How to Reduce Smartphone Traffic Volume by 30%?

Feng Qian[1], Junxian Huang[2], Jeffrey Erman[1], Z. Morley Mao[2],
Subhabrata Sen[1], and Oliver Spatscheck[1]

[1] AT&T Labs – Research
[2] University of Michigan

Abstract. The unprecedented growth in smartphone usage has fueled a massive increase in cellular network traffic volumes. We investigate the feasibility of applying Redundancy Elimination (RE) for today's smartphone traffic, using packet traces collected from 20 real mobile users for five months. For various RE techniques including caching, file compression, delta encoding, and packet stream compression, we present the first characterization of their individual effectiveness, the interaction among multiple jointly applied RE techniques, and their performance on mobile handsets. By leveraging several off-the-shelf RE techniques operating at different layers, we can achieve an overall reduction of smartphone traffic by more than 30%.

1 Introduction

Mobile data traffic is experiencing unprecedented growth. Cisco predicted that from 2011 to 2016, global smartphone traffic will increase by 5000% [4]. Meanwhile, in 2011, the cellular infrastructure expenditure was expected to be only a 6.7% increase over 2010 [1]. From the customers' perspective, reducing the bandwidth consumption effectively lowers usage-based data charges, and decreases page download times.

Network Redundancy Elimination (RE) plays a crucial role in bandwidth reduction by preventing duplicate data transfers and making the transferred data more compact [8]. In our recent work [17], we investigated HTTP caching on smartphones. We found that for web caching, there exists a huge gap between the protocol specification and the implementation on today's mobile devices. A surprisingly high 17% reduction in the traffic volume can be achieved if just the HTTP caching protocol is fully supported and strictly followed by smartphone applications and mobile browsers. This begs the question: *What about other off-the-shelf RE techniques?* The potential savings from applying these techniques to smartphone traffic are not known quantitatively.

To answer this question, we investigate the feasibility of redundancy elimination for today's smartphone traffic, using packet traces collected from 20 real mobile users for five months. For various RE techniques including caching, file compression, delta encoding, and packet stream compression, we present the first characterization of:

- **Their effectiveness on smartphone traffic.** Previous studies [8][6][7][15] investigated RE techniques for wired traffic, whose content and protocol compositions significantly differ from those of smartphone traffic.

M. Roughan and R. Chang (Eds.) PAM 2013, LNCS 7799, pp. 42–52, 2013.

- **Their interaction when jointly applied.** Prior work [10][14][18] only studied RE techniques in isolation for mobile networks. Jointly employing multiple techniques can potentially save more bandwidth.
- **Their computation load on mobile handsets.** Such considerations are important given mobile handsets are more limited in computation capabilities compared to desktop counterparts.

Our key finding is that, a judicious composition of several off-the-shelf RE techniques operating at different protocol layers can achieve an overall reduction in smartphone traffic by more than 30% with acceptable runtime overheads. In comparison, HTTP caching by itself saves as much as 17% of the overall traffic (§5.2). Such high savings become more interesting and somewhat surprising given that a major fraction of the traffic is video, audio, or image which are already compactly encoded.

2 Related Work

We describe related work in three categories.

RE Algorithms. Data compression techniques, such as gzip, are the most well-known RE approach. An orthogonal approach is caching. Specifically, web caching can be extremely useful in reducing HTTP traffic [10]. Other methodologies include delta encoding [13] and packet stream compression [19][14]. We study the effectiveness and efficiency of these well-established techniques for smartphone traffic.

RE Measurements. A recently study [10] explored the potential benefits of in-network caching at the cellular gateway. Gember *et al.* [12] reported high intra-user redundancy of handheld traffic in campus Wi-Fi networks. The above studies motivated us to make a further step by examining different RE techniques and their interplay when applied jointly for mobile traffic. Anand *et al.* [8] conducted a trace-driven study of packet stream compression [19] for university and enterprise traffic. Earlier RE measurements also focused on delta encoding and file compression [15].

