Locomotive: Optimizing Mobile Web Traffic Using Selective Compression

Themis Melissaris
Princeton University
themis@cs.princeton.edu

Kelly Shaw
University of Richmond
kshaw@richmond.edu

Margaret Martonosi
Princeton University
mrm@princeton.edu

***Abstract*—Mobile web traffic and application data demands are growing at a rapid rate and are at odds with resource-constrained, data-capped, wireless mobile devices. Data compression can be used to reduce web traffic, save energy, and make network transfers faster. Compression can, however, hurt performance if not used judiciously. We propose Locomotive, a library that improves the performance of network transfers in wireless mobile networks by employing *selective* compression based on data type and network conditions. We demonstrate that Locomotive improves performance of web transfers by roughly 12-24% while reducing data usage by 39%.**

I. Introduction

Increases in mobile device counts, improved network infrastructure and the broad adoption of mobile services and applications have led to an explosion in mobile data traffic. Mobile traffic is overtaking other network traffic and is expected to increase nearly threefold from 2015 to 2020 [5]. In fact, two-thirds of the total IP traffic by that time will be generated by wireless and mobile devices due to an increase in the number of available connected mobile devices as well as growth in the devices' capabilities and data consumption. Energy is a primary constraint in designing applications and systems for mobile devices and wireless communication accounts for a significant portion of the total energy budget, often dominating that of computation or other factors [4], [18]. With traffic and energy usage expected to surge, one technique proven to be efficient in managing the energy consumption and the traffic volume of wireless mobile devices is compression [13].

Recently, the rise of the Internet of Things (IoT) has led to the emergence of an ecosystem of networked devices, supporting services, and new applications across many different domains [10]. Application areas such as smart surveillance, traffic services, and mobile sensing, rely on data collected at the "edge" (e.g. on smartphones or other mobile devices, rather than wired devices or cloud infrastructure) followed by low-latency analysis of the data [9]. To support these data-intensive applications, our focus is on selectively using on-edge-device compression to reduce transferred bytecounts and improve communication efficiency.

Data compression and decompression are widely available on commodity servers and can also be used at the edge to reduce the data exchanged in the network, sometimes reducing network latencies as well. Compression, however, needs to be used correctly; otherwise it can add unnecessary latency and energy overhead. Whether compression is beneficial or not is determined by several factors outlined below.

First, different mobile services generate different types of content, which vary in size and compressibility. The type of traffic generated on edge devices can change dynamically based on the users' interaction with the devices and the applications in use. Traffic is usually comprised of scripts, plaintext, multimedia and markup documents. Such data can vary significantly in how compressible it is. For example, multimedia data items (e.g. audio, traffic and video) are usually already provided in a compressed format, preventing additional transfer-time compression from yielding large benefits.

Second, network behavior can significantly alter the effect compression has on data exchanged over the network. In cases of low network throughput, compression can reduce the duration of data transfers significantly. Alternatively, compression can stay in the application's critical path and introduce unnecessary overhead when the data compression rate is slower than the network data transfer rate.

To exploit the benefits of compression while mitigating its potential negative impact, we propose Locomotive (Latency Optimizations using COmpressed MObile Transfers In Variable Environments). Locomotive is a tool that allows mobile applications to handle compression intelligently. It dynamically evaluates the data to be transferred and the network conditions and automatically reasons about compression trade-offs. Based on this evaluation, it leverages selective compression and makes an informed compression decision in order to improve the performance of mobile web transfers.

As the edge increasingly includes data-intensive and latency sensitive applications, the bandwidth and performance of wireless mobile devices become key design challenges. Intelligently compressing data going to and from wireless mobile edge devices can aid in improving their functionality.

II. Related Work

Characterizing Mobile Web Traffic & Applications: One category of related prior work pertains to mobile web traffic characterization. The measurement study in [7] discusses mobile traffic composition and investigates the performance and energy efficiency of TCP transfers. Butkiewicz et al. [3] studies parameters that affect web page load times across websites, whereas WProf [16] performs in-browser performance profiling. In contrast to our work, these papers do not study compression, nor how the performance of a web transfer is affected by compressing data of varying data sizes and types.

Optimizing Mobile Web Traffic: Various techniques have been proposed to optimize mobile web transfers for performance and data usage. For example, Procrastinator [12] decides when to prefetch objects in order to manage application data usage depending on a user's connectivity and data plan limitations. Other techniques like Polaris [11] and Shandian [17] use fine grained dependency tracking to identify and eliminate the intrinsic inefficiencies in the page load process.

