Performance Impact of Web Service Migration in Embedded Environments

Kevin J. Ma
Cisco Systems
Boxborough, MA 01719, USA
Email: kema@cisco.com

Radim Bartoš
Department of Computer Science
University of New Hampshire,
Durham, NH 03824, USA
Email: rbartos@cs.unh.edu

Abstract

The benefits provided by Web Service protocols are well recognized. Deployments to date, however, have concentrated on new applications, and existing Web-based applications. A host of legacy applications and protocols continue to exist in their native forms, outdated, yet entrenched due to large installed bases. This paper details our observations in integrating a Web Service infrastructure into the Simple Network Management Protocol (SNMP). SNMP has been in use for over a decade and a half, predominantly in network equipment embedded systems. Our Web Service-based approach allows us to enhance our existing application with XML/SOAP interoperability, SSL/TLS security, and the potential to migrate both application and protocol layers to encompass future extensions and Web browser accessibility. The difficulty with SNMP, and many other legacy networking protocols, is that much of the extensive installed base is hosted on limited capability, legacy hardware. While the benefits of our scheme are quite tangible, the performance impact of adding these features is not well known. We examine two approaches, an integrated solution using a light-weight HTTP/SOAP stack, as well as a standard Java Web server implementation. Our tests reveal unanticipated performance results through both the integrated and proxy methods. We discuss the impact of these anomalies on the viability of our approach and address the broad issue of migrating Web Services to legacy embedded architectures.

1 Introduction

Web Services are quickly gaining traction as the infrastructure of choice for new Web applications. The benefits of XML encoding flexibility, SOAP interoperability, and orchestrated BPEL transactions are enabling developers to implement robust new applications and extend the capabilities of existing Web-based infrastructures. However, the benefits of Web Services are applicable in other environments as well. There is a body of legacy networking protocols which are lacking in flexibility, accessibility, and security, each aspect of which can be addressed through the adoption of a Web Service-based paradigm. The following sections describe our findings in implementing a Web Service interface for the Simple Network Management Protocol (SNMP).

1.1 Web Services and Legacy Applications

The extensible and user definable nature of XML enables migration flexibility from legacy protocols to new, Web Service-based versions of those protocols and makes future migration easier. Most legacy protocols are rooted in proprietary data formats that are difficult to extend. Due to large installed bases for existing applications, the impact of changes must be minimized and backwards compatibility maintained. XML and SOAP may be defined to mimic existing standards and mitigate risk in upgrading and extending. It does not alleviate the initial implementation hurdle, but does provide a future migration path, along with the other benefits of Web Services. We must first, however, understand the impact of choosing to adopt an XML/SOAP representation. The processing overhead, messaging overhead, and potential integration barriers must be examined.

The HTTP binding for SOAP allows new Web Service implementations to take advantage of an underlying infrastructure, well known for its accessibility. Web browser-based interfaces enable ubiquitous access to applications and tools, however, it is not just the browser ubiquity that makes HTTP appealing. The implications of ubiquity are also interesting. HTTP is well deployed, well understood, and well supported. Many protocols have adopted Web interfaces. Consequently, HTTP access is a necessity in most networks and is normally granted access through firewalls and filters. Legacy network protocols have endured and proliferated by providing stable and available foundation. HTTP provides a similar stability and comparable availability. As risk mitigation is critical to legacy protocol migration, an established protocol like HTTP is desirable. The HTTP protocol binding also provides a host of security benefits. The HTTPS and the SSL/TLS protocols strengthen the Web Service position by including a respected suite of cryptographic algorithms and a security protocol with over a decade of refinement. While the Web Service security standards have yet to solidify or find broad-based deployment, the message level, transaction-based security of SSL/TLS provides a suitable migration option. SSL/TLS offer fully capable data privacy with an extensive infrastructure for testing and maintenance. However, processing overhead, differences in protocol interactions, and any potential down-time which may be incurred must be quantified before a transition can occur. Upgrades must be swift, seamless, and straightforward.

