Comparison of Caching Strategies in Modern Cellular Backhaul Networks

Shinae Woo
KAIST
shinae@ndsl.kaist.edu

Eunyoung Jeong
KAIST
notav@ndsl.kaist.edu

Shinjo Park
KAIST
pshinjo@ndsl.kaist.edu

Jongmin Lee
SK Telecom, Inc.
jminlee@sk.com

Sunghwan Ihm*
Princeton University
sihm@cs.princeton.edu

KyoungSoo Park
KAIST
kyoungsoo@ee.kaist.ac.kr

ABSTRACT

Recent popularity of smartphones drives rapid growth in the demand for cellular network bandwidth. Unfortunately, due to the *centralized* architecture of cellular networks, increasing the physical backhaul bandwidth is challenging. While content caching in cellular networks could be beneficial, little is known about the traffic characteristics to devise a highly-effective caching strategy.

In this work, we provide insight into flow and content-level characteristics of modern 3G traffic at a large cellular ISP in South Korea. We first develop a scalable deep *flow* inspection (DFI) system that can manage hundreds of thousands of concurrent TCP flows on a commodity multicore server. Our DFI system collects various HTTP/TCP-level statistics and produces logs for analyzing the effectiveness of conventional Web caching, prefix-based Web caching, and TCP-level redundancy elimination (RE) without a single packet drop at a 10 Gbps link. Our week-long measurements of over 370 TBs of the 3G traffic reveal that standard Web caching can reduce download bandwidth consumption up to 27.1% while simple TCP-level RE can save the bandwidth consumption up to 42.0% with a cache of 512 GB of RAM. We also find that applying TCP-level RE on the largest 9.4% flows eliminates 68.4% of the total redundancy. Most of the redundancy (52.1%∼58.9%) comes from serving the same HTTP objects while the contribution by aliased URLs is up to 38.9%.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network monitoring; C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Measurement, Performance

Keywords

Deep Flow Inspection, Cellular Networks, Caching, Redundancy Elimination

*Current affiliation: *Google, Inc.*

MobiSys'13, June 25-28, 2013, Taipei, Taiwan

1. INTRODUCTION

Recent popularity of smartphones and tablet PCs drives the rapid growth of the mobile data in cellular networks. The global cellular data traffic is predicted to increase by 18 times in the next five years, which is 3 times faster than the growth rate of the fixed IP networks [20]. While many cellular ISPs are planning to invest tens of billions of dollars to prepare their infrastructure [43], it is unclear if the physical capacity will meet the future demand. As a status quo, cellular ISPs curb the demand using pay-per-usage service plans or by throttling access speed [45].

With increasing deployments of high-speed base stations and femtocells [40], the wireless channel bandwidth to radio access networks (RANs) has improved over time. However, cellular networks typically have a *centralized* architecture where all user traffic has to traverse one of a few gateways at core networks (CNs) before reaching the wired Internet [16, 22]. Centralization is necessary for tracking the location of mobile devices, billing, and handovers, but high traffic concentration at CNs could lead to potential congestion as the number of base stations rapidly grows.

Caching popular contents in the backhaul networks is a cost-effective solution that could increase the effective bandwidth. It is especially attractive since deploying IP-based caching equipment is getting easier in packet-switched 3G/4G networks [1]. While there have been many works on the effectiveness of Web caching in cellular backhaul links [22, 23] and at mobile devices [37, 38], relatively little is known about the effectiveness of protocol-independent redundancy elimination (RE) [13, 28, 42] or prefix-based Web caching [15, 34, 35] in the cellular networks. Prefix-based Web caching can address the redundancy from aliased Web objects in addition to regular objects, which is increasingly popular for caching aliased multimedia contents in the Internet. In this work, we compare the efficacy of these caching strategies. Specifically, we explore (a) the characteristics of modern cellular traffic that can impact the caching strategies, (b) the fraction of redundancy that can be addressed by each caching scheme and the source of the redundancy, and (c) system overheads of caching strategies in terms of cache size and concurrent flow counts.

To find the answers to these questions, we analyze the TCP flow and content-level characteristics of commercial 3G traffic at a 10 Gbps backhaul link at one of the largest ISPs in South Korea for one week. We choose to do online analysis as dumping packet contents at high speed to disk for offline analysis is prohibitively expensive. We develop Monbot, a scalable deep *flow* inspection (DFI) system, that analyzes the network-level behavior and content-level redundancy of *every* TCP flow in real time. One challenge in Monbot lies in how it handles millions of packets per second, maintaining hundreds of thousands of concurrent flows. We address

this problem by balancing the flow management and content analysis loads among multiple CPU cores with *symmetric* receive-side scaling (symmetric RSS). Similar to RSS [33], symmetric RSS distributes the incoming packets to multiple CPU cores while ensuring the packet order in each flow. But unlike RSS, it maps the packets in the same TCP connection (*i.e.*, packets in both upstream and downstream flows) to the same CPU core, obviating the need to share the TCP context across different cores. This promises high scalability on a multicore system by avoiding cross-core synchronization or shared lock contention. Monbot also implements per-core memory management for flow buffers and timeout-based buffer recycling, which effectively reduces the memory footprint for a large number of concurrent flows. With these techniques, we could monitor 375.5 TBs of packets (8.3 billion flows) without a single packet drop on a 12-core commodity server.

Our contributions are summarized as follows. First, we show the design and implementation of Monbot that scalably manages 100Ks of active TCP flows at a 10 Gbps link. Monbot can be easily extended to work as high-performance middleboxes such as stateful firewalls, intrusion detection systems, or transparent RE nodes. Second, we develop symmetric RSS that evenly distributes the packets by their TCP connection at line rate. Symmetric RSS can be applied to any commodity hardware-based system that needs to maintain TCP flows in the high-speed networks. Third, we analyze the characteristics of commercial 3G traffic both in the network and content level using Monbot. Our findings are summarized as follows.