Additionally, compression proxies like Flywheel [1] and Flexiweb [14] offer data savings by leveraging compression. These approaches, however, channel mobile content through a proxy server. Such rerouting raises privacy and security concerns if the proxy is untrusted and potentially latency

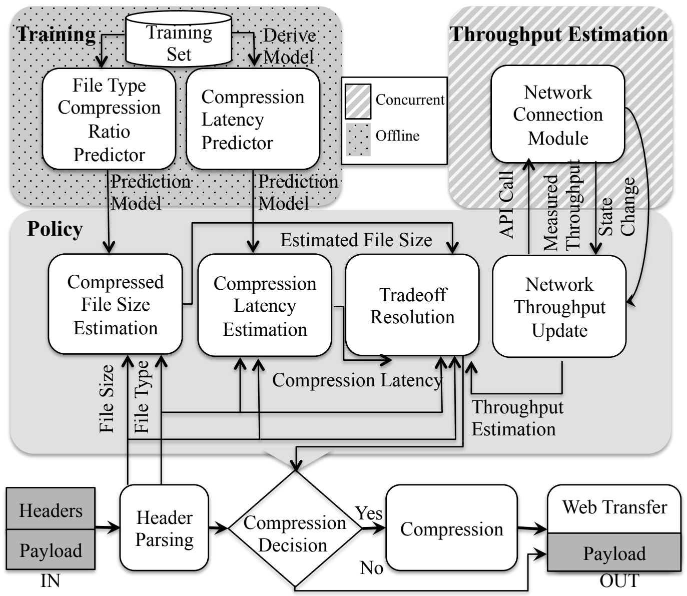

Fig. 1. Presentation of the Policy, Training, Throughput Estimation modules and execution overview in Locomotive.

concerns as well. Locomotive runs on mobile devices and mitigates such concerns.

Recently, industry has developed compression algorithms, like Brotli [8], specifically designed for mobile traffic data savings and performance. Prior work has demonstrated that custom compression algorithms can achieve significant energy and performance gains [13]. Using state of the art compression algorithms for mobile web traffic could complement network adaptive approaches such as Locomotive.

Our Approach: This paper presents Locomotive, a library that optimizes web traffic transfers to mobile edge devices through selective compression. Locomotive dynamically decides whether to compress based on file type characterizations and network conditions. Locomotive is non-browser specific and therefore capable of enhancing all types of mobile applications and mobile traffic. Section III presents Locomotive's architecture, Section IV describes our methodology and Section V presents results on performance and data savings.

III. Locomotive Library

Overview: This section describes Locomotive's usage and design. Locomotive eases IoT and mobile traffic optimization by providing hooks for application programmers to use. Mobile applications can invoke the library, abstracting away compression decisions and consideration of the type of mobile traffic and network conditions. Locomotive focuses on uplink traffic where compression happens on the edge device. To enable Locomotive, a component that responds to Locomotive requests and handles data decompression is running in the cloud. Locomotive uses Android HTTP primitives, but can be extended easily to accommodate other protocols.

Locomotive Policy: For all data transfers, Locomotive makes a two-step compression decision. First, a threshold determines if some requests should not be considered for compression based on compressibility. For small transfer sizes and file types that are encoded such as multimedia, data compressibility is low. Locomotive next determines if the estimated transfer latency is less with or without compression.

TABLE I
Locomotive Optimization Model

(I)	$\frac{S_{Compressed}}{N_{Throughput}} + L_{Compression} > \frac{S_{Original}}{N_{Throughput}}$
(II)	$S_{Compressed}(S_{Original}, T) = \frac{S_{Original}}{compressibility(T)}$
(III)	$L_{Compression}(S_{Original}, T) = \alpha(T) \cdot S_{Original} + \beta(T)$

As shown in Equation (I), Locomotive resolves the trade-off for each request based on the compression latency $L_{Compression}$, the size of the request payload data before ($S_{Original}$) and after ($S_{Compressed}$) compression as well as the estimated network throughput, $N_{Throughput}$. $N_{Throughput}$ changes over time and is periodically estimated by the Network Connection Module.

Compression Size & Time Estimation: Locomotive performs estimations on the data type T and data size of the original data using two separate statistical linear regression based statistical models. In order to estimate the data size after compression, we use a model as described in equation (II) , where $compressibility$ is the ratio between the data size of the original data over the compressed. In order to estimate the data compression latency, equation (III) is used. Latency is a linear function of the data size and the coefficients α, β are functions of the data type, acquired offline via training. In order to determine model parameters for compressibility and compression latency for equations (II), (III), Locomotive performs training during an initialization period that happens after installation. As data patterns and network conditions vary over time, it is important to adapt to these changes. Our model parameters can be updated with an online approach such as stochastic gradient descent using linear regression. However, adaptation to variation is beyond the scope of this paper.