1.2 Simple Network Management Protocol

For our research, we chose the Simple Network Management Protocol (SNMP) and a candidate for migration to a Web Service-based scheme. SNMP has been in existence for over 15 years. Originally standardized in 1990, it was quickly adopted and became entrenched in the network administration landscape. The intent was for SNMP to be a short-term solution while a more robust framework, the Open System Interconnection (OSI) Network Management Model, was developed in parallel [1]. However, the merger of SNMP's simplicity with the rich features of the OSI model, was never realized. Time to market and ease of implementation played pivotal roles in the popularity and mass adoption of SNMP and the inevitable reluctance to migrate.

The SNMP PDU was originally specified to be transmitted plain text [1]. There is no privacy, no authentication, and no access control. Management Information Base (MIB) objects are available (read and write) to any and all users with network access to the management interface. The SNMP community string could be construed as a password of sorts, however, community strings are transmitted clear text and are generally well known. They provide little in the way of security. SNMP is also specified to use UDP as a default transport layer protocol, though the use of other protocols, like TCP, is not prohibited. [1]. There are no acknowledgments, no retransmissions, and no fault recovery.

Traditionally, network management security and reliability has primarily involved: keeping network equipment behind locked doors, using private networks for management traffic and dedicated modems for remote access, and maintaining on-site administrative staff. This type of brick and mortar approach provides near adequate network security and reliability, but lacks scalability. To employ these measures in a global, multi-site topology is an extremely inefficient and expensive paradigm.

New protocols and proprietary schemes [2] have the same barrier to entry as the OSI model: backward compatibility with legacy devices. Proxy schemes [3] and application overlays [4] usually involve adding elaborate front end interfaces to the existing infrastructure. These candidates provide a good compromise between new protocols and legacy devices and is a favorite of software vendors. Protocol extensions, e.g., SNMPv3 (standardized by the IETF in December 2003) [5], provide the most integrated approach. They attempt to address, rather than mask, the deficiencies of the underlying protocol to provide a migration path for new devices.

In the following sections we examine two approaches for augmenting SNMPv2, seeking balance between the power and promise of Web Services and the constraints of migrating a legacy protocol. We detail each approach and present our observations on their performance and their ability to address the security, reliability, scalability, and accessibility.

2 The Application

Our vision for a Web Service-based SNMP proposes a non-intrusive extension to the existing protocol. Working within the confines of the SNMP standard, we propose a scheme for extending and improving the existing SNMPv2 infrastructure through the use of Web Services at the transport level (using an XML encoded, SOAP message encapsulation, bound to HTTP, for SNMP PDU transport). We investigated two options for implementing this extension: one using standard Web server technologies and a Java tool set, the other using a light weight HTTP/SOAP stack. Both were integrated with an existing SNMP daemon and tool set. The goal was to achieve interoperability between the two approaches, as well as maintain interoperability with legacy systems. Each scheme has unique performance and feature characteristics, and both provide SNMP with the benefits of the Web Services, i.e.,:

- a human readable XML alternative to binary ASN.1,
- data privacy provided through SSL/TLS,
- standardization on SOAP for interoperability,
- user authentication provided through HTTP,
- data compression provided through HTTP.

By addressing the security issues of SNMPv2, we enable scalability beyond the physical security boundaries which exist today and enable secure, in-band accessibility. By offering a Web browser based interface, we extend accessibility further through the use of a ubiquitously deployed infrastructure. By adding all of this at a very low layer, we

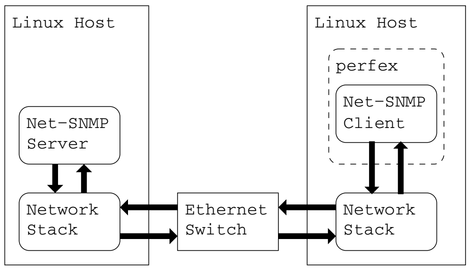

Figure 1. Net-SNMP Test Configuration.

maintain compatibility with the existing application, which is key for adoption and maintainability. Each of these features, though, has an associated cost. Quantification of performance impacts and increased resource utilization is necessary.

2.1 Net-SNMP and gSOAP

The Net-SNMP 5.0.8 open-source software package was chosen as the basis for the first phase of implementation. Net-SNMP is a widely used and respected software package. It is the default SNMP software for the major Linux distributions. Its pervasiveness makes it an ideal choice for these experiments.

