A^3: Application-Aware Acceleration for Wireless Data Networks*

Zhenyun Zhuang†, Tae-Young Chang†, Raghupathy Sivakumar†§, and Aravind Velayutham§

†Georgia Institute of Technology, Atlanta, GA 30332, USA
§ Asankya Networks, Inc., Atlanta, GA 30308, USA

zhenyun@cc.gatech.edu, {key4078,siva}@ece.gatech.edu, vel@asankya.com

ABSTRACT

A tremendous amount of research has been done toward improving transport layer performance over wireless data networks. The improved transport layer protocols are typically application-unaware. In this paper, we argue that the behavior of applications can and do dominate the actual performance experienced. More importantly, we show that for practical applications, application behavior all but completely negates any improvements achievable through better transport layer protocols. In this context, we motivate an application-aware, but application transparent, solution suite called A^3 (application-aware acceleration) that uses a set of design principles realized in an application specific fashion to overcome the typical behavioral problems of applications. We demonstrate the performance of A^3 through emulations using realistic application traffic traces.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication; C.2.2 [**Network Protocols**]: Applications; D.4.8 [**Performance**]: Simulation

General Terms

Algorithms, Design, Performance

Keywords

Wireless Networks, Application-Aware Acceleration

1. INTRODUCTION

A significant amount of research has been done toward the development of better transport layer protocols that can alleviate the problems the transmission control protocol (TCP) exhibits in wireless environments [9, 18, 11, 12]. Such protocols, and several more, have novel and unique design components that are indeed important for tackling the unique characteristics of wireless environments. However, in this paper we ask a somewhat orthogonal question in the very context the above protocols were designed for: *How does the application's behavior impact the performance deliverable to wireless users?*

Toward answering this question, we explore the impact of typical wireless characteristics on the performance experienced by the applications for very popularly used real-world applications including the file transfer protocol (FTP), the Common Internet File Sharing protocol (CIFS)[1], the Simple Mail Transfer Protocol (SMTP)[7], and the Hyper-Text Transfer Protocol (HTTP)[4]. Through our experiments, we arrive at an impactful result: Except for FTP which has a simple application layer behavior, *for all other applications considered, not only is the performance experienced when using vanilla TCP-NewReno much worse than for FTP, but the applications see negligible or no performance enhancements even when they are made to use the wireless-aware protocols.*

We delve deeper into the above observation and identify several common behavioral characteristics of the applications that fundamentally limit the performance achievable when operating over wireless data networks. Such characteristics stem from the design of the applications, which is typically tailored for operations in substantially higher quality local-area network environments (LANs). Hence, we pose the question: *if application behavior is a major cause for performance degradation as observed through the experiments, what can be done to improve the end-user application performance?*

In answering the above question, we present a new solution called *application-aware acceleration* (A^3), which is a middleware that offsets the typical behavioral problems of real-life applications through an effective set of principles and design elements. We present A^3 as a platform solution requiring entities at both ends of the end-to-end communication, but also describe a variation of A^3 called $A^3\bullet$, which is a point solution but is not as effective as A^3. One of the keystone aspects of the A^3 design is that it is *application-aware, but application transparent.*

The rest of the paper is organized as follows: Section 2 presents the motivation results for A^3. Section 3 presents the key design elements underlying the A^3 solution. Sec-

*This work was funded in part by NSF grants CNS-0519733, CNS-0519841, ECS-0428329 and CCR-0313005.

MobiCom'06, September 23–26, 2006, Los Angeles, California, USA.

tion 4 describes realization of A^3 for specific applications. Section 5 evaluates A^3. Section 6 discusses related works, and Section 7 concludes the paper.

2. MOTIVATION

The focus of this work is entirely on applications that require reliable and in-sequenced delivery. In other words, we consider only applications that are traditionally developed with the assumption of using the TCP transport layer.

2.1 Evaluation Model

We now briefly present the setting and methodology employed for the results presented in the rest of the section.

Applications: For the results presented in this section, we consider four different applications. Besides FTP, the applications are: (i) CIFS - The Common Internet File System is a platform independent network protocol used for sharing files, printers, and other communications between computers. While originally developed by Microsoft, CIFS is currently an open technology that is used for all Windows workgroup file sharing, NT printing, and the Linux Samba server[1]. (ii) SMTP - the simple mail transfer protocol is used for the exchange of e-mails either between mail servers, or between a client and its server. Most e-mail systems that use the Internet for communication use SMTP. (iii) HTTP - the hypertext transfer protocol is the underlying protocol used by the World Wide Web.

Traffic generator: We use the IxChariot to generate accurate application specific traffic patterns. IxChariot[13] is a commercial tool for emulating most real-world applications. It is comprised of the IxChariot console (for control), performance end-points (for traffic generation and reception), and IxProfile (for characterizing performance).

Testbed: We use a combination of a real test-bed and emulation to construct the test-bed for the results presented in the section. Since IxChariot is a a software tool that generates actual application traffic, it is hosted on the sender and the receiving machines shown in Figure 12. The path from the sender to the receiver goes through a node running the ns2 network simulator in emulation mode. The network emulator is configured to represent desired topologies including the different types of wireless technologies. More information on the test-bed is presented in Section 5.

Transport protocols: Since we consider wireless LANs (WLAN), wireless WANs (WWAN), and wireless satellite area networks (SAT), we use transport layer protocols proposed in related literature for each of these environments. Specifically, we use TCP-ELN (NewReno with explicit loss notification)[9], WTCP (Wide-area Wireless TCP)[18], and STP (Satellite transport protocol)[11] as enhanced transport protocols for WLANs, WWANs, and SATs respectively.

Parameters: We use average RTT values of 5 ms, 200 ms, and 1000 ms, average loss rates of 1 %, 8 %, and 3 %, and average bandwidths of 5 Mbps, 0.1 Mbps, and 1 Mbps for WLANs, WWANs, and SATs respectively. We use application perceived throughput as the key metric of interest. Wach data point is taken as an average of 10 different experimental runs.

[1]Samba uses SMB on which CIFS is based.

2.2 Quantitative Analysis

Figure 1(a) presents the performance results for FTP under varying loss conditions in WLANs, WWANs, and SAT environments. The tailored protocols uniformly show considerable performance improvements. The results illustrate that the design of the enhancement protocols TCP-ELN, WTCP, and STP, is sufficient enough to deliver considerable improvements in performance for wireless data networks, *when using FTP as the application.* In the rest of the section, we discuss the impact of using such protocols for other applications such as CIFS, SMTP, and HTTP.

Figures 1(b)-(d) show the performance experienced by CIFS, SMTP, and HTTP respectively under varying loss conditions for the different wireless environments. It can be observed that *the performance improvements demonstrated by the enhancement protocols for FTP do not carry over to these three applications.* It also can be observed that the maximum performance improvement delivered by the enhancement protocols is less than 5 % across all scenarios.

