ and SC$_{\downarrow L,\leftrightarrow B}$) NAM transmits data uncompressed, producing significant savings. With Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$, $US_{\leftrightarrow L,\leftrightarrow B}$ and $US_{\leftrightarrow L,\downarrow B}$, NAM switches to Zip compression, providing savings when compared with an uncompressed transmission. With Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$, NAM's savings when compared with an environment that employs XMill always is marginal. This is due to a larger RTT between USC and University of Saarlandes that dominates the network response time. It is interesting to note that with SC$_{\downarrow L,\leftrightarrow B}$,



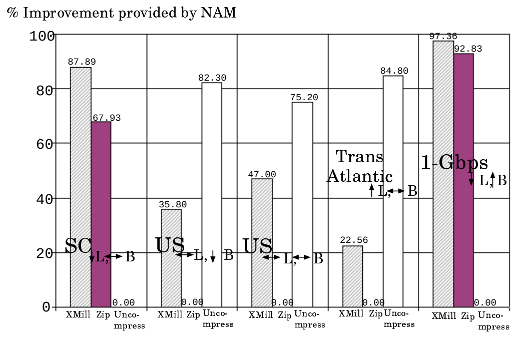
7.a Q_S with 3.06 GHz processors



7.b Q_M with 2 GHz processors



7.c Q_L with 1 GHz processors



7.d Q_H with 450 MHz processors

Figure 7. Percentage improvement with NAM for different processors.

NAM employs an uncompressed transmission for Q_L because of the small RTT (7 ms). With faster processor speeds (2 and 3.06 GHz), NAM switches to Zip compression.

Figure 7.d shows the percentage savings for Q_H and a 450 MHz processor speed. The 1-Gbps$_{\downarrow L,\uparrow B}$ and SC$_{\downarrow L,\leftrightarrow B}$ continue to transmit data uncompressed, providing significant savings when compared with those environments that employ either XMill or Zip always. Even with a 3.06 GHz processor, NAM employs uncompressed transmission with the 1-Gbps$_{\downarrow L,\uparrow B}$ connection. The only difference is a lower percentage savings when compared with XMill and Zip, i.e., NAM outperforms XMill and Zip by 82% and 60%, respectively. With Trans-Atlantic$_{\uparrow L,\leftrightarrow B}$, $US_{\leftrightarrow L,\leftrightarrow B}$ and $US_{\leftrightarrow L,\downarrow B}$, NAM employs Zip compression to provide savings. With faster processor speeds (1 GHz and above), NAM switches to XMill.

Results of Figure 7 show NAM adapts to enhance response time across different output XML data set sizes, processor speeds, and network conditions. It may not provide a superior response time for every single transmission, however, it significantly improves the average response time across many transmissions. Of course, given a specific processor speed and query output size, one may tailor the system to outperform NAM. Discussions of Figure 7.b for $US_{\leftrightarrow L,\downarrow B}$ is a demonstration of this claim. However, NAM is designed to be general purpose and adapt to the characteristics of its environment.

4. Conclusion and Future Research Directions

This paper presents NAM as a middleware to enhance response time of Web Services using compression techniques when appropriate. Our performance results demonstrate how NAM switches from using XMill compression to Zip and uncompressed transmission selectively to improve response time. We analyzed the performance of this middleware with TPC-H queries that produce data set sizes ranging from a few hundred bytes to tens of Megabytes. This was done in the context of different Internet and intranet settings.

NAM is designed to adapt based on the data characteristics of an application, available client and server processor speeds, and network characteristics. It exchanges profile information between a client (NAM_C) and a Web Service (NAM_S) in support of intelligent decisions. This distributes the overhead of NAM between the client and the Web Service in order to free the Web Service to scale to a large number of clients. It is important to note that NAM is not appropriate for multimedia content such as still images, audio and video clips, etc. This is because these data types are compressed using a lossy compression technique. A lossless compression technique, such as Zip, reduces the size of these files by only a few percentage (instead of several factors as with XML, see Table 3).

An immediate research direction is to extend NAM to maintain a history of its decisions and their quality. This would enable it to detect when its estimates are inaccurate. Moreover, it is possible to attribute this error to either the regression models or the observed network characteristics. Based on this, NAM may start to modify its models in order to enhance the quality of its decisions. A challenge here is the division of this activity between NAM_C and NAM_S. Another research direction is to extend NAM to streaming architectures. In its present form, NAM assumes a Web Service produces its output in its entirety prior to its transmission. While this assumption matches today's practices, we anticipate emergence of XML streaming architectures in support of integrated Web Services. These will incrementally produce and transmit their XML formatted output in a pipelined manner, overlapping the server's production of output with the client's consumption and processing of this output. We intend to explore extensions of NAM to these architectures in the near future.

5. Acknowledgments

We wish to thank the following for providing us with accounts at their institutions: Gerhard Weikum of the University of Saarlandes, David Moore of UCSD, Jim Kurose and Don Towsley of Amherst, and Colin Perkins of ISI-East. We also wish to thank Gurmeet Singh and Kuntal Joshier for their help with implementation of NAM and instrumentation of the Linux kernel.

References

[1] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System Support for Bandwidth Management and Content Adaptation in Internet Applications. Technical Report LCS-TR-808, MIT, May 2000. http://nms.lcs.mit.edu/projects/cm/.

[2] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1977.

[3] M. Cai, S. Ghandeharizadeh, R. Schmidt, and S. Song. A Compression of Alternative Encoding Mechanism for Web Services. In *In Proceedings of DEXA Conference*, August 2002.

[4] M. Carson. *NIST Net*. Internetworking Technologies Group, NIST, 1997. http://snad.ncsl.nist.gov/itg/nistnet/.

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

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

[7] S. Ghandeharizadeh, C. A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. L. Ambite, M. Cai, C. Chen, P. Pol, R. Schmidt, S. Song, S. Thakkar, and R. Zhou. Proteus: A System for Dynamically Composing and Intelligently Executing Web Services. In *First International Conference on Web Services (ICWS)*, Las Vegas, Nevada, June 2003.

[8] S. Ghandeharizadeh, C. Papadopoulos, M. Cai, and K. Chintalapudi. Performance of Networked XML-Driven Cooperative Applications. In *In Proceedings of Second International Workshop on Cooperative Internet Computing (CIC, held in conjunction with VLDB)*, August 2002.

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

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

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. Technical Report 2018, Network Working Group RFC, October 1996.

[12] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

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

[14] J. Postel. Transmission Control Protocol. Technical Report 793, Network Working Group RFC, September 1981.

[15] E. C. Shek, R. R. Muntz, and L. Fillion. The Design of the FALCON Framework for Application Level Communication Optimization. Technical Report CST-TR-960039, UCLA, November 1996.

[16] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. An Integrated Model for the Latency and Steady-State Throughput of TCP Connections. *Performance Evaluation*, 46(2-3):139–154, 2001.

[17] A. S. Szalay, T. Budavaria, T. Malika, J. Gray, and A. Thakar. Web Services for the Virtual Observatory. In *SPIE Astronomy Telescopes and Instruments*, Waikoloa, Hawaii, August 2002.

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