RE Systems. [6] proposed incorporating RE into an IP-layer service on routers. The SmartRE [7] architecture eliminates network-wide redundancy by coordinating multiple devices. EndRE [5] is an end-to-end service where packet-stream-based RE is put into the protocol stack. PACK [20] is an RE system designed for cloud computing customers. Our measurement provides useful insights for designing future RE systems for mobile networks, which none of the above systems specifically focuses on.

3 The Measurement Data

The dataset used in this study was collected from 20 users from May 12 to October 12 2011. They were given 11 Motorola Atrix and 9 Samsung Galaxy S smartphones, all running Android 2.2, with unlimited voice, text and data plans from a large 3G carrier in the U.S. This dataset was also used in our earlier study of smartphone HTTP caching [17][1]. We deployed custom data collection software on the 20 handsets. It runs

[1] The dataset will be available for verification purposes under NDA after relevant IRB approvals.

in the background and collects the full packet traces (with payload) for both cellular and Wi-Fi traffic. We collected 118 GB of packet traces during the five-month trial.

The participants were selected to be students from 8 departments at University of Michigan. Their individually contributed traffic volume ranges from 0.3 GB to 23.6 GB. Overall, 15,683 distinct values of **`Host`** fields appeared in HTTP requests. Across all user pairs (X, Y), the overlaps of **`Host`** sets $|H_X \cap H_Y|/|H_X \cup H_Y|$ range from 1% to 25% (H_X consists of all **`Host`** strings in the requests made by user X). We therefore believe the 20 participants are reasonably diverse smartphone users.

4 Explored RE Techniques

We explored four different RE approaches. These are extremely popular and representative techniques for reducing network traffic redundancy.

HTTP Caching. In [17], we found for web caching, there exists a huge gap between the protocol specification and the implementation on today's mobile devices. A 17% reduction in the overall traffic volume can be achieved if the HTTP caching protocol is fully supported and strictly followed by smartphone apps and mobile browsers.

Delta Encoding. In Delta encoding, instead of transferring a file in its entirety, only any difference from its previously transferred version (if exists) is sent. We used VCDIFF [13] (RFC 3284), known as the best overall delta encoding algorithm[16].

File Compression. We study three off-the-shelf file compression techniques selected due to their popularity: gzip, bzip2, and 7-zip[2]. gzip is based on the well-known DEFLATE algorithm [9]. bzip2 employs diverse compression techniques such as Huffman coding, Burrows-Wheeler transform, and run-length encoding. 7-zip uses Lempel-Ziv-Markov chain algorithm (LZMA), which is also a dictionary-based approach similar to DEFLATE but features a higher compression ratio.

Packet Stream Compression. Compression can also be performed in an *application-agnostic* manner where the IP packet stream is compressed at one end of a network path (*e.g.,* the cellular gateway) and is decompressed at the other end (*e.g.,* a handset). We employ MODP [19], a representative packet stream compression algorithm. MODP was also used in existing RE systems such as [6].

We briefly explain how MODP works. Two *packet caches*, whose contents are synchronized, are deployed at both ends of a network path. To compress an incoming packet, the ingress end *(i)* fingerprints byte subsequences in the packet by sliding a running window over it, *(ii)* matches the fingerprints against a *signature table*, which contains mappings from fingerprints to pointers to the cached packets, *(iii)* for matched fingerprints, replaces their byte subsequences with the pointers, *(iv)* inserts the new packet into the packet cache and updates the signature table. The decompression procedure is straightforward: the egress end simply follows the pointers and replaces them with byte sequences in the cache. The algorithm involves two parameters:

[2] `http:// www. gzip. org, http:// bzip. org, http:// www. 7-zip. org`

the packet cache size n, and the sampling rate for fingerprint generation p[3]. Selecting their values involves trading off the compression ratio and the processing speed.

5 Measurement Results

We apply the aforementioned RE techniques to our dataset to study their effectiveness.