Network Throughput Estimation: Another estimate used is the current network throughput, $N_{Throughput}$. For this, we use the open source Network Connection Class library [6]. This library runs concurrently and notifies Locomotive whenever there is a significant change in network throughput. This library uses a moving average approach to adjust to changes in network throughput based on the sampling it performs. The moving average approach allows the library to gradually adjust to changes in network conditions, eliminating bursty behavior. As a result, Locomotive can sample the network throughput less frequently in order to keep performance and energy overheads low. In order to generate network throughput estimates, we provide samples to the Network Connection Class [6]. To achieve that, we log the time the data takes to get transferred to the receiving end, from the first to the last byte. After logging the size of the data, these samples are returned to the edge device and used by Locomotive clients in the Network Throughput module.

IV. Evaluation Methodology

We now present our evaluation methodology and implementation of Locomotive on mobile edge devices.

Edge implementation: In order to evaluate Locomotive, we capture and replay web traffic on mobile devices using a test application that invokes Locomotive. Locomotive generates HTTP requests, which transfers previously collected real traffic data discussed later in this section. Processing via Locomotive proceeds as described in Section III. Once Locomotive has made a compression decision, the processed data will then be used to generate an HTTP request. Locomotive is intended to

TABLE II
AVERAGE COMPRESSIBILITY AND PERCENTAGE OF FILES IN THE DATASET FOR EACH DATA TYPE ARE PRESENTED.

Alexa Top 200 characteristics		
File Type	**Average Compressibility**	**Content Distribution**
Images	1.021	32.07%
HTML	4.177	7.32%
CSS	4.555	10.25%
Javascript	3.388	38.36%
Other	1.172	8.74%

handle arbitrary data transfers. For the purpose of this work, we focus on HTTP, but other data transfer protocols could also benefit similarly. The Locomotive client is implemented on Android and is run on Samsung Galaxy Nexus I9250 phone.

Cloud setup: To enable Locomotive, our infrastructure in the cloud is using a web server capable of responding to HTTP requests and decompressing the compressed data. For our evaluation, we vary network conditions in a controlled manner using Linux traffic shaping tools. The network throughput settings used are 2 Mbps, 5 Mbps and 10Mbps.

Benchmarks: To evaluate Locomotive, we collected web traffic to replay on mobile platforms. Collection of the mobile web traffic was performed offline using Fiddler [15], a web debugging proxy, which captures the raw payload of each request. To emulate real mobile traffic, we generate HTTP requests to transfer captured web traffic data as the request payload. We captured the traffic from mobile versions of the top 200 most popular websites according to the Alexa list [2]. For each of the mobile web pages to load, multiple HTTP requests are generated. The size of the dataset is 350MB and consists of 25 different data formats, including scripts (e.g. HTML, Javascript), text formats (e.g. .txt files, JSON and XML formatted text) and multimedia (e.g. jpeg, png images, audio files). We chose the Alexa Top 200 list as the list contains a broad spectrum of data sizes and data types. (To the best of our knowledge, there are no alternative benchmarks available capable of capturing traffic representative of the whole spectrum of edge devices.) Table II characterizes the Alexa Top 200 dataset per data type and focuses on data compressibility and content distribution by data size.

To study Locomotive's behavior on different levels of compressibility, we created two test sets using traffic from 50 different websites. Test set A includes the 25 most compressible websites (average compressibility 3.07) and test B includes the 25 least compressible websites (average compressibility 1.23) of the Alexa Top 200 list. For our selection, we eliminated web sites that were either very small in size (order of a few tens of Kilobytes) or contained a small number of files. The remaining 150 websites comprise our training set, which we use to train Locomotive's models.

In the experiments performed, we compare Locomotive against other approaches using the aforementioned benchmarks. As the applications and the target hardware vary significantly, we are not using mobile web browsers for our evaluation; instead, we focus on the total time required for a benchmark to complete the transfer over the network as well as the compression and decompression latency at the endpoints.