Net-SNMP supports a number of different transport bindings, known as "Domains". It currently supports UDP, TCP, IPX, ATM-AAL5, UNIX Sockets, and a Callback domain. Supported domains are run time configurable. This structure is quite accommodating to adding support for new transport layer protocols, e.g., Web Service Domains.

For the purposes of this project, two additional domains were implemented to support SOAP over HTTP and SOAP over HTTP with SSL, "SOAP-HTTP" and "SOAP-HTTPS", respectively. The inherent abstraction of transport domain details from MIB processing, in Net-SNMP, helps promote clean integration of new domains.

The gSOAP 2.2.3 software package [11] was chosen as the SOAP stack implementation. It provided excellent documentation, platform independence, and a compiler tool which generates most of the necessary code for the low level SOAP infrastructure. The gSOAP package includes HTTP protocol support built in, as well as an OpenSSL extension for HTTPS support and a Zlib extension for HTTP compression support.

2.2 Apache Axis and Java Servlet

For the second phase of implementation, we set out to create a Web browser based Web Service interface for Net-SNMP. Given its wide following, Apache Web Server 2.0 and its accompanying toolset seemed a natural candidate for the server side. For the client side, Mozilla 1.3 was installed. Apache Axis 1.1 was chosen for SOAP protocol stack implementation. Axis 1.1 relied upon Apache Xerces 2.5 for XML parsing support, and was implemented in Java and built on top of the Apache Jakarta Tomcat 4.1.18 Java servlet architecture. The Sun Java JDK 1.4.1 provided our JVM support. A C++ version of Axis is currently available, however, it was not at the time of our implementation. The C++ is likely to have very different performance characteristics and is a topic for future research.

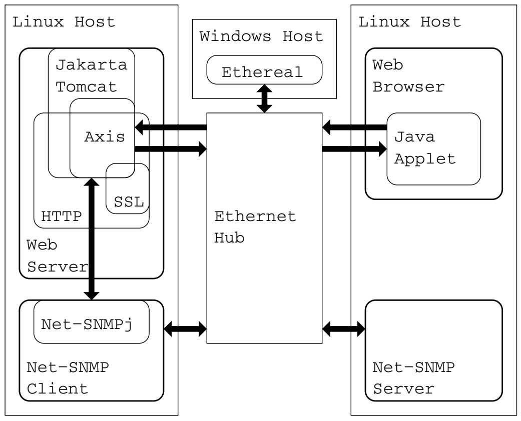

Figure 2. Apache Test Configuration.

While transparent proxy functionality for SNMP SOAP transactions was the ultimate goal, native Net-SNMP interactions were implemented through Net-SNMPj when we encountered difficulties related to the creation of custom serializers and deserializers in Apache Axis. We were unable to achieve full interoperability between the Web browser and integrated `snmpd` approaches. Net-SNMPj is an independently implemented Java API for Net-SNMP. The Net-SNMPj interface was incorporated into the Web Service Java servlet to provide local proxy access capabilities.

3 Evaluation Methodology

The goal of this project was to evaluate the viability of a Web Service-based approach for legacy applications. We specifically targeted low-bandwidth, low-latency, unreliable, insecure network protocols. These types of protocols are typically deployed in embedded systems as well as on server grade hardware. The intent was to answer two key questions:

- Would XML encoding have prohibitive performance impact due to the increased message sizes and additional message parsing?
- Would HTTPS have significant negative performance impact?

We believed that the former should be predictable and reasonable. We expected processing time to be reasonable with a custom, light-weight stack, and that link bandwidth utilization should be predictable and inconsequential. SNMP was designed over a decade ago. Available bandwidth has increased by orders of magnitude since that time. Increases in messaging overhead should be negligible relative to increases in bandwithd. For the latter, we inferred from anecdotal evidence that the implementation should be straightforward, with some performance impact. Given that assumption, we believed that the security of a Web Service-based approach warranted further investigation to quantify what the actual performance impacts might be.

For the first question, we looked at both a custom C-based implementation (gSOAP) and a standard Java-based implementation (Axis) and observed vastly different results. The two approaches are architecturally different and performance parity was not anticipated. However, fundamentally, the wide flexibility of Web Services, which enables both approaches to coexist and interoperate makes each a viable solution. From that standpoint, a comparison must be done to establish the relative benefits of each approach.