5.1 Evaluation Methodology

We perform RE in the following order. In the remainder of this paper, we refer to a web object (*e.g.,* an HTML document) carried by an HTTP response as a *file*.

Step 1: Web Caching. We eliminate redundant transfers due to problematic caching behaviors by assuming a good HTTP caching implementation that *(i)* strictly follows the protocol specification [11], and *(ii)* has a non-volatile LRU cache shared by all applications. The cache size is assumed to be 256 MB. As long as the cache size is not too small (*e.g.,* >50 MB), it has little impact on the RE effectiveness, as shown in [17].

Step 2: Delta Encoding. Assume a handset has requested for file f, and there is already a copy of f in the cache (a file is keyed by its full URL including query strings). If the content of f has changed, we use VCDIFF to encode the delta between the new and the old version, to save the bandwidth. If f is not expired or not changed, the standard caching procedure (Step 1) is used although VCDIFF can also handle two identical inputs and output a delta of zero.

Step 3: File Compression. The file is compressed by an off-the-shelf compression technique such as gzip, unless it is already compressed in the trace or by Step 2.

Step 4: Packet Stream Compression. We use MODP to compress all the IP packets in both directions between the cellular gateway and the handset.

Steps 1 to 3 are object-level RE schemes. In theory these general techniques can be applied to any application-level objects. However, here we apply them to only HTTP traffic that dominates smartphone traffic usage [8][12]. In particular, encrypted HTTPS traffic over TCP port 443 accounts for 11.2% of the bytes – we are unable to apply objected-based RE techniques to them (the data collector runs below the SSL library). Hence the reported RE effectiveness is an *underestimation* of the actual possible gains. Also, Step 1 and 3 are already part of the HTTP specification [11] but today's smartphones and web servers may not strictly follow or fully utilize them. *We quantify the additional benefits that can be gained if they do so.*

The Ordering of the Four Steps is justified as follows. We consider caching (Step 1) first since it can potentially avoid transferring the entire file. If Step 2 is performed, then Step 3 will be skipped, because delta encoding usually yields a more compact output than compressing a single file does. Note that in Step 2, the output of VCDIFF (*i.e.,* the delta) is always compressed (using gzip by default). Step 4 is applied at the end of the pipeline because packet stream compression is performed on a network path after packets leave the server.

[3] Fingerprints are indexed probabilistically since indexing all is computationally impractical.

Table 1. Compression Ratios (CR) for caching, file compression, and delta encoding, when each of them is individually applied. The "HTTP" row and the "All" row correspond to CR values computed for only HTTP traffic, and the overall traffic, respectively.

	1.	2-4. File Compression (lv 1–9)			5-7. Lower Bound (lv5)			8-9. Δ Encoding	
	Caching	gzip	bzip2	7-zip	gzip	bzip2	7-zip	T & NT*	NT only
HTTP	79.8%	83.9–84.5%	84.4–84.9%	82.5–82.5%	80.4%	78.9%	71.7%	77.8%	98.0%
All	82.7%	86.3–86.8%	86.7–87.1%	85.1–85.1%	83.3%	82.0%	75.8%	81.0%	98.3%

* "T": trivial cases (two versions are identical); "NT": non-trivial cases (two versions are different).

Implementation of RE Techniques. Step 1 was realized by a standard web caching simulator correctly following the HTTP protocol. Step 2 and 3 were implemented by using open-source projects of xdelta 3.0 (`http:// xdelta. org/`, for VCDIFF), LZMA SDK 9.20 (for 7-zip), bzip2 1.0.6, and gzip 1.2.4, all having a tunable parameter between 1 (least compact but fastest) and 9 (most compact but slowest) allowing users to balance between compression ratio and speed. We implemented the MODP algorithm in C++ based on a recent paper [14] that improves the original algorithm [19].

The Key Evaluation Metric is the *Compression Ratio* (CR), defined as the ratio of traffic volume after compression to the traffic volume of the original trace. A smaller CR indicates more effective compression. CR is consistently used in Tables 1 to 4.