While the trend evident from the results discussed above is that the enhanced wireless transport protocols do not provide any performance improvements for three very popularly used applications, we argue in the rest of the section that *this is not due to any fundamental limitations of the transport protocols themselves, but due to the specifics of the behavior of the three applications under consideration.*

2.3 Impact of Application Behavior

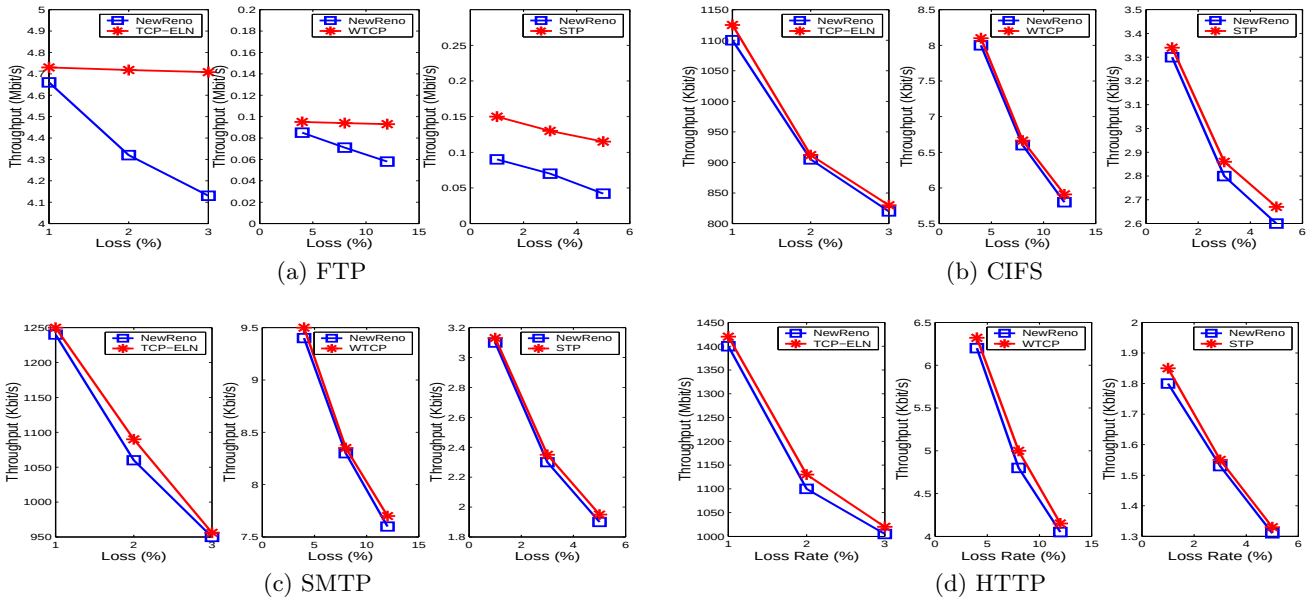
We now explain the lack of performance improvements when using enhanced wireless transport protocols with applications such as CIFS, SMTP, and HTTP. We use the conceptual application traffic pattern for the three applications in Figure 2 for most of our reasonings[1, 7, 4].

2.3.1 Thin session control messages

All three applications, as can be observed in Figures 2(a)-(c), use *thin session control message exchanges* before the actual data transfer occurs, and *thin request messages* during the actual data transfer phase as well. We use the term "thin" to refer to the fact that such messages are almost always contained in a single packet of MSS (maximum segment size).

The observation above have two key consequences: (i) When a loss occurs to a thin message, an entire round-trip is taken to recover from such a loss. When the round-trip time is large like in WWANs and SATs, this can result in considerably inflating the overall transaction time for the applications. Note that a loss during the data phase will not have such an adverse impact, as the recovery of that loss can be multiplexed with other new data transmissions, whereas for thin message losses, no other traffic can be sent anyway. (ii) Most protocols, including TCP, rely on the arrival of out-of-order packets to infer packet losses and hence trigger loss recovery. In the case of thin messages, since there are no packets following the lost message, the only means for loss detection is the expiry of the retransmission timer. Retransmission timers typically have coarse minimum values to keep overheads low. TCP, for example, uses a minimum Retransmission Time Out (RTO) value of *one second*[2].

[2]While newer Linux releases have lower minimum RTO values, they still are in the order of several hundred ms.



(a) FTP (b) CIFS (c) SMTP (d) HTTP

Figure 1: Impact of Wireless Environment Characteristics

2.3.2 *Batched data fetches*

Another characteristic of the applications, especially CIFS and HTTP, is that although the total amount of data to be fetched can be large, *the data transfer is performed in batches, with each batch including a "request-response" exchange.* CIFS uses its *request-data-block* message to send the batched requests, with each request typically requesting only 16 KB - 32 KB of data.

Such a batched fetching of data has two implications to performance: (i) When the size of the requested data is smaller than the Bandwidth Delay Product (BDP), there is a gross underutilization of the available resources. Hence, when the SAT network has a BDP of 128 KB, and CIFS uses a 16 KB request size, the utilization is 12.5 %. (ii) Independent of the size of each requested data batch, one *rtt* is spent in sending the next request once the current requested data arrives. When the *rtt* of the path is large like in WWANs and SATs, this can inflate the overall transaction time, and hence lower throughput performance.

2.3.3 *Flow control bottlenecked operations*

Flow control is an important function in communication that helps in preventing the source from overwhelming the receiver. In a mobile/wireless setting, flow control can kick in and prove to be the bottleneck for the connection progress due to two reasons: (i) If the application on the mobile device reads slowly or is temporarily halted for some other reason, the receiver buffer fills up and the source is eventually frozen till the buffer empties. (ii) When there are losses in the network, and the receiver buffer size is of the same order as the BDP (which is typically true), flow control can prevent new data transmissions even when techniques such as fast recovery is used due to unavailability of buffer space at the receiver. The dominant effect of flow control is however undesirable in wireless environments because of its resultant low throughput performance.

2.3.4 *Other reasons*

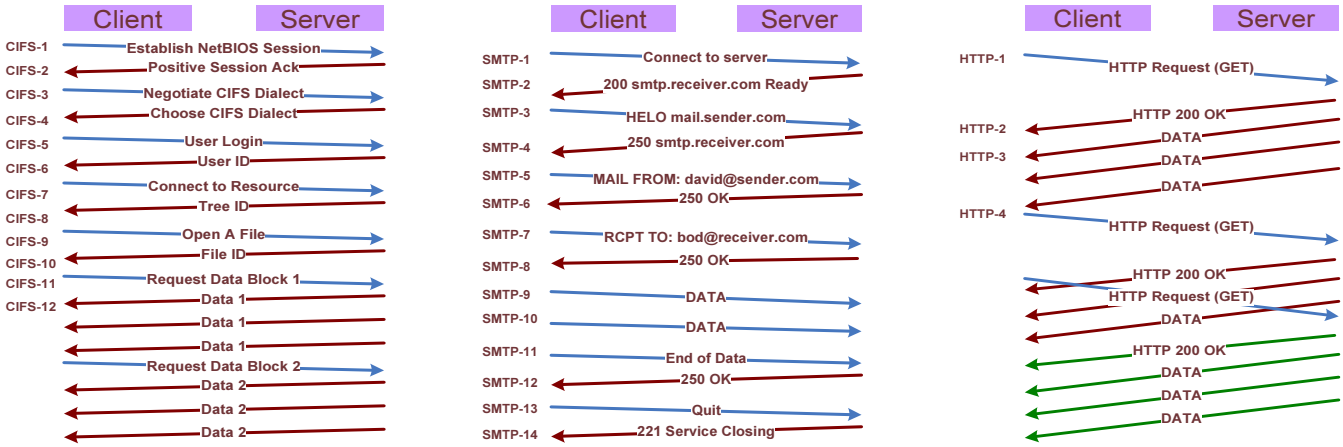
While the above discussed reasons are behavioral "acts of commission" by the applications that result in lowered performance, we now discuss two more reasons that can be seen as behavioral "acts of omission". These are techniques that the applications could have used to address conditions in a wireless environment, but do not.

Non-prioritization of data: For all three applications considered, no explicit prioritization of data to be fetched is performed, and hence all the data to be fetched are given equal importance. However, for certain applications prioritizing data in a meaningful fashion can have a profound impact on the performance experienced by the end-system or user. For example, consider the case of HTTP used for browsing on a small-screen PDA. When a webpage URL request is issued, HTTP fetches all the data for the webpage with equal importance. However, the data corresponding to the *visible* portion of the webpage on the PDA's screen is obviously of more importance and will have a higher impact on the perceived performance by the end-user. Not leveraging such means of prioritizing data hence results in HTTP suffering performance as defined by the original data size and the low bandwidths of the wireless environment.