- Commercial 3G traffic shows heavier dominance of HTTP and TCP than in the wired Internet. 95.7% of the downlink traffic uses TCP, and the HTTP traffic constitutes 74.6% of all downstream bytes.
- Most TCP flows are small, but the majority of the transferred bytes belong to large flows. That is, only 9.4% of the flows are larger than 32 KB, but these flows contribute to 93.7% of the transferred bytes. Monbot processes as many as 1.3 million new TCP connections per minute, and up to 270K concurrent flows at any given time.
- The 3G traffic usage exhibits a strong diurnal pattern as in the wired Internet. We see local bandwidth peaks in the morning rush hour (8 - 9 am) and lunchtime (12 - 1 pm) every working day. The daily bandwidth peak happens around 11 pm - 12 am, mostly driven by video contents.
- With infinite cache, we find that up to 59.4% of the traffic is redundant with TCP-level redundancy elimination (TCP-RE). While standard Web caching achieves only 21.0∼27.1% of bandwidth savings despite with infinite cache, TCP-RE achieves 26.9∼42.0% bandwidth savings with only 512 GB of memory cache. Prefix-based Web caching sits in between, producing 22.4∼34.0% of bandwidth savings with infinite cache.
- Unlike for enterprise traffic [13], TCP-RE for the 3G traffic effectively lowers the bandwidth peaks in the morning time and lunchtime. It also provides significant bandwidth savings (30.7∼36.9%) during the peak usage in the night time.
- The chunk popularity follows strong Zipf distribution. The LRU cache replacement is slightly better than FIFO in terms of bandwidth savings.
- 52.1%∼58.9% of the redundancy is from serving the same HTTP objects. Aliased objects contribute to 30.4%∼38.9% of the redundancy.

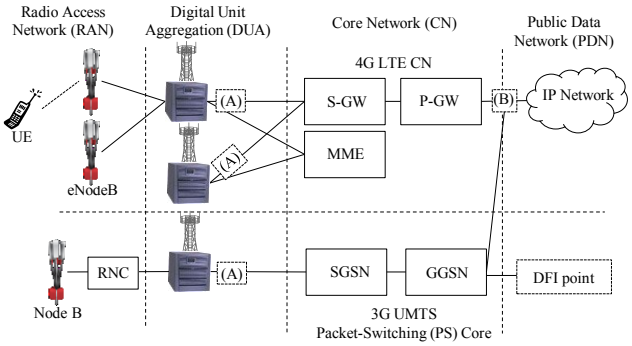


Figure 1: Overall 3G/4G cellular network architecture. (A) and (B) are candidate locations for deploying caching middleboxes.

To the best of our knowledge, we analyze and report the largest amount of the 3G cellular traffic on commodity hardware. We believe our findings are useful in designing a middlebox-based TCP-RE system with tunable parameters that control the level of bandwidth savings and system overheads.

2. BACKGROUND

In this section, we provide the background on the structure of cellular networks, and popular redundancy elimination approaches.

2.1 Modern Cellular Networks

Figure 1 shows the overall architecture of 3G Universal Mobile Telecommunications System (UMTS) and 4G Long Term Evolution (LTE) cellular networks. A radio access network (RAN) consists of a set of base stations (called Node B in UMTS, and eNodeB in LTE) and User Equipments (UEs). A UE communicates wirelessly with a base station and the data is forwarded to a regional digital unit aggregation point (DUA). There are tens to hundreds of DUAs per ISP in South Korea that are connected to a cellular core network. A S-gateway (S-GW) in LTE acts as a mobility anchor for UE handovers between eNodeBs (or between LTE and other 3GPP networks), and forwards packets to P-gateway (P-GW), which serves as a gateway router to IP networks (called public data networks (PDNs)). A mobility management entity (MME) is in charge of many control-plane tasks such as tracking and paging UEs, performing handovers between LTE core networks (CNs), authenticating users, etc. Similar functionality is found in 3G UMTS where a Serving GPRS Support Node (SGSN) takes the role of S-GW and MME, and a Gateway GPRS Support Node (GGSN) acts as a P-GW in LTE. Since 3G/4G networks use IP packets via GPRS Tunneling Protocol (GTP) throughout its packet-switched domain, it is easy to deploy IP-based caching/monitoring systems.

2.2 Available Caching Techniques

As shown in Figure 1, cellular networks have a *centralized* architecture where all packets are concentrated at a few P-GWs or GGSNs in a country. With the improvement of RAN technologies and wider deployments of cell towers, it is predicted that the backhaul links to CNs could become the next bottleneck[6, 9]. If we can reduce redundant data transfers between DUAs and a CN, we can increase the effective bandwidth in the backhaul networks. We consider Web caching and protocol-independent RE as possible caching solutions here.

Web caching is a popular technique that suppresses redundant Web transfers. Since a large portion of the cellular traffic uses

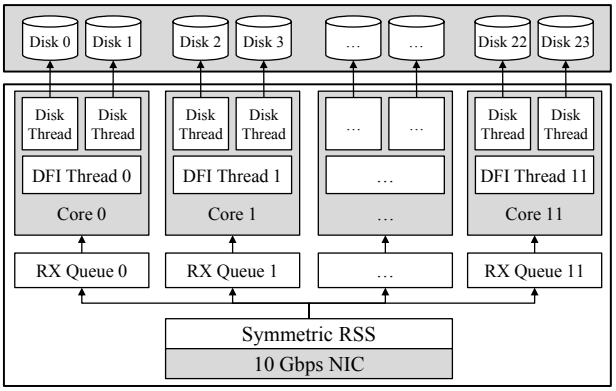


Figure 2: Overall architecture of Monbot

HTTP [22], we expect significant bandwidth savings as well as fast response if Web caches are deployed at DUAs ((A) in Figure 1), or near the gateways ((B) in Figure 1). However, it poses two problems. First, standard Web caching cannot suppress duplicate objects that are uncacheable or that have different URLs (*i.e.*, aliases). It cannot suppress the partial content overlap in similar objects as well. Second, it is challenging to handle UE handovers across DUAs while the content is being delivered. If a UE moves its attachment point to another DUA, content from a Web cache at one DUA should be re-routed to the new location, which is difficult since the route is centrally managed by MME or SGSN.

Prefix-based Web caching alleviates the first problem of Web caching by suppressing the aliased or duplicate objects that are uncacheable. A typical approach is to confirm an alias by comparing the hash of the first *N* bytes of an object (called `prefix key`), and the `content-length` fields. It assumes that two HTTP objects of the same size have the same content if their prefix keys match. While this potentially presents a correctness issue especially if *N* is small, it is already widely used in the wired Internet to serve aliased multimedia contents (e.g., aliased videos) [15, 34, 35]. Unfortunately, little is known about its effectiveness and false positives, which we study in Section 6.2.

Protocol-independent RE can address both of the problems of Web caching at the cost of high engineering complexity and computational overheads. It assumes two middleboxes deployed at (A) and (B) in Figure 1. (B) divides incoming content into small chunks, and sends a stream of chunk hashes to (A). (A) reconstructs the original content with those hashes, and delivers it to UEs. If any chunks are cache misses at (A), (A) fetches the original content from (B), and caches them for future references.

There are many variants of RE, but we use TCP-RE to compare the effectiveness with other caching strategies. Unlike packet-level RE, TCP-RE works over the content of each TCP flow. We reassemble the segments in the same TCP flow and divide them into smaller chunks (either fixed-sized or variable-sized) that serve as the unit of caching. While TCP-RE has additional overhead to manage TCP flows, it allows scalable cache management as well as flow-based caching policies. We discuss packet-based RE and TCP-based RE approaches in more detail in Section 6.4.