5.2 Applying Individual RE Approaches

We first examine each individual RE approach.

Overall Statistics. The dataset consists of 118 GB of packet traces dominated by downlink traffic (93% of the bytes go from the Internet to handsets). As identified by the HTTP parser, 85.4% of all traffic is HTTP that can be potentially optimized by the object-based RE techniques described in §5.1.

TCP/IP Headers. We exclude all TCP/IP headers from our analysis because they can be effectively compressed by mobile networks (*e.g.,* UMTS uses the Packet Data Convergence Protocol [2] for header compression) but in our data collected on handsets they were captured as uncompressed.

We discuss key results in Table 1. As indicated by the "Caching" column (Column 1), good web caching implementation reduces the overall traffic volume by 17%.

File Compression. In Columns 2 to 4, all three file compression techniques effectively achieve compression ratios (CR) between 82.5% and 84.9% for HTTP traffic. Neither the algorithm nor the compression level impacts the CR significantly. This can be explained as follows. We first note that compression is likely to yield more gains for *smaller* files, which tend to be uncompressed text files. In contrast, large files are usually audio, video, or files already compressed in the data. Compressing them further brings little additional benefits regardless of the compression method (Table 2). This is validated by the fact that the overall CR value (gzip level 5) for all responses under 100KB (they account for 33% of the total HTTP response volume) is 71%, compared to 93% for all responses of at least 100KB. This confirms the intuition that most

Table 2. Effectiveness of gzip compression (lv 5) on different content types. Content types with CR values less than 30% (*i.e.,* compression is under-utilized) are highlighted.

`Content-Type`	% bytes	% NC*	CR (gzip)	`Content-Type`	% bytes	% NC*	CR (gzip)
video/mp4	19.34%	100.00%	97.84%	video/x-flv	4.65%	99.85%	98.64%
app/octet-stream	13.14%	99.44%	95.21%	text/html	3.74%	70.11%	23.49%
(App market)	12.46%	100.00%	86.83%	text/javascript	2.57%	58.17%	27.44%
image/jpeg	10.20%	99.47%	88.90%	image/png	2.40%	97.77%	90.86%
audio/mpeg	8.14%	99.99%	97.07%	app/x-javascript	2.34%	59.48%	29.45%
video/3gpp	6.34%	100.00%	96.86%	video/flv	1.61%	100.00%	97.86%
text/xml	5.23%	98.18%	14.59%	text/css	1.27%	85.84%	19.34%

* "NC": The fraction of bytes that are *Not Compressed*.

gains come from small files. Secondly, most reasonable compression techniques tend to perform similarly for small files – probably because redundancy patterns in smaller files are usually easier to discover so even using a lightweight compression technique less aggressively (*e.g.,* gzip with a small dictionary) can achieve a reasonable CR.

Under-utilization of compression can be caused by either a handset or the server. Specifically, 60% of HTTP requests, whose responses account for 79% of the total HTTP response traffic[4], do not contain an **`Accept-Encoding`** header field, making it impossible for the server to transfer a compressed file. Compressing the responses using gzip yields a CR of 82% for the corresponding 79% of the HTTP response traffic. Also 26% of HTTP requests do have **`Accept-Encoding`** header fields but their responses are not compressed by the server. Compressing them reduces their HTTP response traffic volume by 10%.

Table 2 lists the top **`Content-Type`** strings appearing in HTTP responses (Column 1), their contribution to the overall HTTP traffic volume (Column 2), the fraction of bytes that are not compressed in the original data (Column 3), and their CR values (Column 4). For example, for all bytes in the original trace belonging to **`text/xml`** files, they are responsible for 5.23% of the total HTTP traffic volume, and 98.18% of such bytes belong to files that were not compressed in the original trace. By compressing those files, the transferred **`text/xml`** data size can be reduced to 5.23%*14.59%=0.76% of all (unoptimized) HTTP traffic volume. Table 2 indicates a bimodal distribution of CR values across content types. Compression is under-utilized in the original trace for most text files (html, xml, javascript, and css) accounting for 15% of all HTTP traffic. For each of such content types, 58% to 98% of the response data is not compressed. If compression is used, more than 70% of their bytes can be saved. In contrast, images, videos and most binary data already have compact file formats so further compression brings marginal benefits.