For the second question, OpenSSL seems to hold a monopoly on cryptographic implementations across all applications. SNMPv3, an alternative approach to adding security to SNMP, uses the OpenSSL library for cryptography, as do gSOAP and the Apache Web Server. However, while SSL/TLS cryptographic operations are abstracted from the Web Service implementation, the OpenSSL library cannot be overlooked in its feature role for Web Service transactions. We found the performance impact of OpenSSL to be non-insignificant.

3.1 Net-SNMP Evaluation

For the Net-SNMP implementation the following statistics were gathered:

- packets transfered per transaction,
- bytes transfered per transaction,
- wall clock latency per transaction,
- x86 CPU instructions executed,
- x86 CPU clock cycles expended.

The transaction was defined as a single `snmpget` request and response. The `sysContact` information was retrieved from a remote host, over a network with only our client and the server attached. Packet count, byte count, and latency data were gathered using the standard `tcpdump` utility included with Linux and parsed with simple scripts to extract the desired statistics. The `tcpdump` utility provides sniffer functionality at the network interface level of the host (in our case client) system. Consequently, the latency statistics include only the round trip network delay and the server side processing time. The client side processing is not included. While a full protocol analyzer offers more decoding capability, `tcpdump` provides a more easily automated solution for parsing the data we required. An in-line, or look-aside protocol analyzer ignores client side processing as well, so there was no significant loss of visibility from this decision. An Ethereal [7] sniffer was used to validate functionality, but not to gather data. The CPU statistics, on the other hand, do take into account the client side processing impacts. The `perfex` utility [9] was used to gather CPU statistics. The `perfex` utility is a freely available Linux tool for accessing the performance monitoring registers available on IA32 processors. It is based on the freely available `perfctr` Linux kernel patch [10]. One thousand samples were taken per run, to assess consistency between transactions and ten runs were performed to ensure reproducibility.

3.2 Apache Servlet Evaluation

To evaluate the relative performance of the Apache applet/servlet based approach, compared to Net-SNMP, a subset of the same network statistics were gathered, including: packets transfered per transaction, bytes transfered per transaction, and wall clock latency per transaction.

The network statistics were gathered using an external sniffer. A third host, running Ethereal sniffer software was added to the Ethernet hub, as seen in Fig. 2. Given the amount of network traffic we observed, per transaction, we required a more intelligent tool to analyze our samples. Using a full protocol analyzer, over shared media, created a much more cumbersome test setup, which inhibited our ability to automate data acquisition. The logistic issues limited our results gathering capability to much smaller sample sets, relative to the Net-SNMP results.

There was no suitable method for gathering comparable CPU statistics for the Java applet that interacted with our Java Web Service. The applet is not a self contained entity which we could easily track, as it was running within the context of Mozilla. The GUI poses additional complications, as we were unable to determine how cycles spent on rendering would be counted. A similar issue arises in the accounting for JVM interactions and other background tasks.

4 Results

Using the test scenarios and methodologies described in the preceding sections, we gathered data through each phase of the project and present our findings below. Detailed information about test setups and hardware, as well as additional graphs and data discussion may be found in technical report [6]. The tests involed two hosts running Red Hat Linux 8.0 and an additional Windows NT workstation for the Ethereal protocol analyser, in the Apache setup. A dedicated 100 Mbps NetGear Ethernet Hub was used for interconnect.

4.1 Net-SNMP Results

Table 1 contains a summary of the network statistics results from a single sample run of Net-SNMP. The table shows the packet count, byte count, and latency numbers for the four transport domains: UDP, TCP, HTTP (with SOAP), and HTTPS (with encrypted SOAP).

Packet counts were static across all runs as can be seen through the minimum and maximum values (and consequently the average values as well) being equal. The values were in-line with expectations: two packets for the simple RPC case of UDP, ten packets for TCP, adding the three-way handshake and teardown, ten packets for the HTTP case, which is just a degenerate case of TCP, and eighteen packets for HTTPS which includes TLS handshake and cleanup.
We did notice that packet counts for HTTPS did vary slightly. Analysis of the `tcpdump` data showed that this was due to differences in the piggybacking of acknowledgments by TCP. While not anticipated, this type of latitude, taken by the TCP stack, is not uncommon.