3. SCALABLE ANALYSIS OF REDUNDANT TRAFFIC

In this section, we describe the design and implementation of Monbot, a high-performance DFI system on commodity hardware. At a high-level, Monbot produces three types of log data in real time for later analysis – (1) TCP flow statistics, (2) HTTP request / response statistics, and (3) SHA-1 hashes of content chunks. We primarily focus on the TCP and HTTP traffic since they take up the majority of the total traffic volume.

While there are many high-end DPI solutions that produce various protocol-level traffic statistics or dump packets to disk [2, 4, 8], we do not know of any system that analyzes the content-level redundancy in the high-speed 10 Gbps network. Dumping packets to disk is not an option since our traffic volume for a week would require more than 120 3 TB disks, not to mention the enormous time for offline analysis. Instead, we focus on online analysis, which requires high-performance packet capture, scalable flow management and content hash calculation, and parallel disk I/O for writing the log data. Monbot supports this goal on a pure commodity server without any dedicated packet capture cards such as [3, 7].

One challenge of Monbot is to process a large amount of concurrent flows without a packet drop. Indeed, our measurement shows that the system should handle 1.4 million packets per second and 270K concurrent TCP connections at peak. To cope with this scale, Monbot exploits processing parallelism in modern multicore systems, making each core handle a partition of flows in parallel. To avoid expensive inter-core communication, we also develop symmetric RSS that assigns both upstream and downstream packets that belong to the same TCP connection to the same core. Our current prototype has two 3.47 GHz hexacore Xeon X5690 CPUs in dual-NUMA nodes with 144 GB of RAM and a dual-port 10 Gbps NIC with the Intel 82599 chipset, and has 24 3TB SATA disks for storing log data.

3.1 Parallel Flow Monitoring Architecture

Figure 2 shows the overall architecture of Monbot. Monbot transparently receives IP packets from the monitored 10 Gbps link (*e.g.*, via port-mirroring), and distributes them to CPU cores with symmetric RSS. For high-speed packet reception (RX), we use PacketShader I/O engine (PSIO) [26]. PSIO is known for achieving multi-10 Gbps packet I/O even for 64B packets by batch processing of RX packets and efficient CPU cache usage.

The central goal of the Monbot design is to achieve high core scalability. Given that the majority of the TCP flows are small (in Section 5), it is important to efficiently handle small TCP control packets in the same CPU core without any lock contention [27, 36]. Each CPU core in Monbot runs one DFI thread and two disk threads pinned to it. The DFI thread is responsible for packet RX with PSIO, and for producing TCP/HTTP flow statistics and content hashes. Each disk thread buffers the logs passed from the DFI thread, and writes them to its dedicated disk in batches. This way, all 12 CPU cores and 24 disks are utilized in parallel completely independent of each other.

To efficiently manage buffers for a large number of concurrent flows while dealing with packet reordering from packet losses or retransmissions[1], Monbot also implements per-core memory pools with different buffer sizes. Dynamically adjusting buffer sizes with `realloc()` is prohibitively expensive as it incurs lock contention across cores, and having a uniform buffer size risks either packet losses or out of memory. Instead, each DFI thread pre-allocates $\frac{1}{N}$-th of the total memory where *N* is the total number of cores, and creates a set of memory pools with different buffer sizes (*e.g.*, 32 KB, 64 KB, ..., 2 MB). A new flow is assigned to the smallest buffer, and moves to a larger buffer as more space is needed. Based

[1]When a flow with a large receive window (*e.g.*, we see as large as 16 MB) sees burst packet losses, a large gap in the buffer is inevitable.

Algorithm 1 RSS Hash Computation Algorithm

```
function RSS_HASH(INPUT, RSK)
  u_int32 ret = 0;
  for each bit b in INPUT
    if (b == 1) then ret = ret xor (left-most 32 bits of RSK);
    shift RSK left by 1 bit;
  end for
  return ret;
end function
```

INPUT Field	INPUT Bit Range	RSK Bit Range
Source IP (32 bits)	1 .. 32	1 .. 63
Destination IP (32 bits)	33 .. 64	33 .. 95
Source port (16 bits)	65 .. 80	65 .. 111
Destination port (16 bits)	81 .. 96	81 .. 127
Protocol (8 bits)	97 .. 104	97 .. 135

Table 1: RSK bit ranges that affect the portions of an IPv4 header for RSS hash calculation

```
0x6d 0x5a 0x6d 0x5b 0x6d 0x5a 0x6d 0x5b
0x6d 0x5a 0x6d 0x5a 0x6d 0x5a 0x6d 0x5a
0xc6 ...
```

Table 2: Sample RSK that satisfies Equation (3)

on our observation that 90.6% of the total TCP flows are smaller than 32 KB and short-lived, we assign 90% of the total memory to 32 KB buffers, and proportionally decrease the size of the memory pools for larger buffer sizes. This way, Monbot can process 40K concurrent flows per DFI thread, handling 480K concurrent TCP flows with 12 cores.

3.2 Symmetric Receive-Side Scaling

RSS is a modern NIC feature that balances the packet processing load across multiple CPU cores. It distributes incoming packets into multiple hardware receive queues (RX queues) that are mapped to each CPU core by running the Toeplitz hash function [30] on the five tuples of IP packet headers (src/dest IPs, ports, and a protocol). This way, packets in the same flow are accessed exclusively by a core without lock contention.

However, RSS is not suitable for high-performance DFI middleboxes that monitor TCP connections in *both* directions, as the packets in the same connection may end up in two different RX queues, one for upstream packets and another for downstream packets, and accessing them requires acquiring locks. While one can re-run hashing and distribute packets in software [5, 44], it not only wastes CPU cycles but also loses the benefit of hardware-supported RSS.

To overcome this limitation, we develop symmetric RSS that maps the packets in the same connection into the same RX queue regardless of their directions, by leveraging the existing RSS feature. More importantly, symmetric RSS does not modify the RSS algorithm itself, thus it can still distribute the packets at line rate. The basic idea is to produce the same RSS hash value even if the src/dest IPs and ports are swapped.

Algorithm 1 shows the pseudo code of the Toeplitz hash function that takes INPUT and RSK as inputs. INPUT is typically five tuples of the IP header, and RSK is a 320-bits random secret value that gets mixed with INPUT. Since the value of RSK can be configured when activating NICs, we can craft the value of RSK that satisfies the following condition where s and d represent source and destination (*e.g.*, sIP is a source IP), and p represents a one-byte protocol field (*e.g.*, TCP or UDP).

$$\begin{aligned} &RSS\_Hash(sIP : dIP : sPort : dPort : p, RSK) \\ &= RSS\_Hash(dIP : sIP : dPort : sPort : p, RSK) \end{aligned} \quad (1)$$

We can further limit the scope of Equation (1) with the following observation, which derives Table 1 that shows RSK bit ranges affecting each IP header field in RSS hash calculation.