To understand the limits of the effectiveness of the file compression techniques, we combine all HTTP requests and responses for each of the 20 users into a single large file, run compression for each file, and then compute a CR value across all these 20 files. The gaps between these lower bounds (Columns 5 to 7 in Table 1) and their corresponding CRs of object-based compression (Columns 2 to 4) vary between 3.5% and 10.8%, depending on the compression technique. Also, HTTP/1.1 does not compress HTTP

[4] Unless otherwise specified, a percentage such as "x% of HTTP traffic" and "x% of all traffic" refers to the percentage of traffic in the original data *before* being optimized by RE techniques.

	p=1/4	p=1/8	p=1/16	p=1/32
n=512k	70.2%	71.9%	73.7%	75.3%
n=256k	71.8%	73.4%	75.2%	76.8%
n=128k	73.1%	74.7%	76.4%	78.0%
n=64k	74.3%	75.8%	77.5%	79.0%
n=512k (no loss)	69.3%	71.0%	72.8%	74.4%

Table 3. Applying the MODP algorithm on all traffic

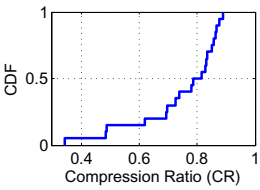

Fig. 1. CR distribution across users (caching+gzip+delta)

headers [11], which account for 5% of the total HTTP bytes in the trace. Compressing them reduces CR of HTTP traffic by about 1.4% (considered by Columns 2 to 4). This is performed by SPDY [3] that has been implemented in the Google Chrome browser.

Delta Encoding. Consists of two scenarios: a trivial case where the two versions are identical (*i.e.,* the delta is zero), and a non-trivial case where they are different. The trivial case is already handled by today's HTTP caching. Column 8 in Table 1 includes both cases while Column 9 in Table 1 only considers *additional* benefits brought by handling non-trivial cases using VCDIFF, a feature not widely deployed. We observe that doing so only slightly outperforms using only standard caching because trivial cases are much more prevalent than non-trivial cases. Specifically, 19.0% of HTTP bytes belong to cacheable files whose previous instances remain unchanged. Requests for these files can be served either by the local cache before expiration, or by a **`304 Not Modified`** response after expiration. In contrast, only 4.7% of HTTP bytes belong to files whose previous instances (with the same URL) differ. But for those 4.7% of HTTP bytes, VCDIFF does make them more compact than gzip does: VCDIFF achieves a CR value of 57.4%, while using gzip without leveraging similarities between the two versions yields a much higher CR of 72.4%.

Packet Stream Compression. Table 3 quantifies the effectiveness of MODP by changing two critical parameters n, the size of the packet cache in terms of the number of packets, and p, the sampling rate for fingerprint generation (§4). The overall CR is encouragingly good, between 70.2% and 79.0%. Exponentially decreasing p from 1/4 to 1/32 does not dramatically increase CR because the similarity between an input packet and a cached packet is often high so generating fingerprints less frequently can still yield a reasonably high matching rate. Decreasing n from 512k to 64k causes limited increase of CR as well due to the temporal locality of cache access [5].

Table 3 considers packet loss, which may hinder the MODP algorithm from functioning correctly. Consider a packet P that is lost after entering the ingress end's cache. The egress end thus cannot decode any subsequent packet that is compressed using the reference packet P. To address this issue, we assume that if the receiver cannot decode a packet, it immediately requests that the sender retransmit the lost reference packet(s) to synchronize the two packet caches [14].