Non-use of data reduction techniques: Finally, another issue is applications not using knowledge specific to their content or behavior to employ effective data reduction techniques. For example, considering the SMTP application, "email vocabulary" of users has evolved over the last couple of decades to be very independent of traditional "writing vocabulary" and "verbal vocabulary" of the users. Hence, it is an interesting question as to whether SMTP can use email vocabulary based techniques to reduce the actual content transferred between SMTP servers, or a SMTP server and a client. Not leveraging such aspects prove to be of more significance in wireless environments where the baseline performance is poor to start with.

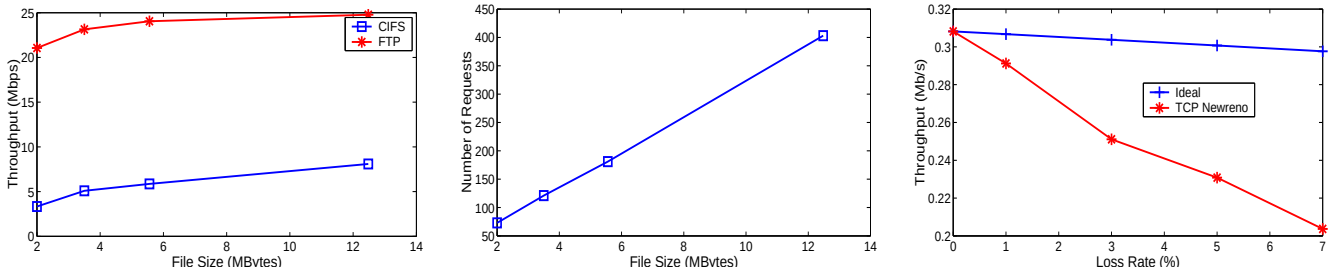
3. DESIGN

Since we have outlined several behavioral problems with



(a) CIFS (b) SMTP (c) HTTP

Figure 2: Application Traffic Patterns



(a) Throughput (b) Number of Requests (c) SMTP

Figure 3: Motivation for TP and RAR

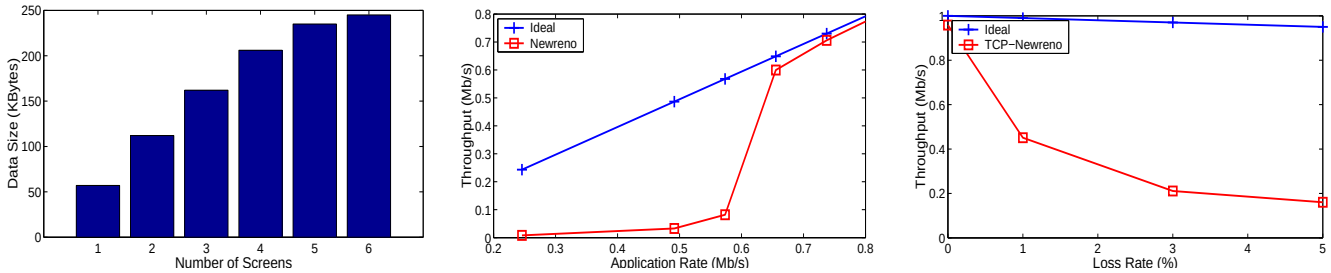
applications in Section 2, an obvious question to ask is: "*Why not change the applications to address these problems?*" We believe that is indeed one possible solution. Hence, we structure the presentation of the A^3 solution into two distinct components: (i) the key design elements or principles that underlie A^3; and (ii) the actual realization of the design elements for specific applications in the form of an *optimization middleware that is application-aware, but application transparent.* The design elements generically present strategies to improve application behavior and can be used by application developers to improve performance by incorporating changes to the applications directly. In the rest of this section, we outline the design of five principles in the A^3 solution.

3.1 Transaction Prediction

Transaction prediction (TP) is an approach to *deterministically predict* future application data requests to the server, and issue them ahead of time. Note that this is different from techniques such as "prefetching" where content is heuristically fetched to speed up later access, but is not guaranteed to be used. In TP, A^3 is fully aware of application semantics, and knows exactly what data to fetch and that the data will be used. TP will aid in conditions where the BDP is larger than the default application batch fetch size, and where the RTT is very large. Under both cases, the overall throughput will improve when TP is used. Figure 3(a) shows the throughput performance of CIFS when fetching files of varying sizes. It can be seen that the performance is substantially lower than that of FTP, and this is due to the batched fetching mechanism described in Section 2. Figure 3(b) shows the number of transactions it takes CIFS to actually fetch a single file, and it can be observed that the number of transactions increases linearly with file size. Under such conditions, TP will"parallelize" the transactions and hence improve throughput performance. Good examples of applications that will benefit from using TP include CIFS and HTTP for reasons outlined in Section 2.

3.2 Redundant and Aggressive Retransmissions

Redundant and aggressive retransmissions (RAR) is an approach to protect thin session control and data request messages better from losses. The technique involves recognizing thin application messages, and using a combination of packet level redundancy, and aggressive retransmissions to protect such messages. RAR will help address both issues with thin messages identified in Section 2. The redundant transmissions reduce probability of message losses, and the aggressive retransmissions that operate on tight *RTT* granularity timeouts reduce the loss recovery time. The key challenges in RAR is to recognize thin messages in an application-aware fashion. *Note that only thin messages require RAR because of reasons outlined in Section 2.* Regular data messages should not be subjected to RAR both because their loss recovery can be masked in the overall transaction time by performing the recovery simultaneously with other data packet transmissions, and because the overheads of performing RAR will become untenable when applied to large volume messages such as the data. Figure 3(c) shows the throughput performance of SMTP under lossy conditions.



(a) CDF of Transfer Size per Screen (b) Application Rate (c) Loss Increase

Figure 4: Motivation for PF and IB

The dramatic effect of a 35 % drop in throughput performance for a loss-rate increase from 0 % to 3 % is much higher than the 15 % drop in performance in the *FTP* performance for the same corresponding loss-rate increase shown in Section 2. Typical applications that can benefit from RAR include CIFS, SMTP, and HTTP.

3.3 Prioritized Fetching

Prioritized fetching (PF) is an approach to prioritize subsets of data to be fetched as being more important than others, and to fetch the higher priority data *faster* than the lower priority data. A simple approach to achieve the dual rate fetching is to use default TCP-like congestion control for the high priority data, but use congestion control like in TCP-LP[14] for low priority data. An important consideration in PF is to devise a strategy to prioritize data intelligently, and on the fly. Figure 4(a) shows the average *transfer size per screen* for the top fifty accessed webpages on the world-wide web[2]. It can be seen that nearly 80 % of the data (belonging to screens 2 and higher) are not directly impacting response time experienced by the user, and hence can be de-prioritized in relation to the data pertaining to the first screen. Note that the results are for a 1024x768 resolution laptop screen, and will in fact be better for smaller screen devices such as PDAs. Good examples of applications that can benefit from PF include HTTP and SMTP.

3.4 Infinite Buffering

Infinite buffering (IB) is an approach that prevents flow control from throttling the progression of a network connection terminating at the mobile wireless device. IB prevents flow control from impacting performance by providing the sender the impression of an *infinite buffer* at the receiver. Secondary storage is used to realize such an infinite buffer, with the main *rationale being that reading from the secondary storage will be faster than fetching it from the sender over the wireless network when there is space created in the actual connection buffer at a later point.* With typical hard-disk data transfer rates today being at around 250 Mbps[5], the abovementioned rationale is well justified for wireless environments. Note that the trigger for using IB can be both due to application reading slowly or temporarily not reading form the connection buffer, or due to losses on the wireless path. Figures 4(b)-(c) show the throughput performance of SMTP under both conditions. It can be observed that for both scenarios, the impact of flow control drastically lowers performance compared to what is achievable. Due to lack of space, in the rest of the paper we focus on IB specifically in the context of the more traditional trigger for flow control - application reading bottleneck. Typical applications that can benefit from IB include CIFS, SMTP, and HTTP - essentially, any application that transfers multiple BDPs worth of data at a time.