Performance Evaluation: For the purpose of performance evaluation, we compare Locomotive against a policy that performs all data transfers uncompressed (denoted as Uncompressed), an approach that compresses all data before they get transferred (denoted as Compressed) and against an oracle (denoted as Oracle). The oracle always makes a correct

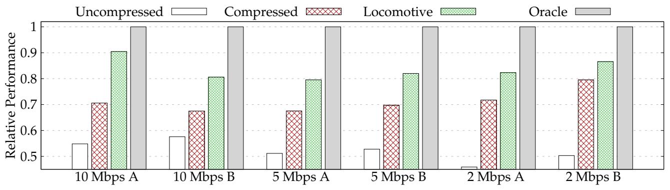

Fig. 2. Locomotive results across different throughput settings and test sets. *Uncompressed* refers to performing all data transfers uncompressed, *Compressed* refers to compressing all data without selectivity and the *Oracle* always answers the question whether to compress or not correctly. Test set A has high and test set B has low compressibility. Performance is normalized to the *Oracle*.

decision when reasoning about the compression decision as it is constructed by choosing the minimum request latency between compressing data and leaving it uncompressed for each individual web data transfer.

For the library's performance evaluation, we use the Alexa test set discussed previously. For each of those websites, there are multiple requests required to be completed in order for a page to be fetched and loaded on the client. In our test set, each website contains multiple files in the range of 2 to 200. In our experiments, we replay HTTP requests for each data file that is captured as part of the website. The resulting time required to transfer a website is the aggregate time of individual data transfer times. Locomotive's evaluation spans across both test sets and across different fixed network settings.

V. LOCOMOTIVE EVALUATION

This section presents Locomotive's evaluation. More specifically, we compare the performance of Locomotive against the (i) Uncompressed, (ii) Compressed and (iii) Oracle approaches. In addition, we present statistics that showcase Locomotive's efficiency and discuss the most significant prediction errors that affect its accuracy. We perform all experiments under controlled network settings, as discussed previously.

Figure V presents the comparison of the aforementioned approaches using bandwidth thresholds at 2Mbps, 5Mbps and 10Mbps. The benchmarks used are distributed across two different test sets, A and B, with different compressibility characteristics as described in Section IV. In Figure V, we present relative performance of Locomotive, as indicated by averaging the transfer times of all data requests for each of the benchmarks. The averaged times for each approach are shown normalized relative to Oracle—an ideal scenario in which Locomotive makes decisions correctly across all web data. Oracle is plotted at an ideal of 1, and other techniques seek to approach it; higher is better.

Figure V indicates that Locomotive performs on average better than the Compressed and Uncompressed approaches regardless of the network conditions or the benchmarks used. When comparing results across different network conditions, Locomotive performs better relative to the Uncompressed approach when the available bandwidth is low. That is expected since network transfers become gradually more expensive as bandwidth declines. Compression makes better use of limited bandwidth, both in terms of sending less data to begin with, and also in terms of requiring fewer retries. Similarly, Locomotive performs better as the network throughput increases, compared against the Compressed approach. Using compression when the network is fast can be inefficient, as the additional compression latency can outweigh the benefits of transferring

TABLE III
OVERVIEW OF PERFORMANCE AND ACCURACY STATISTICS FOR LOCOMOTIVE ACROSS DIFFERENT NETWORK CONDITIONS. COMPARISON TO THE *Uncompressed*, *Compressed* AND *Oracle* APPROACHES PRESENTS THE PERCENTAGE OF BENCHMARKS SPED UP WITH LOCOMOTIVE, THE RESPECTIVE AVERAGE SPEEDUP AND THE LIBRARY'S OVERHEAD.

	Network conditions		
	2 Mbps	**5 Mbps**	**10 Mbps**
Percentage win over Compressed	76%	86%	86%
Percentage win over Uncompressed	92%	88%	90%
Speedup vs Uncompressed	1.78	1.56	1.53
Speedup vs Compressed	1.12	1.19	1.24
Locomotive overhead	0.26%	0.36%	0.56%
Slowdown vs Oracle	0.84	0.81	0.86

less data. Locomotive shows performance advantages against both the Compressed and Uncompressed approaches.

By dividing the test sets according to compressibility, we can compare Locomotive's advantages in two distinct scenarios. Test set A (more compressible) provides better performance than test set B, since there are more opportunities to reduce network transfer times when utilizing compression. Test set B contains data with low compressibility, which Locomotive often chooses to leave uncompressed, and often due to the file size and type threshold criteria. This approach proves to be beneficial in the case of low throughput at 2Mbps and 5Mbps. When throughput is low, compressing is the common case, as savings from network transfers are more significant. Therefore, Locomotive makes fewer errors on test set B and has less slowdown against the Oracle compared to test set A. However, that does not hold for 10Mbps, where Locomotive performs marginally better on test set A.