Byte counts between domains increased expectedly, for the UDP, TCP, and HTTP domains, due to handshake and teardown overhead. From sample to sample, within a given domain, byte counts varied only by the difference in encoded length of the randomly generated `request-id` field. The packet count and byte count results were predictable and reproducible, as expected.

Table 1. Net-SNMP Statistics (`tcpdump`).

Domain	*Statistic*	*min*	*max*	*avg*
UDP	packets	2	2	2
	bytes	157	159	159
	latency (ms)	0.255	10.649	0.505
TCP	packets	10	10	10
	bytes	637	639	639
	latency (ms)	0.900	14.952	1.548
HTTP	packets	10	10	10
	bytes	2502	2508	2507
	latency (ms)	2.770	27.576	3.132
HTTPS	packets	16	19	18
	bytes	4491	4635	4575
	latency (ms)	80.722	585.253	183.965

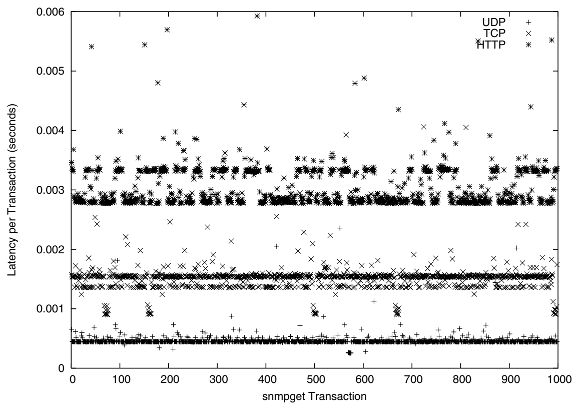

Figure 3. Net-SNMP Network Transaction Latency.

The non-HTTPS domain latency values contained a couple of high data points, but overall, the times were consistent across the samples. We can see this better in Fig. 3 which shows a plot of the latencies for the UDP, TCP, and HTTP transport domains. Aside from a few anomalous points, the general trend is flat over the sample sets. The three distinct bands from bottom to top, represent the UDP, TCP, and HTTP domains respectively. Each domain required slightly more processing time than the previous one, as expected. The TCP domain took four times as long as the UDP domain, strictly due to messaging overhead. The HTTP domain, in turn, took an additional four times as long, due to message processing overhead, above and beyond TCP.

However, when we look at the latency for the HTTPS domain we find that predictability and reproducibility are no longer achievable once we add in cryptographic operations. We had anticipated an increase in latency due to the addition of cryptographic operations, however the magnitude and variability of the results were quite unexpected.

Fig. 4 shows a field of seemingly unbounded data points with no best fit line. While there is a concentration of points near the bottom, the band is too wide to define a tight trend. Overlaid on the scatter plot of sample data is a plot of the data set sorted. It exhibits no flat spots to indicate a trend. While somewhat more consistent, it still spans too wide a range of times to suggest predictability. It looks more like an even distribution over a very wide range of times. We also should note the plot of the base HTTP domain along the zero line. The multiple orders of magnitude difference between the latency of HTTP and HTTPS becomes much more apparent, when illustrated by this graph. The HTTP domain is dwarfed in comparison to HTTPS.

Cryptographic operations are known to be computationally intensive. For this reason, many applications rely on hardware co-processors for accelerating these functions.

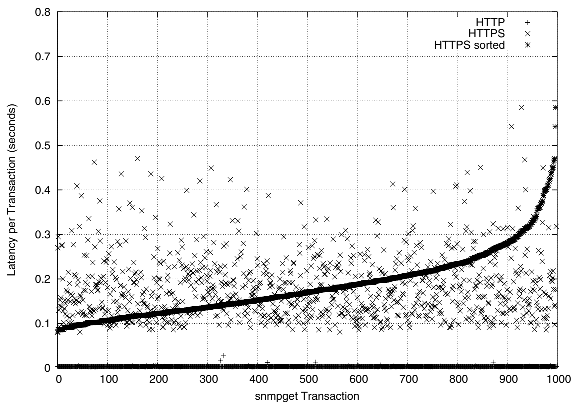

Figure 4. Net-SNMP HTTPS Transaction Latency.