3.5 Application-aware Encoding

Unique Words	Total Words	Char. per Word	Bits per Email for Binary Coding	Bits per Email for Simple AE
1362	6383	6.22	3176	664.6

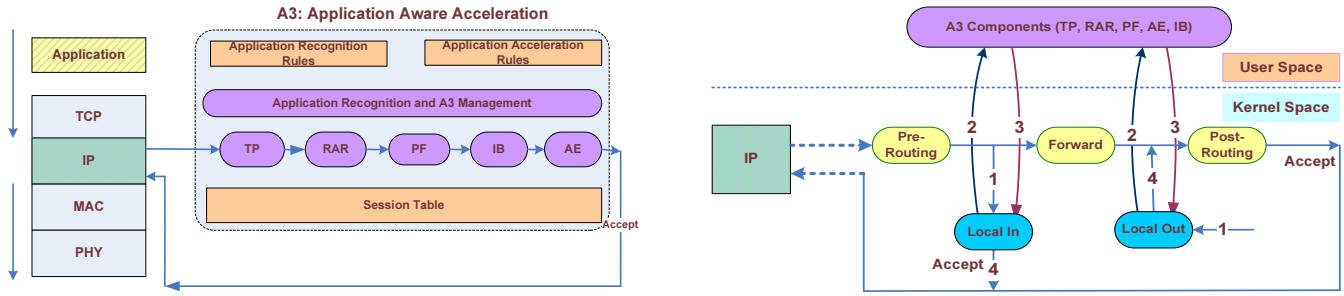
Table 1: Averaged Statistics of 10 Email Folders

Application-aware encoding (AE) is an approach that uses application specific information to better encode or compress data during communication. Traditional compression tools such as *zip* operate on a given content in isolation without any context for the application corresponding to the content. AE, on the other hand, explicitly uses this contextual information to achieve better performance. *Note that AE is not a better compression algorithm. However, it is a better way of identifying data-sets that need to be operated on by a given compression algorithm.* Table 1 shows the average e-mail vocabulary characteristics of ten different graduate students based on 100 emails sent by each person during two weeks. It is interesting to see the following characteristics in the results: (i) the e-mail vocabulary size across the ten people is relatively small - a few thousand words; and (ii) even a simple encoding involving this knowledge will result in every word being encoded with only 10 - 12 bits, which is substantially lower than using 40 - 48 bits required using standard binary encoding. In Section 5, we show that such vocabulary based encoding can considerably outperform other standard compression tools such as *zip* as well. Moreover, further benefits can be attained if more sophisticated compression schemes such as Huffman encoding is employed instead of simple binary encoding. Typical applications that can benefit from using AE include SMTP and HTTP.

4. SOLUTION

4.1 Deployment Model and Architecture

The A^3 deployment model is shown in Figure 5. Since A^3 is a platform solution, it requires two entities at either end of the communication session that are A^3 aware. At the mobile device, A^3 is a software module that is installed in user-space. At the server side, while A^3 can be deployed as a software module on all servers, a more elegant solution would be to deploy a packet processing network appliance that processes all content flowing from the servers to the wide-area network. We assume the latter model for our



(a) Deployment with Netfilter

(b) Software Architecture

Figure 6: A^3 Architecture

discussions. However, note that A^3 can be deployed in either fashion as it is purely a software solution.

This deployment model will help in any communication between a server behind the A^3 server, and the mobile device running the A^3 module. However, if the mobile device communicates with a non A^3 enabled server, one of two options exists: (i) As we discuss later in the paper, A^3 can be used as a point-solution with lesser effectiveness; or (ii) the A^3 server is brought closer to the mobile device, perhaps within the wireless network provider's access network. In the rest of the paper, we don't delve into the latter option. However, we do revisit the point-solution mode of operation of A^3.

We present an A^3 implementation that resides in user-space, and uses the NetFilter utility in Linux for the capturing of traffic outgoing and incoming at the mobile device. NetFilter is a Linux specific packet capture tool that has hooks at multiple points in the Linux kernel. The A^3 hooks are registered at the Local-In and Local-Out stages of the chain of hooks in NetFilter. While our discussions are Linux centric, our discussions can be mapped on the Windows operating system through the use of the Windows Packet Filtering interface, or wrappers such as PktFilter that are built around the interface. Figure 6(a) shows the A^3 deployment on the mobile device using NetFilter.

The A^3 software architecture is shown in Figure 6(b). Since the design elements in A^3 are to a large extent independent of each other, a simple chaining of the elements in an appropriate fashion results in an integrated A^3 architecture. The specific order in which the elements are chained in the A^3 realization is TP, RAR, PF, IB, and AE. While RAR protects the initial session control exchanges and the data requests, it operates on traffic after TP, given that TP can generate new requests for data. PF manipulates the priority with which different requests are served, and IB ensures that data responses are not throttled by flow control. Finally, AE compresses any data outgoing, and decompresses any data incoming.

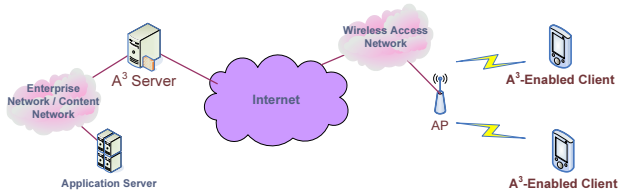


Figure 5: Deployment Model

4.2 Application Overviews

Since we describe the actual operations of the mechanisms in A^3 in the context of one of the three applications, we now briefly comment on the specific message types involved in typical transactions by those applications. We then refer to the specific message types when describing the operations of A^3 subsequently.

Due to lack of space, instead of presenting all message types again, we refer readers back to Figure 2 to observe the message exchanges for the three applications. The labels such as CIFS-x refer to particular message types in CIFS and will be referred to in the A^3 realization descriptions that follow.

CIFS, also sometimes known as Server Message Block (SMB), is a platform independent protocol for file sharing. The typical message exchanges in a CIFS session are as shown in Figure 2(a). Overall, TP manipulates the CIFS-11 message, RAR operates on CIFS-1 through CIFS-11, and IB aids in CIFS-12.

SMTP[7] is Internet's standard host-to-host mail transport protocol and traditionally operates over TCP. The typical message exchanges in an SMTP session are shown in Figure 2(b). Overall, RAR operates on SMTP-1 through SMTP-8, and SMTP-12 through SMTP-14, IB and AE operates on SMTP-9 and SMTP-10.

The HTTP message standard exchanges are relatively simple, and typically consist of the messages shown in Figure 2(c). Typical HTTP sessions consist of multiple objects, including the original HTML file, and hence appear as a sequence of overlapping exchanges of the above format. Overall, RAR operates on HTTP-1, and PF and IB operate on HTTP-3.

4.3 A^3 Implementation

In the rest of the section, we take one design element at a time, and walk through the algorithmic details of the element with respect to a single application. Note that A^3 is an application-aware solution, and hence its operations will be application specific. Since we describe each element in isolation, we assume that the element resides between the application and the network. In an actual usage of A^3, the elements will have to be chained as discussed earlier.

4.3.1 Transaction Prediction

Figure 7 shows the flow chart for the implementation of TP for CIFS. When A^3 receives a message from the application, it checks to see if the message is CIFS-9, and records state for the file transfer in its TP-File-States data structure.

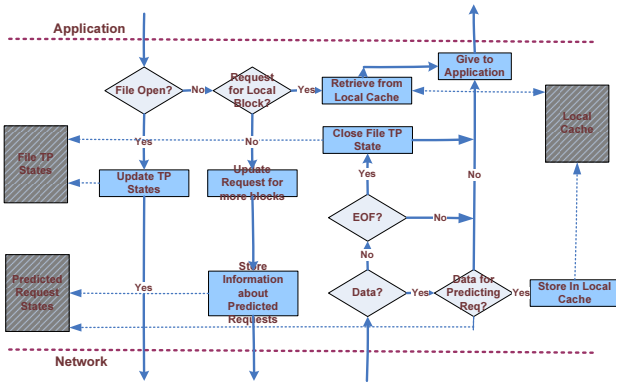


Figure 7: Transaction Prediction

It then passes through the message. If the incoming message was a request, TP checks to see if the request is for a locally cached block, or for a new block. If the latter, it updates the request for more blocks, stores information about the predicted requests generated in the Predicted-Request-States data structure, and forwards the requests.