OBSERVATION 1. *For calculation of the n^{th} bit of INPUT, only the n^{th} to $(n+31)^{th}$ bit range of RSK is used.*

Plugging Table 1 into Equation (1) yields the following condition where $\text{RSK}_{A..B}$ represents a bit range of RSK from the A^{th} bit to the B^{th} bit.

$$RSK_{1..63} = RSK_{33..95}, \quad RSK_{65..111} = RSK_{81..127} \quad (2)$$

By removing the overlapping regions from Equation (2), we obtain the final condition as follows.

$$\begin{aligned} &i)\ RSK_{1..15} = RSK_{17..31} = RSK_{33..47} = RSK_{49..63} \\ &\quad = RSK_{65..79} = RSK_{81..95} = RSK_{97..111} = RSK_{113..127} \\ &ii)\ RSK_{16} = RSK_{48} = RSK_{80} = RSK_{96} = RSK_{112} \\ &iii)\ RSK_{32} = RSK_{64} \end{aligned} \quad (3)$$

One such example of RSK for IPv4 packets is shown in Table 2 where each byte is alternating except for the 4^{th}, 8^{th}, and 17^{th} bytes. This is because Equation (3) requires that each group of 16 bits should be identical except for the 32^{nd} and the 64^{th} bit positions. Similarly, we can further tweak this RSK to be used for both IPv4 and IPv6 packets with the following observation.

OBSERVATION 2. *For calculation of the n^{th} byte of INPUT, only the n^{th} to $(n+4)^{th}$ byte range of RSK is used.*

This implies that if every odd byte of RSK has the same value, and so does every even byte, we can ensure the byte-level symmetry in RSS hash calculation.[2] For example, we only need to replace `0x5b` with `0x5a` in Table 2 to support IPv6 packets.

Evaluation. To confirm if symmetric RSS evenly balances TCP connections, we analyze the number of flows handled by each core on Monbot during the week-long measurement. Figure 3 presents the maximum deviation, which is the maximum difference in the number of flows processed by each core divided by the per-minute average. We find that both algorithms behave almost the same. In fact, as long as the IPs and ports are randomly distributed, we can prove that the level of load balancing in symmetric RSS is not worse than that in the original RSS [46].

The maximum deviations look somewhat high (9.5%) in both algorithms because the distribution is skewed since the number of CPU cores (12) is not a power of two. In fact, the first eight cores receive about 9% more flows than the remaining four cores. The deviation among the first 8 cores is less than 0.2% and so is for the remaining 4 cores. Our simulation on 16 CPU cores confirms that symmetric RSS bounds the deviations within 0.2% across all cores.

One concern with symmetric RSS is that one might craft an attack such that all traffic is veered onto one or a few CPU cores. While more thorough cryptanalysis is required, we think that brute-force attacks are non-trivial since trying all 24-bit key space for RSK is

[2]More details can be found in our technical report [46].

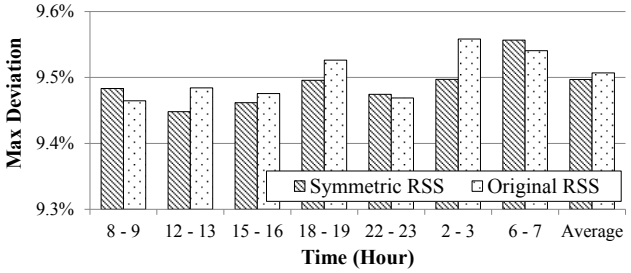


Figure 3: Maximum deviation in the number of flows per core

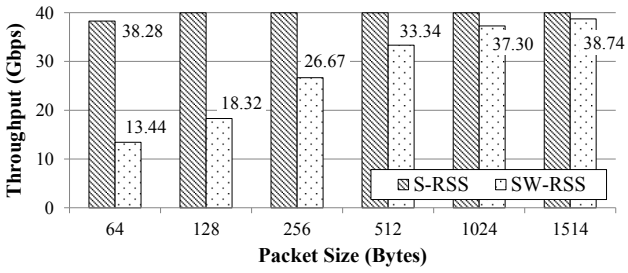


Figure 4: Packet RX performance of PSIO+S-RSS (S-RSS) vs. PF_RING+S/W hashing (SW-RSS)

time consuming and the attacker may not know how many cores are employed for the system. We leave it as our future work.

Next, Figure 4 compares the packet RX performances of PSIO with symmetric RSS (S-RSS), and PF_RING 5.5.1 with software hashing (SW-RSS) that distributes the packets in the same connection to the same thread [5]. PF_RING uses RSS and distributes the packets to CPU threads by zero copy. Both libraries support interrupt and polling modes, and we use interrupt mode here. [3]

To gauge the maximum performance, we add an extra 10 Gbps NIC to the machine so that its maximum RX rate reaches 40 Gbps. We use our own packet generator that blasts packets at 40 Gbps regardless of packet size [26], and use only 8 CPU cores to avoid RSS load imbalance. IPs and port numbers are randomly populated in a packet, and for each packet, we send its corresponding ACK packet by swapping IPs and port numbers. Except for 64B packets, S-RSS achieves a 40 Gbps line rate for all packet sizes. In contrast, SW-RSS shows 1.5x∼2.8x lower performance than S-RSS at 64B∼256B packets, although its performance is close to 40 Gbps at larger sizes.

4. DATASET

This section describes our measurement environment and the limitations in our dataset.

Measurement Environment. We place Monbot at a core network switch on GGSN in Seoul (as shown in Figure 1), which serves as a gateway to IP networks. This point of presence (PoP) covers the 3G traffic from half of all regions in South Korea served by the ISP, which has 12.5 million 3G subscribers. Monbot monitored all IP traffic at a 10 Gbps link from 12 pm on July 7th (Saturday) to 2 pm on July 14th (Sunday) in 2012. During the period, it logged 8.3 billion TCP connections from 1.8 million unique client IPs and 4.5 million unique server IPs, which amounts to 370 TBs in volume or 590 billion packets. UEs use the shared private IP space (*e.g.*, 10.*.*.*) allocated by DHCP, and their IPs are NAT-translated to public IPs after our measurement point while the server-side IPs are seen as public at Monbot.