We measured the overall retransmission rate of TCP traffic, which accounts for 98% of the overall traffic volume, to be 2.1% (1.8% for Wi-Fi and 2.2% for 3G). For all but the last row in Table 3, we conservatively treat all TCP packets that are later retransmitted as lost packets. The last row corresponds to a hypothetical scenario

Table 4. Jointly applying multiple RE techniques

	caching +gzip	caching +bzip2	caching +7-zip	caching +gzip +delta	caching +bzip2 +delta	caching +7-zip +delta	All * n=512k p=1/4	All * n=256k p=1/8	All * n=128k p=1/16	All * n=64k p=1/32
HTTP	71.4%	71.6%	70.4%	71.1%	71.3%	70.1%	-	-	-	-
All	75.5%	75.8%	74.7%	75.3%	75.5%	74.5%	68.1%	68.6%	69.2%	69.9%

* All = caching + gzip (lv 5) + delta + MODP (for all traffic). n and p are MODP parameters.

with no loss. In that ideal case, the CR decreases by about 1% due to the eliminated retransmission overhead of reference packets, implying that the impact of packet loss observed in the five-month dataset on CR is small.

5.3 Combining Multiple Approaches

We now apply multiple RE techniques together by following the order described in §5.1. The left seven columns in Table 4 indicate that jointly employing caching, file compression, and delta encoding is beneficial in that it reduces the CR to as low as 70.1% (for HTTP traffic) and 74.5% (for all traffic). Caching and file compression are complementary schemes: the former makes traffic due to multiple requests of the same file more efficient, while the latter improves the efficiency of a single file transfer.

Figure 1 plots the CR distribution (for all traffic) across the 20 users, assuming caching, gzip (lv 5), and delta encoding are jointly used. The CR for each user ranges from 34% to 89%, implying the heterogeneity of traffic generated by diverse users (§3). Clearly, the effectiveness of RE techniques depends on traffic content that differs across users, but the incurred bandwidth savings are unanimously non-trivial ($> 10\%$).

We then take a further step by applying MODP in addition to the three object-based RE techniques. By further looking at the right four columns in Table 4, we learn the additional CR reduction due to MODP is non-trivial (ranging from 5.4% to 7.2%) but is much smaller than the saving brought by using MODP alone (21.0% to 29.8% as depicted in Table 3). This implies that object-based RE techniques have already eliminated most redundancies for the HTTP traffic that dominates the trace. In fact, MODP further reduces the HTTP traffic volume by 6.2% to 7.8%, most of which comes from cross-file redundancy of non-cacheable files. In contrast, MODP results in much more reduction of CR for non-HTTP traffic, between 16.1% and 21.7%.

5.4 Performance

We measure the performance of each RE technique on a real server and a smartphone device. Our equipment includes a Dell PowerEdge server with an Intel Xeon E5620 quad-core CPU at 2.4 GHz and a Motorola Atrix 4G smartphone with a Tegra 2 dual-core CPU at 1 GHz. The server ran Ubuntu 11.04 and the phone used Android 2.2.

Two Macro-benchmarks were employed to evaluate the file compression and the packet stream compression technique, respectively. The *file benchmark* consisted of 1000 HTTP responses randomly sampled from the dataset. The *packet stream benchmark* was a 2GB packet trace generated by a random user. We produced five instances

Table 5. Throughput (in Mbps) of object-based RE techniques on the *File Benchmark*

	gzip (level 1 – 9)		bzip2 (level 1 – 9)		7-zip (level 1 – 9)		VCDIFF (δ 10%–90%)	
	comp	decomp	comp	decomp	comp	decomp	comp	decomp
Server	80–132	380–392	24–25	57–60	14–17	20–20	5.4–5.5	479–808
Phone	19–37	223–231	5.2–5.6	18–21	4.4–5.4	10–10	1.9–1.9	231–392

Table 6. Throughput (in Mbps) of MODP on the *Packet Stream Benchmark*

Compress	n=128k p=1/16	n=64k p=1/32	Decompress	n=128k p=1/16	n=64k p=1/32
Server	19	41	Server	320	348
Phone	4.2	8.9	Phone	40	41

of this benchmark, all yielding very similar performance results. We report the results for one instance.