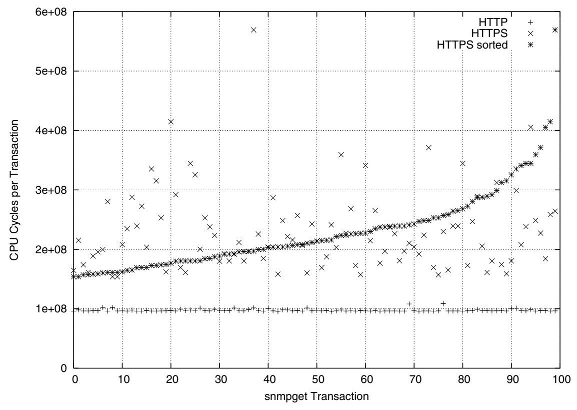

Figure 5. Net-SNMP HTTPS CPU Cycles.

Cryptographic support for our tests was provided through OpenSSL 0.9.6b. We expected some performance degradation due to the software-based cryptographic operations, however we did not anticipate the wide variance in processing time that we observed. Most SSL/TLS statistics are averaged over seconds and do not display the granularity which we attempted to achieve. We did verify that session caching was not being used and that random number quality was not an issue (i.e., that `/dev/urandom` was being used by OpenSSL). Root cause analysis of these anomalies is also a topic for future research.

Table 2 shows instruction count and cycle count data for the four Net-SNMP transport domains. Again, as with the network statistics, the data for the UDP, TCP, and HTTP domains were generally in line with expectations. The data in Table 2 was taken over a set of one hundred samples.

The CPU instruction counts are extremely consistent across the board, within domains, with the HTTP domain exhibiting less than a one percent increase due to XML, SOAP, and HTTP processing. It is expected that the instructions required to process the same request over and over again should exhibit reproducibility. The cycle count, however, is somewhat harder to quantify. Individual instructions have non-deterministic latencies associated with them due to main memory and cache accesses and I/O device response times. For the UDP, TCP, and HTTP domains, we do see fairly flat trends for CPU cycles, though with more variances than the CPU instructions counts. But again, the addition of cryptographic operations introduced large variability into the results.

Table 2. Net-SNMP CPU Statistics (`perfex`).

Domain	*CPU Instructions (millions)*		
	min	*max*	*avg*
UDP	81.545	81.545	81.545
TCP	81.536	81.536	81.536
HTTP	82.171	82.171	82.171
HTTPS	123.894	399.082	173.495
	CPU Cycles (millions)		
	min	*max*	*avg*
UDP	93.939	105.399	95.218
TCP	94.066	102.455	95.505
HTTP	95.802	108.663	97.427
HTTPS	153.737	569.306	229.282

Fig. 4 again depicts the magnitude and variability phenomena of the HTTPS domain. The network latency results are reinforced by these CPU utilization statistics. Looking at the plot of the CPU cycles, we again see a wide band of data with no intelligible trend. The scatter plot in Fig. 5 has again been overlaid with a plot of the data sorted. There are no noticeable flat spots in the overlay, and no discernible correlation to the HTTP domain plotted below it.

Finally, we also assessed the disk space requirements for our Web Service interface. The impact of the former was an increase in memory footprint of the Net-SNMP shared library of 436,294 bytes (from 1,258,362 bytes to 1,694,656 bytes). The addition of the gSOAP support and two new transport domains results in an increase in size of over a third. This does not include the space required for OpenSSL and/or Zlib.

4.2 Apache Results

Table 3 contains the network statistic results for the Web server proxy implementation. The "Preloaded" statistics were taken from a set of fifty sample runs with a web browser running our Java applet connecting to our Java servlet after the applet has been loaded and initialized.

The "Initial Load" statistics were taken from a set of ten samples in which the applet is not pre-initialized, to determine the initial load characteristics. Initial load is defined here as an applet load after restarting Mozilla (i.e., killing and restarting the Mozilla process), negating the effects of any information cached by Mozilla. Upon starting Mozilla, no default homepage load was configured so that the applet load would be the first and only operation performed by Mozilla. No library loads or setup should have tainted the initial load results.