Monbot stores a log record for each TCP connection and HTTP request. A TCP connection log includes source and destination IPs and ports, start and end timestamps, amount of transferred bytes, connection setup RTT, etc. A HTTP log includes a request URL, host name, start and end timestamps of each request, `content-type`, `content-length`, `user-agent`, and other caching-related header fields. It also records SHA1 hashes of 8 different prefix sizes of the response body, from 1 KB to 1 MB, as well as the hash of the entire object to gauge the effectiveness of prefix-based Web caching. Furthermore, Monbot handles HTTP pipelining and persistent connections, and stores each HTTP session log.

[3]PF_RING polling mode did not improve performance.

For content-level analysis, Monbot logs SHA1 hashes of fixed 4 KB content chunks for all of the downlink TCP traffic, except for HTTP response headers. We exclude HTTP response headers since some fields (*e.g.*, `Date`) are dynamically regenerated regardless of content. For any chunk / content that is smaller than 4 KB, we hash the entire chunk / content. The total size of the logs is 1 TB for TCP session logs, 2.8 TB for HTTP logs, and 11.4 TB for the content chunk hashes.

Limitations. We summarize the limitations in our dataset here. First, the 10 Gbps link being monitored is one of the flow-level load-balanced links by equal-cost multi-path (ECMP). Due to this setup, our dataset cannot capture all flows from a specific UE, but all packets in the same connection are monitored. Also, we do not have UE identifiers such as IMSI for client-side IPs, and a private IP can be reused by another device. When we need to identify a client, we use a heuristic that compares the IPs and the OS versions in the HTTP `user-agent` field. This limitation, however, is only subject to false positives in Figure 18(b), and all other findings remain intact.

Second, while we did not see any packet loss at the NIC of Monbot, we observe packet losses from switch-based mirroring. We confirm this by seeing client-side ACKs for the packets that are not captured by Monbot. We skip the TCP buffer region with a hole for SHA1 hash calculation, and ignore the potential redundancy for prefix-based Web caching when the incomplete region happens within the prefix range. The skipped bytes due to incomplete regions constitutes about 1.92% of the downlink traffic.

Finally, we use 4 KB fixed-sized chunking, which may miss some redundancy that is otherwise observed with a smaller chunk size or variable chunking methods. This choice was inevitable given the limitation in the available storage space and CPU cycles. Since our primary focus is to find if we can suppress a reasonable level of traffic by a software-based RE system on commodity hardware for 10+ Gbps environments, we believe our findings are still useful for practical RE systems design.

5. FLOW-LEVEL CHARACTERISTICS

In this section, we analyze TCP and application-level characteristics of our traffic, and also discuss the performance requirements and trade-offs of flow management overhead, latency, and throughput, which shapes the design of caching middleboxes.

Protocol and Application Mix. We first analyze the overall protocol composition in Table 3. As observed in other mobile networks, the downlink traffic volume is much larger (13x) than the uplink traffic volume [29]. TCP accounts for 95.7% of the entire downlink traffic, implying most of the traffic is downloads of content from servers to mobile devices. The remaining traffic is

IP Version	Volume	Volume Ratio	# Packets	Protocol	Volume	Volume Ratio	# Packets	# Packet Ratio
IPv4 Downlink	343.5 TB	91.5%	341,920,148,702	TCP	328.7 TB	95.7%	318,498,981,294	93.2%
IPv4 Uplink	25.6 TB	6.8%	243,711,521,675	UDP	14.1 TB	4.1%	21,543,090,962	6.3%
Others	6.4 TB	1.7%	20,140,919,576	ICMP	6.5 GB	-	71,621,981	0.02%

Port	Application	Volume (TB)	Volume Ratio(%)	# Flows Ratio(%)	Port	Volume (TB)	Volume Ratio(%)	# Flows Ratio(%)
80	HTTP	256.2	74.6	78.1	≥ 1024	59.6	17.4	10.8
443	HTTPS	10.2	3.0	10.4	17600	9.7	3.6	0.01
110	POP3	0.37	0.14	0.04	8405	3.8	1.4	0.01
995	POP3	0.16	0.06	0.03	9000	3.5	1.3	0.02
993	IMAP	0.30	0.11	0.20	8012	2.3	0.9	0.00
554	RTSP	0.17	0.06	0.01	8008	2.3	0.8	0.00

# Uniq Server IPs / 7 days	4,502,812
# Uniq client IPs / 7 days	1,807,499
# Uniq server IPs / hour	119,460
# Uniq client IPs / hour	922,631
# TCP flows	8,307,724,934
# TCP concurrent flows	174,322
Payload unmergeable ratio	1.92%
# Total 4K chunks	87,330,807,203

Content-type	Average Size	# Flows	Volume	Volume (%)	Cacheability (# Objects)	Cacheability (Volume)	Web Cache (Volume)	Prefix Cache (Volume)	TCP-RE (Volume)
HTTP Total	34.5 KB	7,738,867,184	248.3 TB	100.0%	53.9%	40.7%	23.7%	27.7%	32.7%
Video	1982 KB	45,457,010	83.9 TB	33.8%	19.8%	35.9%	18.7%	20.0%	28.4%
Image	19.9 KB	3,748,164,242	69.4 TB	28.0%	83.4%	93.1%	66.8%	67.5%	53.1%
App	45.8 KB	1,413,098,511	56.3 TB	22.7%	22.2%	62.2%	21.9%	28.3%	61.6%
Text	11.0 KB	2,047,840,830	20.9 TB	8.5%	30.4%	44.0%	33.7%	37.6%	43.4%
Audio	1073 KB	16,707,509	16.8 TB	6.8%	12.6%	18.3%	7.0%	32.4%	39.5%
Etc	2.60 KB	467,599,082	1.1 TB	0.4%	5.2%	18.4%	6.3%	6.7%	20.2%

Table 3: Overall statistics of the 3G traffic trace

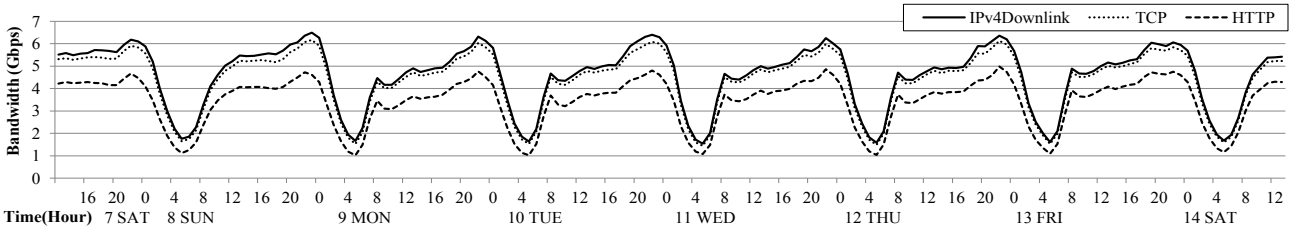


Figure 5: 3G traffic volume for seven days (each point is a 1-hour average)

largely UDP (4.1%) and a tiny amount of ICMP and GRE (less than 10^{-5}%) traffic.