In the reverse direction, when data comes in from the network, TP checks to see if the data is for a predicted request. If yes, it caches the data in secondary storage and updates its state information, and forwards the data to the application otherwise.

The number of additional blocks to request is an interesting design decision. For CIFS, A^3 uses a TP request for the entire file size, since the overall performance is not affected in any way given the CIFS server semantics that allows for multiple simultaneous requests. The file size information can be retrieved from the CIFS-10 message. If the incoming message is for an earlier retrieved block, TP retrieves the block from secondary storage, and provides it to the application.

While CIFS servers accept multiple data requests from the same client simultaneously, it is possible that for some applications, the server might not be willing to accept multiple data requests simultaneously. In such an event, the A^3 server will let only one of the client requests go through to the server at any point in time, and will send the other requests one at a time once the previous requests are served.

4.3.2 *Redundant and Aggressive Retransmissions*

Figure 8 shows the flow chart for the implementation of

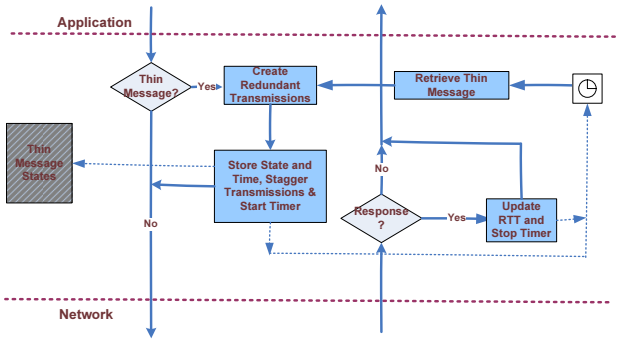


Figure 8: Redundant and Aggressive Retransmissions

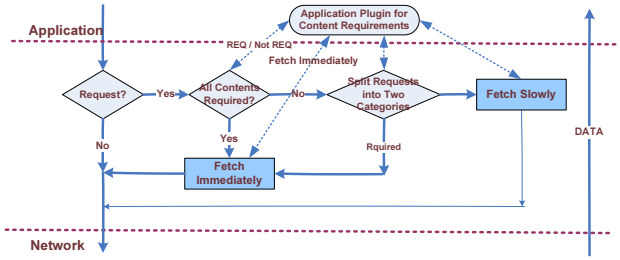


Figure 9: Prioritized Fetching

RAR for CIFS. When A^3 receives a message from the application, it checks to see if it is a thin message. The way A^3 performs the check is to see if the message is one of the messages between CIFS-1 and CIFS-11. All such messages are interpreted as thin messages.

If the incoming message is not a thin message, it is let through as-is. Otherwise, redundant copies of the message are created, information about current time noted, retransmission alarm started, and the copies sent out in a staggered fashion. When a response arrives, the timestamp for the corresponding request is checked, and RTT estimate updated. The message is then passed on to the application.

If the alarm expires for a particular thin message, the message is again subjected to the redundant transmissions. Successful arrivals of redundant copies of the same message are filtered at the A^3 server.

The key issues of interest in the RAR implementation are: (i) How many redundant transmissions are performed? Since packet loss rates in wireless data networks rarely exceed 10 %, even a redundancy factor of *two* (two additional copies created) reduces the effective loss-rate to 0.1 %. Hence, A^3 uses a redundancy factor of two. (ii) How should the redundant messages be staggered? The answer to this question lies in the specific channel characteristics experienced by the mobile device. However, at the same time, the staggered delay should not exceed the round-trip time of the connection, as otherwise the mechanism would lose its significance. Hence, A^3 uses a staggering delay of $\frac{RTT}{10}$ between any two copies of the same message. This ensures that within 20 % of the RTT duration, all messages are sent out at the mobile device. (iii) How is the aggressive timeout value determined? Note that while the aggressive timeout mechanism will help under conditions when all copies of a message are lost, the total message overhead by such aggressive loss recovery is negligible when compared to the overall size of data transferred by the application. Hence, A^3 uses a timeout value of the $RTT_{avg} + \alpha$, where α is a small guard constant, and RTT_{avg} is the average RTT observed so far. This ensures that the timeout values are tight, and at the same time the mechanism adapts to changes in network characteristics.

4.3.3 *Prioritized Fetching*

Figure 9 shows the flow chart for the implementation of PF in the context of HTTP. Once again, the key goal in PF for HTTP is to retrieve HTTP objects that are required for the display of the visible portion of the webpage quickly in relation to the objects on the page that are not visible.

Unlike in the other mechanisms, PF cannot be implemented with some additional interactions with the application itself. Fortunately, browser applications have well

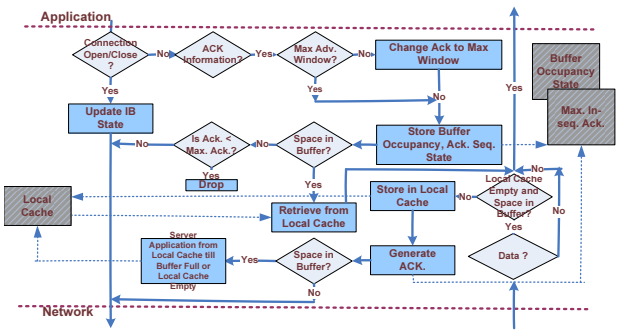


Figure 10: Infinite Buffering

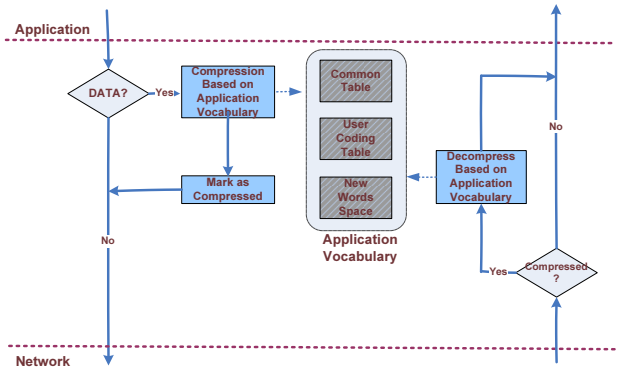


Figure 11: Application-aware Encoding

defined interfaces for querying state of the browser including the current focus window, scrolling information, etc. Hence, the implementation of PF relies on a separate module called the application state monitor (ASM) that is akin to a browser plug-in to coordinate its operations.

When a message comes in from the application, PF checks to see if the message is a request. If it is not, it is let through. If it is, PF checks with the ASM to see if all the requested content are immediately required. ASM classifies the objects requested as being of immediate need (visible portion of webpage) or as those that are not immediately required. PF then sends out fetch requests immediately for the first category of objects, and uses a low-priority fetching mechanism for the remaining objects.

Since A^3 is a platform solution, all PF has to inform the A^3 server is that certain objects are of low priority through A^3 specific piggybacked information. The A^3 server then de-prioritizes the transmission of those objects in preference to those that are of higher priority. Note that the relative prioritization is used not only between the content of a single end-device, but also across end-devices as well to improve overall system performance. Approaches such as TCP-LP[14] are candidates that can be used for the relative prioritization, although A^3 currently uses a simple priority queuing scheme at the A^3 server.

Note that while the ASM might classify objects in a particular fashion, changes in the application (e.g. user scrolling down) will result in a re-prioritization of the objects accordingly. Hence, the ASM has the capability of gratuitously informing PF about priority changes. Such changes are immediately notified to the A^3 server through appropriate requests.

4.3.4 *Infinite Buffering*

Figure 10 shows the flow chart for the implementation of IB in the context of SMTP. IB keeps track of TCP connection status, and monitors all ACKs that are sent out by the TCP connection serving the SMTP application for SMTP-9 and SMTP-10. If the advertised window in the ACK is less than the maximum possible, IB immediately resets the advertised window to the maximum value, and appropriately updates its current knowledge of the connection's buffer occupancy and max in-sequence ACK information.