We measured the in-memory compression/decompression time (excluding disk I/O) for the two benchmarks on both the server and the phone, using binaries compiled from the same source code. Table 5 shows the results for the *file benchmark*. Each file was compressed (decompressed) separately and the measured throughput is the total file size divided by the sum of the processing time of all files. For VCDIFF, we artificially generated a previous version of each file by randomly changing its content by a fixed percentage of δ. Table 6 summarizes the *packet stream benchmark* results. We measured the processing time of the second-half data of the 2GB packet trace, whose first-half data of 1GB was used to fill the packet cache and the signature table (for compression). Changing this 1GB to 0.5GB or 1.5GB has negligible impact on the results.

In Table 5 and Table 6, the throughput was estimated in an extreme case where the data was fed into the compressor/decompressor as fast as possible without any interruption. We repeated each test 10 times and measured the average running time, from which we derived the throughput value. The standard deviation of the running time across 10 runs was always less than 2% of the average.

The benchmark results deliver several observations. *(i)* As expected, compression is slower than decompression. But compressed files can be cached by servers to avoid having to repeat the compression for each incoming request for the same file. *(ii)* gzip is much faster than the more sophisticated bzip2 and 7-zip (for both compression and decompression) while its achieved CR is only slightly higher for small files from which most benefits of compression come (§5.2). *(iii)* VCDIFF is more expensive than all three file compression techniques, because it involves heavy computation for comparing *two* versions of a file. *(iv)* For gzip, bzip2, VCDIFF, and MODP, their low decompression overheads make it possible to keep up with a high data rate (*e.g.,* 15Mbps), incurring very small impact on page processing/rendering time on a handset. *(v)* MODP is quite efficient for small n and p. Exponentially increasing n and p worsens the performance (not shown in Table 6), and doing so provides little additional traffic savings when object-based RE is performed beforehand (Table 4). The performance could be further improved by enhancements of MODP, such as MAXP and SAMPLEBYTE [5].

6 Summary and Recommendations

We summarize our main findings and recommendations as follows.

1. Under-utilization of compression contributes to significant redundancy, *i.e.,* 15% of the overall traffic volume for our trace. It is imperative that the content providers utilize the compression feature supported by all mainstream Web servers. Handsets should use the **`Accept-Encoding`** header field, which appeared in only 40% of HTTP requests within the dataset, to enable compression.

2. Considering both effectiveness and performance, gzip is the best compression approach for small files from which most benefits of compression come (yet the traffic volume contribution of such small files is considerable, see §5.2). Applying delta encoding on non-trivial cases (§5.2) brings limited benefits, because less than 5% of HTTP bytes belong to files with a different previous version. Except for 7-zip, decompression performance is generally not an issue on mobile devices, leading to very small impact on page processing/rendering time.

3. Special emphasis should be put on html, xml, javascript, and css files. They account for 15% of the HTTP traffic in the dataset (17% reported in [12] for handheld traffic in campus Wi-Fi networks), but are usually (58% to 98% bytewise) not compressed. More than 70% of their bytes can be saved using compression.

4. Using packet stream compression alone, represented by the MODP algorithm, effectively reduces the traffic volume by up to 30%. If object-based RE techniques, which are already part of the HTTP specification, are applied beforehand, the benefit of MODP decreases but is still non-trivial, *i.e.,* a reduction of 5.4% to 7.2% of all traffic. In that case, the impact of the aggressiveness level on CR is much less significant. We therefore recommend that MODP be deployed in a less aggressive manner, *e.g.,* $n \leq 64$k packets and $p \leq 1/16$ for downlink. This achieves most of the bandwidth savings possible from MODP while limiting the performance overhead for compression as well as decompression. Note that packet stream compression provides benefits despite idiosyncrasies in application implementations.