Table 3 shows the contributions by various application protocols identified by server-side port numbers. We observe that HTTP (74.6%) and HTTPS (3.0%) dominate the traffic. However, the dominance of HTTP in the cellular network is much heavier than in broadband (57.6 %) [31]. We suspect that this is because most smartphone applications use HTTP as its communication protocol [32]. We also see the other traffic such as POP3, IMAP, RTSP, SSH, FTP, and TELNET, but their individual contribution is less than 1%, with the aggregate volume of only 3.7%. Remaining applications with ports larger than 1023 account for non-negligible 17.4% of the traffic. Further investigation shows that the top-ranked port (17600, 3.6%) is used for serving local P2P live video content [10]. We also find that other ports are used only by specific IP subnet ranges, implying that they are mainly custom protocols.

Figure 5 shows the hourly bandwidth consumption over the measurement period. We observe a clear diurnal pattern where most of the bandwidth usage happens during the daytime between 8 am and 11 pm. The peak bandwidth is about 6 Gbps at 11 pm (7.5 Gbps as per-minute average), and the lowest bandwidth is 2 Gbps at 4 am, 3x smaller than the peak bandwidth. Interestingly, the bandwidth demand during the weekend is slightly larger than that of the weekdays. We also observe small peaks every weekday in the rush hour (8 - 9 am) and during lunchtime (12 - 1 pm), which implies that people use smartphones for Internet access while commuting or having lunch. We further investigate the amount of traffic by their HTTP content types in Figure 6. Interestingly, the numbers (left plot) and byte volumes (right plot) of flows by each content type remain stable regardless of the time of the day, exhibiting self-similarity [21]. Multimedia objects contribute to only 0.8% by the flow count, but its byte volume constitutes 40.6% of the traffic. This implies that caching solutions focusing on multimedia objects could deal with a significant portion of traffic at a small flow management cost.

Percentile (%)	Concurrent Flows (# flows)	Created Flows (# flows/min)	Duration (sec)
1%	55,000	218,000	0.09
10%	67,000	288,000	0.17
50%	197,000	931,000	1.01
90%	237,000	1,087,000	11.01
99%	257,000	1,168,000	64.52
Average	174,000	812,000	6.52

Table 4: TCP flow statistics for a week. Flow counts are snapshotted every minute.

TCP Flow Characteristics. We examine the characteristics of the TCP flows more in depth, which can suggest the performance requirements of middleboxes such as RE systems. Table 4 shows TCP flow-level statistics for the entire week. We observe that the number of concurrent flows changes dynamically from 50K (at 5 am) to 270K (at 11 pm), and the number of newly-created TCP flows per minute ranges from 0.2M to 1.3M. While most of the flows are short-lived (1 second as median, 11 seconds as 90^{th}%-tile), the distribution has a long tail that spans more than one day.

Figure 7 analyzes the flow sizes by their number (left plot) and by their byte volume (right plot) over seven one-hour time slots,

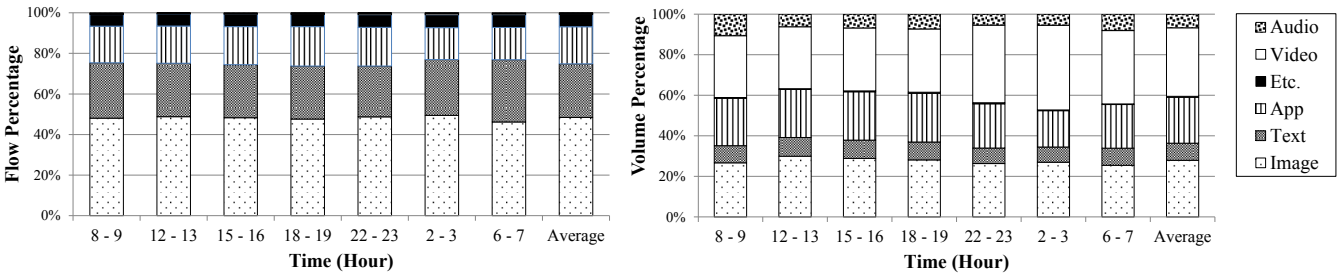


Figure 6: The percentage of flows and byte volume by the HTTP content type

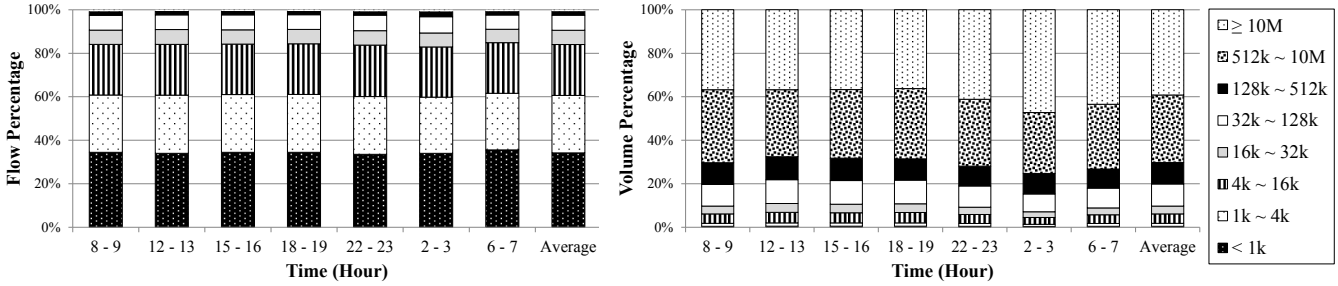
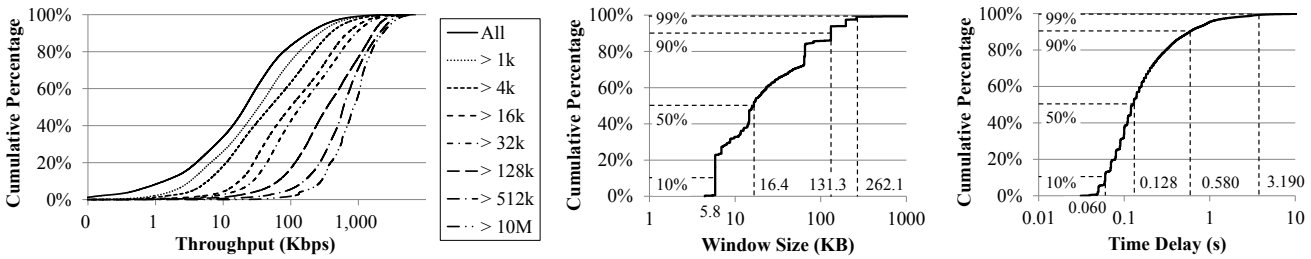


Figure 7: Distributions of TCP flows by flow size over various time slots



(a) Throughput by flow size (b) Advertised window size (c) Connection setup delay

Figure 8: Cumulative distribution of throughputs, advertised window sizes, and connection setup delays

aggregated for one week. On average, 60.6% of the flows are smaller than 4 KB and only 9.4% of the flows are larger than 32 KB. In contrast, the volume of the flows larger than 32 KB accounts for 90.3% of the total traffic. This implies that if TCP-RE systems bypass these small flows (*e.g.*, by checking the `content-length` field), we can reduce almost 90% of the flow management overhead while still covering more than 90% of the total volume. Similar to Figure 6, flow size distributions also exhibit self-similarity.