Hence, IB prevents anything less than the maximum buffer size from being advertised. However, when data packets arrive from the network, IB receives the packets and checks to see if the local disk based cache is empty and the connection buffer can accommodate more packets. If both conditions are true, IB delivers the packet to the application. If the disk cache is non-empty, the incoming packet is directly added to the cache. In this case, IB generates a proxy ACK back to the server. Then, if the connection buffer has space in it, packets are retrieved from the disk cache and given to the application till the buffer becomes full again. When the connection sends an ACK for a packet already ACKed by IB, IB suppresses the ACK. When the connection state is torn down for the CIFS application, IB resets state accordingly.

4.3.5 *Application-aware Encoding*

Figure 11 shows the flow-chart for the implementation of AE for SMTP. When AE receives data (SMTP-9) from the SMTP application, it uses its application vocabulary table to compress the data, and marks the message as being compressed and forwards it to the network. The marking is done to inform the A^3 server about the need to perform de-compression. Similarly, when incoming data arrives for the SMTP server, and the data is marked as compressed, AE performs the necessary de-compression.

The mechanisms used for the actual creation and manipulation of the vocabulary tables are of importance to AE. In A^3, the SMTP vocabulary tables are created and maintained purely on a user pair-wise basis. Not only are the table created in this fashion, but the data sets over which the vocabulary tables are created is also restricted to this pair-wise model. In other words, if A is the sender and B is the receiver, A uses its earlier emails to B as the data set on which the A-B vocabulary table is created, and then uses this table for encoding. B, having the data set already (since the emails were sent to B), can exactly recreate the table on its side and hence decode any compressed data. This precludes the need for exchanging tables periodically, and also takes advantage of changes in vocabulary sets that might occur based on the recipient.

4.4 A^3 Point Solution - A^3•

While the A^3 deployment model assumed so far is a platform model requiring participation by A^3 enabled devices at both the client and server ends, in this section we describe how A^3 can be used as a point-solution, albeit with somewhat limited capabilities. We refer to the point-solution version of A^3 as A^3•.

Of the five design elements in A^3, the only design element for which the platform model is mandatory is the application-aware encoding mechanism. Since compression or encoding is an end-to-end process, A^3• cannot be used with AE. However, each of the other four principles can be

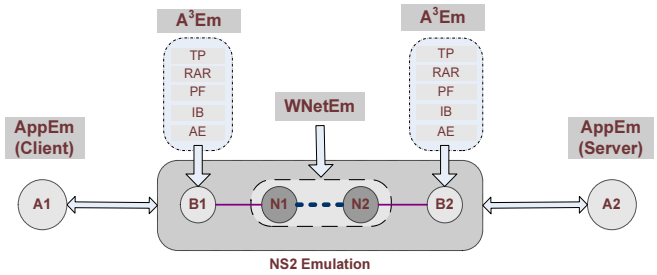


Figure 12: Evaluation Network Topology

employed with minimal changes in A^3•.

TP involves the generation of predictive data requests, and hence can be performed in A^3• as long as the application server can accept multiple simultaneous requests. For CIFS and HTTP, the servers do accept simultaneous requests. IB is purely a flow control avoidance mechanism, and can be realized in A^3•. RAR involves redundant transmissions of messages, and hence can be implemented in A^3• as long as application servers are capable of filtering duplicate messages. If the application servers are not capable of doing so (e.g. HTTP, which would respond to each request), the redundant transmissions will have to be performed at the granularity of transport layer segments as opposed to application layer messages, since protocols such as TCP provide redundant packet filtering. Finally, PF can be accomplished in A^3• in terms of classifying requests and treating the requests differently. However, the slow fetching of data not required immediately has to be realized through coarser receiver based mechanisms such as delayed requests as opposed to the best possible strategy of slowing down responses as in A^3.

5. EVALUATION

5.1 Experimental Setup

The experimental setup for the performance evaluation is shown in Figure 12. The setup consists of three desktop machines running the Fedora Core 4 operating system with the Linux 2.6 kernel. An application-emulator (AppEm) module runs on both the end machines. The AppEm module is a custom-built user-level module that generates traffic patterns and content for three different application protocols - CIFS, SMTP, and HTTP.

The traffic patterns are modeled based on traffic traces generated by the IxChariot emulator, and documented standards for the application protocols. The AppEm module also generates traffic content based on both input real-life data-sets (for Email and Web content), and random data-sets (File transfer)[3]. The traffic patterns shown in Figure 2 are representative of the traffic patterns generated by AppEm.

The system connecting the two end-systems runs the emulators for both A^3 and the wireless network. Both the emulators, A^3-Em and WNetEm, are implemented within the framework of the *ns2* simulator, and ns2 is running in the emulation mode. Running ns2 in its emulation mode allows for the capture and processing of live network traffic. The emulator object in ns2 taps directly into the device driver of the interface cards to capture and inject real packets into the network.

All five of the A^3 mechanisms are implemented in the A^3-Em module, and each mechanism can be enabled either independently or in tandem with the other mechanisms. The WNetEm module is used for emulating different wireless network links representing the WLAN, WWAN, and SAT environments. The specific characteristics used to represent wireless network environments are the same as those presented in Section 2.

The primary metrics monitored are throughput, response time (for HTTP) and confidence intervals for the throughput and response time. Each data point is the average of 20 simulation runs and in addition we show the 90 % confidence intervals. The results of the evaluation study are presented in two stages. We first present the results of the performance evaluation of A^3 principles in isolation. Then, we discuss the combined performance improvements delivered by A^3.

5.2 Principles in Isolation

5.2.1 *Transaction Prediction*

We use CIFS as the application traffic for evaluating the performance of Transaction Prediction. The results of the TP evaluation are shown in Figure 13. The x-axes of the graphs show the size of the transferred file in MBytes and the y-axes are the application throughput in Mbps. The results show several trends: (i) Using wireless aware TCP (such as ELN, WTCP, and STP), the increase in throughput is very negligible. This trend is consistent with the results in Section 2. (ii) Transaction Prediction improves CIFS application throughput significantly. In the SAT network, for instance, TP improves CIFS throughput by more than 80 % when transferring a 10 MByte file. (iii) The improvement achieved by TP increases with increase in file size. This is because TP is able to reduce more number of request-response interactions with increasing file size. (iv) TP achieves the highest improvement in SAT network. This is due to the fact that TP's benefits are directly proportional to the RTT and the BDP of the network, and SATs have high RTTs and large BDPs when compared to the other wireless environments.

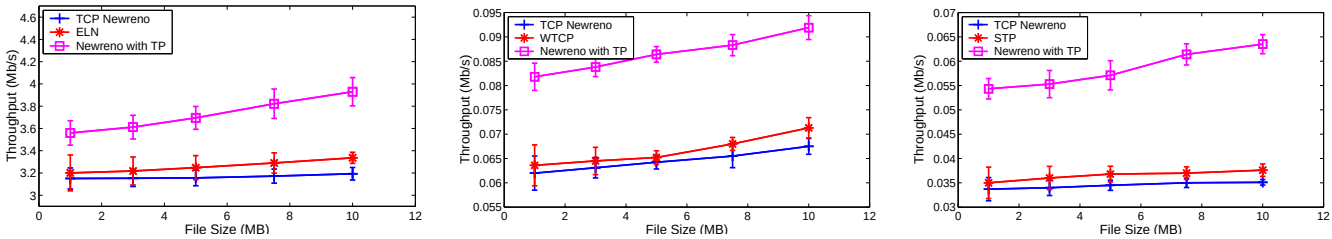
5.2.2 *Redundant and Aggressive Retransmissions*

We evaluate the effectiveness of the RAR principle using the CIFS application protocol. The results of the RAR evaluation is presented in Figure 14. The x-axis in the graphs is the requested file size in MBytes and the y-axis is the CIFS application throughput in Mbps. We observe that RAR delivers better performance when compared to both TCP-NewReno and the tailored transport protocols, delivering up to 80 % improvement in throughput performance for SATs. RAR is able to reduce the chances of experiencing a timeout when a wireless packet loss occurs. The reduction of TCP timeouts leads to better performance using RAR.