The disparity in magnitude between all the results in the three data sets stands out immediately. The difference between the Net-SNMP HTTP domain statistics (Table 1) and the Web browser Proxy is two orders of magnitude. There is an additional two orders of magnitude difference between the proxy statistics and the initial load statistics. Though the number of data points is relatively small to be drawing conclusions about consistency, such a large difference in magnitude is unlikely to be reconcilable with more data points. The existence of latency and bandwidth usage this large is disturbing and can not be discounted. Performance impacts from a Java applet/servlet based approach was expected, however, the magnitude of the impact was quite unforeseen. We anticipate that the inclusion of SSL/TLS will only further compound this performance issue. We feel that the investigation of these anomalies, without the added performance degradation of HTTPS, is an important topic for further research.

Fig. 6 shows a graph of the latency data from the applet implementation. While there are quite a few stray cases of higher latency, a noticeable trend line along the bottom can clearly be seen. The other interesting piece of data to note is the plot of the Net-SNMP HTTP domain hovering just above or on the zero line. As was obvious from Table 3, it can be seen that the Apache based approach cannot begin to compete with the performance of the stripped down C implementation of the gSOAP stack.

Even more alarming is the latency cost of the initial applet load. Fig. 7 shows the inordinate amount of extra wall clock time that applet download takes. An applet is a self-contained entity which can be dynamically loaded and executed by a web browser. If an applet needs to execute functions in non-standard libraries (e.g., Apache Axis) it must download all the necessary libraries it requires. This can involve retrieval of hundreds of thousands of bytes of data. In time critical applications, this can have a significant effect on first response time. This download on the fly feature does, however, make the application significantly more portable than traditional applications. For applications that require accessibility from anywhere, the applet provides a handy option. It does not require any explicit installation and it can be run from any web enabled, Java capable machine. This flexibility and accessibility comes at a cost, however. What is interesting, and unclear, though, is that the increase in latency does not correspond linearly with the byte count increase. The number of bytes transferred during the initial load is little more than twice that of the standard proxy, however, the latency increases by more than ten times. We infer from this that some large amount of CPU processing overhead is also involved for each extra

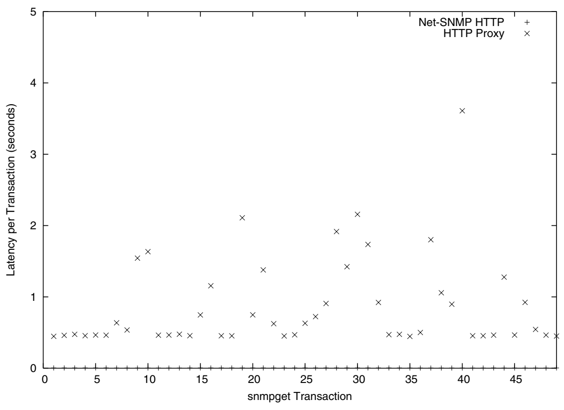

Figure 6. HTTP Proxy Transaction Latency.

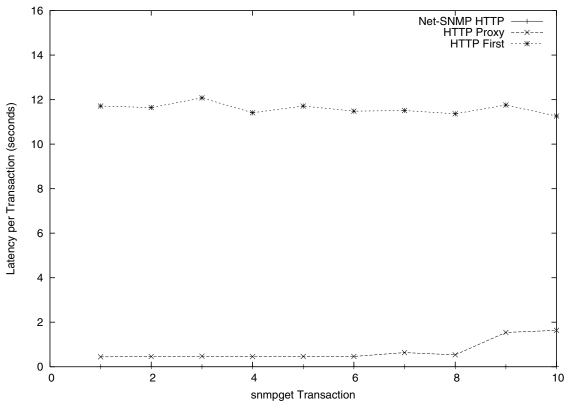

Figure 7. Initial HTTP Proxy Transaction Latency.

Table 3. HTTP Proxy Statistics (Ethereal).