TCP Performance. Figure 8(a) shows the throughput distributions of individual TCP flows grouped by the flow size between 10 - 11 pm for one week. Overall, the median throughput is 21.1 Kbps, and even 90^{th}%-tile is only 182.4 Kbps. The low throughput seems to be because most flows are short-lived, and thus do not leave the TCP slow-start phase. This is in line with Figure 8(b), which shows the median advertisement window as only 16 KB. On the other hand, for larger flows, we observe the maximum throughput of 13.6 Mbps and the window size of up to 16.0 MB. Figure 8(c) also shows the connection setup delay between client-side SYN and ACK, which includes the round-trip time and server-side connection creation overhead. We observe the minimum, median, and 99^{th}%-tile delays of 39 ms, 128 ms, and 3.19 seconds, respectively.

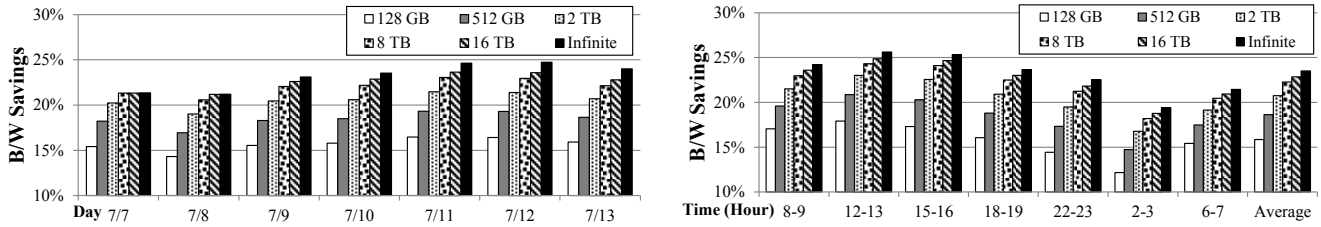
6. REDUNDANCY IN THE 3G TRAFFIC

In this section, we compare the effectiveness of standard Web caching, prefix-based Web caching, and TCP-RE from various perspectives. We also discuss different RE designs for high-speed cellular backhaul networks, and study the origin and temporality of the redundancy.

6.1 Effectiveness of Web Caching

We measure the effectiveness of Web caching by simulating our HTTP log trace. We follow the HTTP 1.1 standard [24] for caching, and assume a log-based disk cache as in [17]. Overall, we find that 53.9% of the total objects are cacheable and they account for 40.7% of the downlink traffic (by the byte volume) during the period. With infinite Web cache, one can save 23.7% of the downlink bandwidth for the week by standard Web caching.

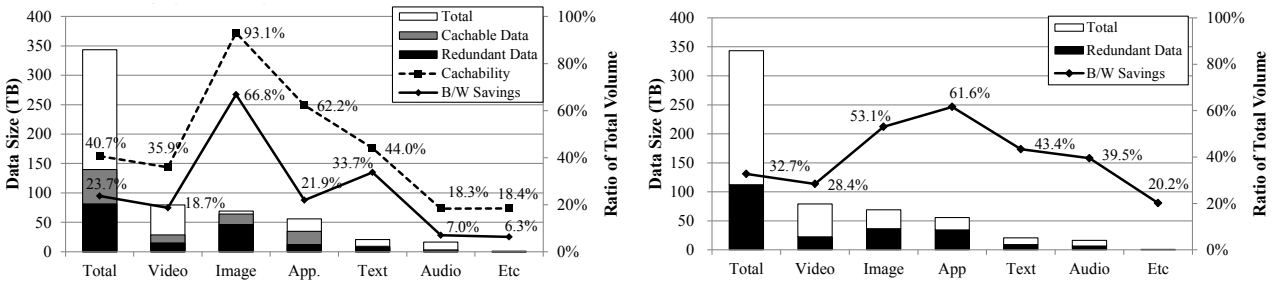
Figure 9(a) shows daily bandwidth savings with various cache sizes from 128 GB to 16 TB as well as infinite cache. The difference in bandwidth savings slowly grows over time, but the working set size is relatively small since even 512 GB cache, which fits in memory, can provide bandwidth savings ratio of 18.6%. Figure 9(b) shows the difference in bandwidth savings at different times of the day, averaged for seven days. Morning rush hour (8 - 9 am), lunch time (12 - 1 pm), and mid-afternoon (3 - 4 pm) produce relatively



(a) Various Web cache sizes

(b) Different times of a day, averaged for seven days

Figure 9: Bandwidth savings by standard Web caching



(a) Web cache (infinite size)

(b) TCP-RE (512GB)

Figure 10: Redundancy by content type with Web caching and TCP-RE

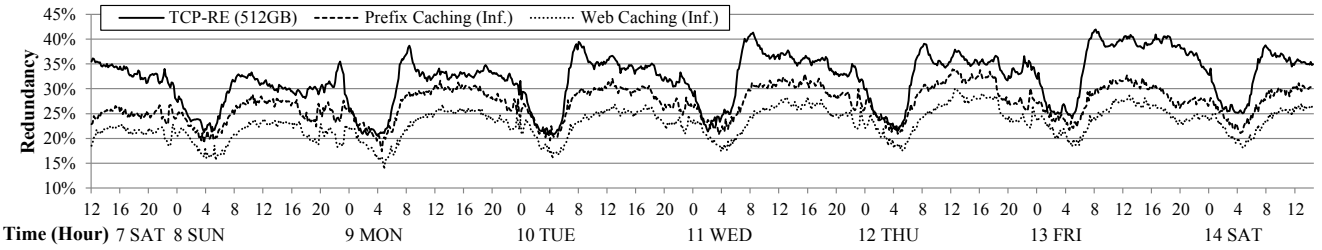


Figure 11: Comparison of bandwidth savings by Web caching, prefix-based Web caching, and TCP-RE

larger bandwidth savings, while early morning (2 - 3 am, 6 - 7 am) shows the smallest savings.