5.2.3 *Prioritized Fetching*

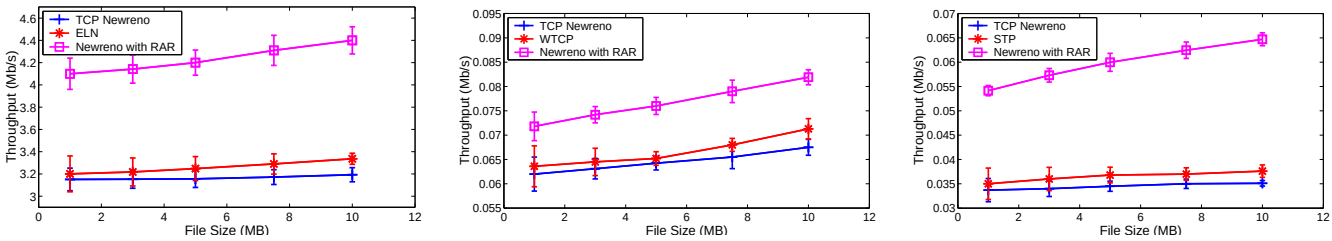
The performance of the PF principle was evaluated with HTTP traffic and results are shown in Figure 15. The x-axis in the graphs is the requested web-page size in KBytes, and the y-axis is the response time in seconds for the initial

[3]While the IxChariot emulator can generate representative traffic traces, it does not allow for specific data sets to be used for the content, and hence the need for the custom built emulator.



(a) WLAN (b) WWAN (c) SAT

Figure 13: Transaction Prediction: CIFS



(a) WLAN (b) WWAN (c) SAT

Figure 14: Redundant and Aggressive Retransmissions: CIFS

screen. In the figure, it can be seen that as a user accesses larger web pages, the response time difference between default content fetching and PF increases. PF consistently delivers a 15 % to 30 % improvement in the response time performance. PF reduces aggressive traffic volumes by deprioritizing the out-of sequence fetching of unseen objects. Note that PF, while improving the response time, does not improve raw throughput performance. In other words, only the effective throughput, as experienced by the end-user, increases when using PF.

5.2.4 *Infinite Buffering*

The effectiveness of IB is evaluated using CIFS traffic, and the results are shown in Figure 16. The x-axes of the graphs are requested file size in MBytes and the y-axes are the application throughput in Mbps. We can see that: (i) Transferring larger data size with IB achieves higher throughput. This is because of the fact that IB helps most during the actual data transfer phase, and will not help when the amount of data to be transferred is less than a few times the BDP of the network. (ii) IB performs much better in a SAT network than the other two networks, delivering almost a 400 % improvement in performance. Again, the results are as expected because IB's benefits are higher when the BDP of the network is higher.

5.2.5 *Application-aware Encoding*

Application-aware Encoding is designed primarily to accelerate e-mail delivery using SMTP and hence we evaluate the effectiveness of AE for SMTP traffic. In the evaluation, emails of sizes ranging from 1 KBytes to 10 KBytes (around 120 to 1200 words) are used. We show the results in Figure 17 where the x-axis is the e-mail size in KBytes and y-axis is the application throughput in Mbps. Varying degrees of throughput improvements are achieved, and in WWAN, an increase of 80 % is observed when transferring a 10 KBytes email. We can see that AE achieves the highest improvement in WWAN due to its relatively low bandwidth.

We also show the effectiveness of AE in terms of compression ratio in Figure 19. In the figure, the results of ten persons' emails using three compression estimators (WinRAR, WinZip and AE) are shown. We can see that WinRAR and WinZip can compress an email by a factor of 2 to 3, while AE can achieve a compression ratio of about 5.

5.3 Integrated Performance Evaluation

In this section, we present the results of the combined effectiveness of all applicable principles for the three applications, CIFS, SMTP and HTTP. We employ RAR, TP, and IB on the CIFS traffic. For SMTP, the RAR, AE and IB principles are used. In the case of HTTP, the A^3 principles applied are RAR, PF and IB. As expected, the throughput of the applications (CIFS and SMTP) when using the integrated A^3 principles is higher than when any individual principle is employed in isolation, while the response time of HTTP is lower than any individual principle. The results are shown in Figure 18, with A^3 delivering performance improvements of approximately 70 %, 110 %, and 30 % respectively for CIFS, SMTP, and HTTP.

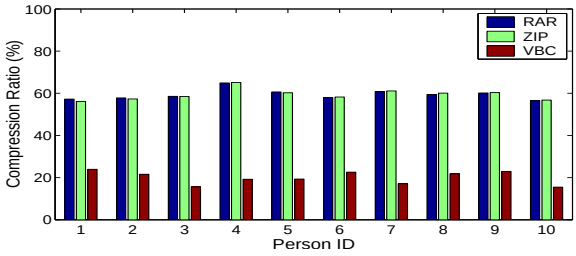
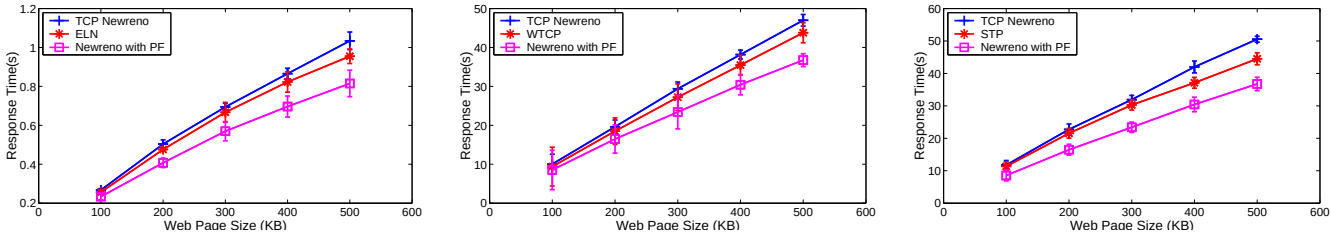


Figure 19: Efficiency of AE

6. RELATED WORKS

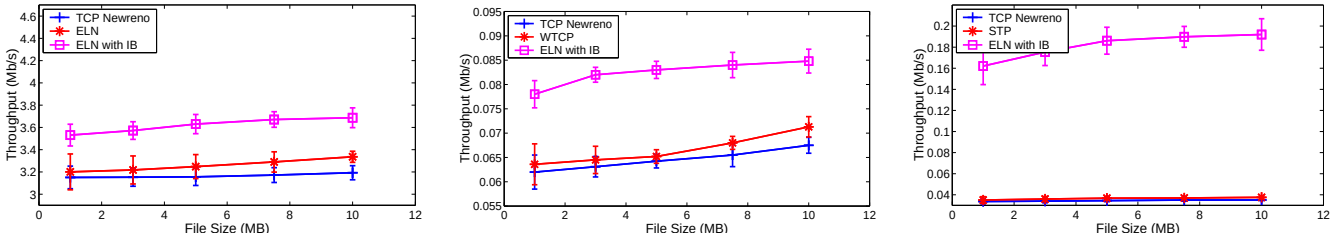
6.1 Wireless-aware Middleware/Applications

The Wireless Application Protocol (WAP) is a protocol developed to allow efficient transmission of WWW content to handheld wireless devices. The transport layer proto-



(a) WLAN (b) WWAN (c) SAT

Figure 15: Prioritized Fetching: HTTP



(a) WLAN (b) WWAN (c) SAT

Figure 16: Infinite Buffering: CIFS

cols in WAP consists of the Wireless Transaction Protocol and Wireless Datagram Protocol, which are designed for use over narrow band bearers in wireless networks and are not compatible with TCP. WAP is highly WWW centric, and does not aim to optimize any of the application behavioral patterns identified earlier in the paper. Browsers such as the Pocket Internet Explorer (PIE)[6] are developed with capabilities that can address resource constraints on mobile devices. However, they do not optimize communication performance which is the focus of A^3.