5. A judicious combination of all RE techniques achieves an overall reduction of the smartphone traffic studied in this measurement by more than 30% with acceptable computational overhead. This is even more interesting and somewhat surprising given that a major fraction of the traffic is video, audio, or image that are already compressed. In comparison, caching by itself only saves 17% of the overall traffic (§5.2).

Acknowledgements. This work is partly funded by NSF grants CNS-1059372, CNS-1050157, CNS-1039657 and Navy grant N00014-09-1-0705. We thank Emir Halepovic and the shepherd Marios Iliofotou for their valuable comments on the paper. We would also like to thank anonymous reviewers whose comments improved the final version.

References

1. Invest in Cell Phone Infrastructure for Growth in 2010 (2010), `http://pennysleuth.com/invest-in-cell-phone-infrastructure-for-growth-in-2010/`
2. Packet Data Convergence Protocol (PDCP) specification. 3GPP TS 25.323

3. SPDY: An experimental protocol for faster web, http:// dev. chromium. org/ spdy
4. Cisco Visual Networking Index (2012), http:// newsroom. cisco. com/ press-release-content? type=webcontent& articleId=668380
5. Aggarwal, B., Akella, A., Anand, A., Balachandran, A., Chitnis, P., Muthukrishnan, C., Ramjee, R., Varghese, G.: EndRE: An End-System Redundancy Elimination Service for Enterprises. In: NSDI (2010)
6. Anand, A., Gupta, A., Akella, A., Seshan, S., Shenker, S.: Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. In: SIGCOMM (2008)
7. Anand, A., Sekar, V., Akella, A.: SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination. In: SIGCOMM (2009)
8. Anand, A., Muthukrishnan, C., Ramjee, R.: Redundancy in Network Traffic: Findings and Implications. In: SIGMETRICS (2009)
9. Deutsch, P.: DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (1996)
10. Erman, J., Gerber, A., Hajiaghayi, M., Pei, D., Sen, S., Spatscheck, O.: To Cache or not to Cache: The 3G case. IEEE Internet Computing (2011)
11. Fielding, R., Gettys, J., Mogul, J., Masinter, H.F.L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol - HTTP/1.1. RFC 2616 (1999)
12. Gember, A., Anand, A., Akella, A.: A Comparative Study of Handheld and Non-handheld Traffic in Campus Wi-Fi Networks. In: Spring, N., Riley, G.F. (eds.) PAM 2011. LNCS, vol. 6579, pp. 173–183. Springer, Heidelberg (2011)
13. Korn, D., MacDonald, J., Mogul, J., Vo, K.: The VCDIFF Generic Differencing and Compression Data Format. RFC 3284 (2002)
14. Lumezanu, C., Guo, K., Spring, N., Bhattacharjee, B.: The Effect of Packet Loss on Redundancy Elimination in Cellular Wireless Networks. In: IMC (2010)
15. Mogul, J., Douglis, F., Feldmann, A., Krishnamurthy, B.: Potential benefits of delta encoding and data compression for HTTP. In: SIGCOMM (1997)
16. Mogul, J., Krishnamurthy, B., Douglis, F., Feldmann, A., Goland, Y., van Hoff, A., Hellerstein, D.: Delta encoding in HTTP. RFC 3229 (2002)
17. Qian, F., Quah, K.S., Huang, J., Erman, J., Gerber, A., Mao, Z.M., Sen, S., Spatscheck, O.: Web Caching on Smartphones: Ideal vs. Reality. In: Mobisys (2012)
18. Sanadhya, S., Sivakumar, R., Kim, K.H., Congdon, P., Lakshmanan, S., Singh, J.P.: Asymmetric Caching: Improved Network Deduplication for Mobile Devices. In: Mobicom (2012)
19. Spring, N.T., Wetherall, D.: A Protocol-Independent Technique for Eliminating Redundant Network Traffic. In: SIGCOMM (2000)
20. Zohar, E., Cidon, I., Mokryn, O.O.: The Power of Prediction: Cloud Bandwidth and Cost Reduction. In: SIGCOMM (2011)