Table III presents statistics about Locomotive's performance across different network settings. First, the table presents the percentage of benchmarks for which Locomotive outperforms the Compressed and Uncompressed approaches. The table presents results at a per-benchmark granularity, comparing between the aggregate network transfer time of each website per approach. In addition, the table depicts the average speedup of Locomotive across both test sets for the Uncompressed and Compressed approaches and its performance relative to the oracle. It also presents Locomotive's average overhead, which ranges from 0.26% to 0.56% of the total transfer time of a request. Because the overhead is low, it enables using Locomotive to reason about every network transfer. The average speedup of Locomotive against the Compressed approach is ranging from $1.1\times$ at 2Mbps to $1.2\times$ at 10Mbps, whereas it ranges from $1.5\times$ at 10Mbps to almost $1.8\times$ at 2Mbps when compared to the Uncompressed approach. When compared to the oracle, Locomotive is on average 14.5%-19.9% slower.

Locomotive is designed to inherently provide data savings, as it bases its approach on compression. We measured the different data savings the library provides across different network conditions. Locomotive saves roughly 40% at 2, 5, and 10Mbps, while providing applications with performance speedup. The majority of data savings originates from large transfers of highly compressible data. Since the theoretical bound for data savings is calculated at 43.4% using gzip, Locomotive demonstrates results close to the theoretical best.

The performance results comparing Locomotive to the Uncompressed and Compressed policies are on par with Locomotive's prediction accuracy, as depicted in Table IV. These results demonstrate the accuracy of Locomotive across different network conditions. For this comparison, an oracle is used as ground truth. Locomotive is hindered by two different kinds of errors: when it decides to compress when it should not (false positives) and when it fails to identify that compression is beneficial (false negatives). The most significant source of error is false negatives, which are responsible for the majority of incorrect predictions, ranging between 18.6%-21.9%. We observe that the false negative error rate decreases as the bandwidth increases. In addition, false positives range between 3.4% and 4.9%.

TABLE IV
CHARACTERIZATION OF THE MOST SIGNIFICANT PREDICTION ERRORS FOR LOCOMOTIVE USING AN ORACLE AS GROUND TRUTH.

Network conditions	**2 Mbps**	**5 Mbps**	**10 Mbps**
Correct Prediction	74.66%	74.36%	77.07%
False positives	3.41%	4.90%	4.34%
False negatives	21.93%	20.74%	18.59%

VI. CONCLUSION

This paper presented Locomotive, a library for optimizing mobile web traffic. Locomotive implements selective compression—using it only when it is likely to benefit performance. To support this, the Locomotive approach uses compression latency and network throughput estimates to reason about the compression decision of each web transfer.

We find that Locomotive performs consistently better than uniform policies requiring either all-compressed or all-uncompressed data. Its prediction accuracy is above 70% and its resulting runtime latency outperforms these naïve policies more than 76% of the time. Furthermore, it also approaches the oracle policy in many cases.

Overall, Locomotive represents an important building block towards broader implementation of traffic-reduction techniques that can improve latency, save energy, and reduce the bandwidth requirements for mobile applications and devices.

REFERENCES

[1] V. Agababov et al. Flywheel: Google's data compression proxy for the mobile web. In *NSDI'15*, 2015.
[2] Alexa list web page. http://www.alexa.com/topsites.
[3] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding Website Complexity: Measurements, Metrics, and Implications. In *IMC '11*. ACM, 2011.
[4] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *USENIX ATC'10*, 2010.
[5] Cisco Visual Networking Index: Forecast and Methodology, 2015-2020.
[6] Facebook Network Connection Class. https://code.facebook.com/projects/1547113495553528/network-connection-class/.
[7] H. Falaki et al. A First Look at Traffic on Smartphones. In *IMC '10*, 2010.
[8] Google. Brotli Compression Format. https://github.com/google/brotli.
[9] K. Hong et al. Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things. In *MCC '13*, 2013.
[10] R. Khan et al. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In *FIT '12*, 2012.
[11] R. Netravali et al. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *NSDI '16*, 2016.
[12] L. Ravindranath et al. Procrastinator: Pacing Mobile Apps' Usage of the Network. In *MobiSys '14*, 2014.
[13] C. M. Sadler and M. Martonosi. Data Compression Algorithms for Energy-constrained Devices in Delay Tolerant Networks. In *SenSys '06*, 2006.
[14] S. Singh et al. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *MobiCom '15*, 2015.
[15] Telerik Fiddler Debugging Proxy. http://www.telerik.com/fiddler.
[16] X. S. Wang et al. Demystifying Page Load Performance with WProf. In *NSDI '13*, 2013.
[17] X. S. Wang, A. Krishnamurthy, and D. Wetherall. Speeding up Web Page Loads with Shandian. In *NSDI' 16*, 2016.
[18] P. Zhang et al. Hardware Design Experiences in ZebraNet. In *SenSys '04*, 2004.