Work in [15] aims to save bandwidth/power by adapting the contents based on user semantics and contexts. The adaptations, however, are exposed to the end-applications and users. This is different from the A^3 approach which is application-transparent.

The Odyssey project[16] focuses on system support for collaboration between the operating system and individual applications by letting them both be aware of the wireless environment, and thus adapt their behaviors. Comparatively, A^3 does not rely on OS-level support, and is totally transparent both to the underlying OS and the applications.

The Coda file system[17] is based on the Andrew File System (AFS), but supports disconnected operations for mobile hosts. When the client is connected to the network, it *hoards* files for later use during disconnected operations. During disconnections, Coda emulates the server, serving files from its local cache. Coda's techniques are specific to file systems, and require applications to have changed semantics for the data that they use.

6.2 Related Design Principles

Some related works in literature have been proposed to accelerate applications with various mechanisms. We present a few of them here, and identify the differences vis-a-vis A^3.

TP-related: In [10], the authors propose to "upload" clients' task to the server side, thus eliminating many RTTs required for applications like SMTP. This approach is different from the A^3 approach in terms of application protocols applied and the overall mechanism.

RAR-related: Mechanisms like FEC use error control coding for digital communication systems. Another work[19] proposes aggressive retransmission mechanism to encourage legitimate clients to behave more aggressively in order to fight attack against servers. Compared to these approaches, A^3 only applies RAR to control messages in application protocols, and it does so by retransmitting the control message when a maintained timer expires. We present arguments earlier in the paper as to why protecting control message exchanges is a major factor affecting application performance.

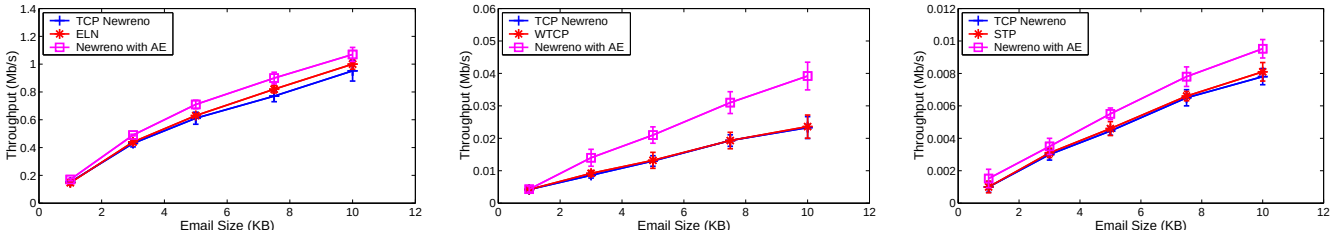
PF-related: To improve the web-access performance, work in [15] proposes out-of-order transmission of HTTP objects above UDP, and break the in-order delivery of an object. However, unlike the A^3 framework, it requires the conformation of both client and server sides.

IB-related: [8] shows that *overbuffering* on routers increases end-to-end delay in the presence of congestion, and complicates the design of high-speed routers. IB is different from overbuffering, which aims at fully utilizing the network resources by removing the buffer length constraint. IB specifically applies to applications with large bulk of data transfer, such as FTP, and is meant to counter impact of flow control.

AE-related: Companies like Converged[3] provide application aware compression solutions through compressing the data for some applications based on priority and application natures. These mechanisms share the property of being *application aware*, meaning only subset of applications will be compressed. However, AE has the property of being *user-aware*, that is take into considerations user-pattern information, and thus can achieve better performances.

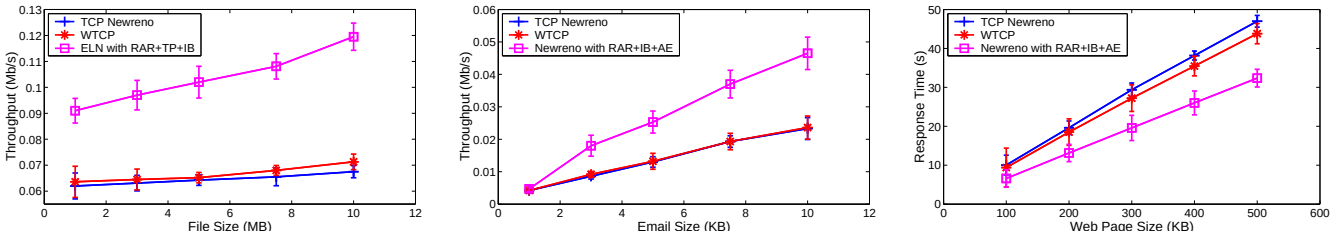
6.3 Commercial WAN Optimizers

Several companies, such as Riverbed and Peribit, sell WAN-optimization application-acceleration products. However, (1) Almost all the commercial solutions are proprietary ones; (2) The A^3 principles such as RAR, IB, AE and PF are not seen in commercial solutions; and (3) Many of the techniques used in commercial solutions, such as bit-level caching and



(a) WLAN (b) WWAN (c) SAT

Figure 17: Application-aware Encoding: SMTP



(a) CIFS (b) SMTP (c) HTTP

Figure 18: Integrated A^3 Results in WWAN

LPZ-based compression, are hardware-based approaches, and require large amounts of storage. The above properties render the commercial solutions inapplicable for environments where easy deployment is required. Also, A^3 is a middleware approach, and does not require large amounts of storage.

7. CONCLUSIONS

In this paper, we motivate the need for application acceleration for wireless-data networks, and present the A^3 solution that is application-aware, but application transparent. Using a combination of principles targeted toward tackling design problems in popular real-world applications, A^3 provides significant improvements in end-user application performance.

8. REFERENCES

[1] CIFS: A common internet file system. http://www.microsoft.com/mind/1196/cifs.asp.

[2] Comscore media metrix top 50 online property ranking. http://www.comscore.com/press/release.asp?press=547.

[3] Converged access wan optimization. http://www.convergedaccess.com/.

[4] Hypertext transfer protocol– http/1.1. http://www.ietf.org/rfc/rfc2616.txt.

[5] Linux magzine. http://www.linux-magazine.com/issue/15/.

[6] Pocket internet explorer. http://www.microsoft.com/windowsmobile/.

[7] Simple mail transfer protocol. http://www.ietf.org/rfc/rfc2821.txt.

[8] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM*, Portland, Oregon, 2004.

[9] H. Balakrishnan and R. Katz. Explicit loss notification and wireless web performance. In *Proceedings of IEEE GLOBECOM*, Sydney, Australia, Nov. 1998.

[10] S. Czerwinski and A. Joseph. Using simple remote evaluation to enable efficient application protocols in mobile environments. In *Proceedings of the 1st IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, 2001.

[11] T. Henderson and R. Katz. Transport protocols for Internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 17(2):345–359, Feb. 1999.

[12] H. Hsieh, K. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proceedings of ACM MOBICOM*, 2003.

[13] IXIA. http://www.ixiacom.com/.

[14] A. Kuzmanovic and E. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. *In Proceedings of IEEE INFOCOM*, 2003.

[15] I. Mohomed, J. C. Cai, S. Chavoshi, and E. de Lara. Context-aware interactive content adaptation. In *Proceedings of the 4th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Uppsala, Sweden, 2006.

[16] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, Saint Malo, France, 1997.

[17] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[18] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. In *Proceedings of ACM MOBICOM*, Seattle, WA, USA, Aug. 1999.

[19] M. Walfish, H. Balakrishnan, D. Karger, and S. Shenker. Dos: Fighting fire with fire. In *Proceedings of the 4th ACM Workshop on Hot Topics in Networks (HotNets)*, College Park, MD, 2005.