Preloaded Applet			
Statistic	*min*	*max*	*avg*
packets	420	594	452
bytes	75410	86894	77545
latency (ms)	445.467	3610.068	872.117
Initial Applet Load			
Statistic	*min*	*max*	*avg*
packets	993	1085	1013
bytes	173254	179870	174691
latency (ms)	11264.101	12079.382	11594.271

byte of data being downloaded. Additionally, the penalty is not completely contained in the initial load. The unexpectedly large values for the non-initial load, proxy cases allude to this. An initial load penalty was expected, though not one of this magnitude, and it was assumed that the initial load penalty would occur only once and not carry over to each subsequent transaction. These phenomena have not yet been investigated in detail to determine whether the bottleneck exists on the server side, the client side, or both. This phenomena, too, is a topic for future research.

On the server side, the memory footprint of this implementation is well over one hundred megabytes (10MB for the Web Server, 50MB for Tomcat and Axis, 25MB for Ant, and 75MB for the Java SDK). Full installations of all the packages should not be required for our applet, however, if approached as a black box installation, the footprint ends up being around 160 MB, not including OpenSSL or Zlib.

5 Conclusions

This paper presents a look at the performance impacts of augmenting the standard Net-SNMP implementation of the legacy SNMPv2 protocol with a Web Service-based transport infrastructure. Our first phase implementation employed the gSOAP protocol stack, integrated into Net-SNMP. Our second phase implementation used industry standard infrastructure technologies including the Apache Web server, and Java applet and servlet technologies to provide a browser-based interface to Net-SNMP. The Web browser accessible scheme provides a high level of user accessibility, but is apparently accompanied by severe performance penalties. Custom implementations offer better performance, but without the interface ubiquity. In either case, both schemes offer access to the security features of HTTPS.

The addition of an HTTP/SOAP infrastructure alone, using the gSOAP package, showed noticeable increases in all the statistical areas, though the penalties were not crippling and were quite predictable. When adding SSL/TLS encryption for data privacy, however, order of magnitude increases in latency and processing, along with extreme jitter made quantification difficult from a security feature performance standpoint. It is unclear how HTTP compression might affect the performance characteristics of SSL/TLS. Compression will add processing overhead, however, the decreased amount of data being encrypted should have increasing returns in cryptographic throughput. Compression integration and analysis is an area of current research.

From the Apache implementation perspective, the processing penalties and the memory requirements dwarfed those of our Net-SNMP implementation by many orders of magnitude, even without SSL. This approach would definitely be infeasible for most legacy systems. Custom server installations and removal of the Java infrastructure might make this solution more manageable, however, platform limitations (e.g., proprietary operating systems, low-power CPUs, and limited memory, lack of hard disk, etc.,) must be taken into consideration in migratory environments.

We feel that the emerging Web Service technologies allow for powerful advancements beyond the simple framework of legacy applications. However, for these schemes to succeed, migratory concerns, such as those highlighted herein, must be addressed. More granular performance analysis is required with consideration given to embedded-scale applications. Performance issues should not be a barrier to entry to the tangible benefits of a Web Services based approach. A focused effort to understand the limiting factors in implementation performance will ultimately result in a broader scope for Web Service deployment and a more robust general infrastructure.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). `ftp://ftp.rfc-editor.org/in-notes/std/std15.txt`, May 1990.

[2] M. Choi, H. Choi, and J. Hong. XML-Based Configuration Management for IP Network Devices. *IEEE Communications Magazine*, 42(7), July 2004.

[3] L. Menten. Experiences in the Application of XML for Device Management. *IEEE Communications Magazine*, 42(7), July 2004.

[4] T. Klie and F. Strauss. Integrating SNMP Agents with XML-Based Management Systems. *IEEE Communications Magazine*, 42(7), July 2004.

[5] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. `ftp://ftp.rfc-editor.org/in-notes/std/std62.txt`, December 2002.

[6] K. Ma. Web Service-Based SNMP. Dept. of Computer Science, University of New Hampshire, December 2004. Tech. Rep. TR 04-02.

[7] The Ethereal website. `http://www.ethereal.com`.

[8] The Apache website. `http://www.apache.org`.

[9] A. Ertl. Linux Perfex. `http://www.complang.tuwien.ac.at/anton/linux-perfex`.

[10] M. Pettersson. The Linux/x86 Performance Monitoring Counters Driver. `http://www.csd.uu.se/~mikpe/linux/perfctr/`.

[11] R. A. van Engelen. The gSOAP Toolkit. `http://www.cs.fsu.edu/~engelen/soap.html`.