Figure 10(a) shows the breakdown of cacheability and cache hit ratios by content types. Images produce the largest bandwidth savings. 93.1% of the image bytes are cacheable, and 66.8% of them lead to bandwidth savings. Applications are highly cachaeable as well (62.2%), but their bandwidth savings ratio is limited to 21.9%. Video takes up the largest traffic, but only 35.9% is cacheable with 18.7% of bandwidth savings. Audio exhibits the lowest cacheability, and we suspect this is one of the reasons why rush hour peaks cannot be suppressed by Web caching as much as TCP-RE as shown in Figure 11.

6.2 Effectiveness of Prefix-based Caching

Prefix-based Web caching complements the weakness of standard Web caching by serving aliased, or redundant but uncacheable content. It operates as standard Web caching for the HTTP requests, but when a request is a cache miss, it compares the hash of the N-byte prefix and the length of the response to confirm an alias. If the aliased object is found in its cache, it stops downloading from the origin, and delivers the leftover from its cache instead.

Prefix Key Size vs. False Positives. The prefix key size is crucial for the effectiveness of prefix-based Web caching. In general, a smaller prefix key size would produce more bandwidth savings at the risk of having a higher false positive rate. On the other hand, a larger prefix key size cannot suppress aliased objects that are smaller than the key, so it produces smaller bandwidth savings, but with a lower false positive rate.

Table 5 shows the number of false positives over various prefix key sizes from 1 KB up to 1024 KB for *all* HTTP objects downloaded for the measurement period. Surprisingly, we find false positives in every prefix key size while the number of false positives decreases as the key size increases. We manually inspect some of false positives. For small prefix keys, we find that many of them are due to modifications on the texts (*e.g.*, HTML pages or scripts like CSS and Javascript) or metadata (*e.g.*, timestamps). We also find some large animated GIFs that have the same content up to the 1 MB prefix key size, but contain different image frames located beyond 1 MB.

Bandwidth Savings. Figure 13 shows the bandwidth savings with infinite cache across various content types. We choose 1 MB as the prefix key size as it minimizes the false positives in our trace. We perform similar simulations as in Web caching for 2 TB and infinite cache sizes. Prefix-based Web caching saves 27.7% of the downlink traffic (excluding the false positives) during the period, improving Web caching by only 4%. While audio, application, and video show 25.4%, 6.4%, 1.3% additional bandwidth savings than

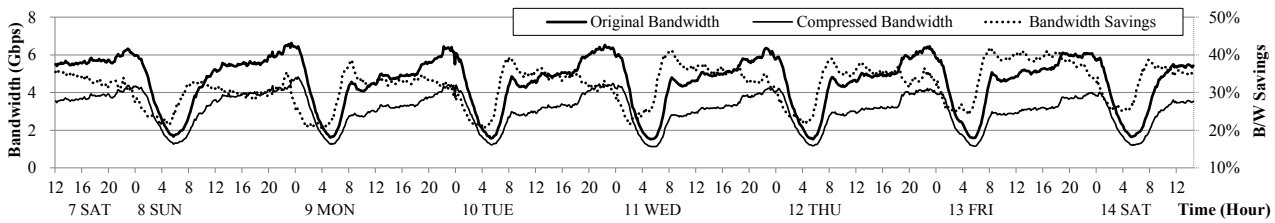


Figure 12: Original and estimated bandwidth usages by TCP-RE (512 GB cache)

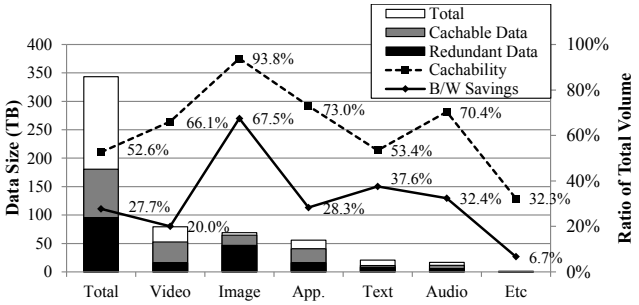


Figure 13: Bandwidth savings by prefix-based caching

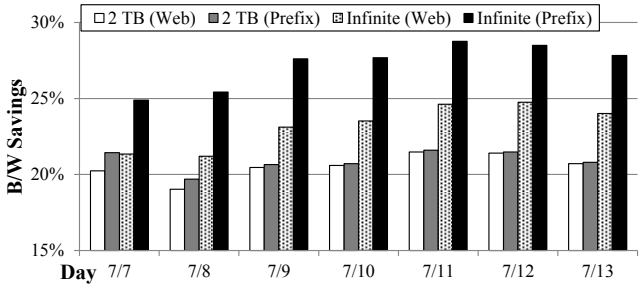


Figure 14: Comparison of daily bandwidth savings by standard and prefix-based Web caching

Prefix Key Size	Number of FP Object	% of Total Object	Size of FP Object	% of Total Traffic
1 KB	52,936,642	0.68%	3.79 TB	1.10%
4 KB	14,993,908	0.19%	3.49 TB	1.02%
16 KB	4,173,023	0.05%	2.93 TB	0.85%
128 KB	277,039	0.004%	0.60 TB	0.18%
256 KB	102,689	0.001%	0.33 TB	0.10%
512 KB	39,519	0.001%	0.25 TB	0.07%
1024 KB	24,732	0.0003%	0.19 TB	0.05%

Table 5: False positives for prefix-based Web caching

standard Web caching, respectively, other types are typically smaller than 1 MB, and do not see further savings.

Finally, Figure 14 compares daily bandwidth savings of standard and prefix-based Web caching. On the first day, prefix-based Web caching with 2 TB cache shows slightly larger bandwidth savings than Web caching with infinite cache. The difference between standard and prefix-based Web caching with infinite cache stays more or less the same (3∼4%) over time. This implies that (a) the additional redundancy detected by prefix-based Web caching is limited by a large prefix key size, and (b) old content may not benefit from prefix-based Web caching as much as the new content does.

6.3 Effectiveness of TCP-RE

We present the effectiveness of TCP-RE over the measurement period in Figure 11 where we use the LRU cache replacement algorithm with 512 GB chunk cache size. Overall, we find 30∼40% of bandwidth savings most of the day (8 am - midnight) while it drops below 20% early in the morning (4 - 6 am).

Figure 10(b) compares bandwidth savings by HTTP content type. Video, image, application take up over 80% of the total HTTP traffic, and contribute to 85% of the total bandwidth savings. Image and application show high redundancy while the redundancy in video is relatively lower. Compared with Web caching, however, more redundancy is found for video since they are often uncacheable with `sessionID/userID` embedded in their URLs (*e.g.*, YouTube).

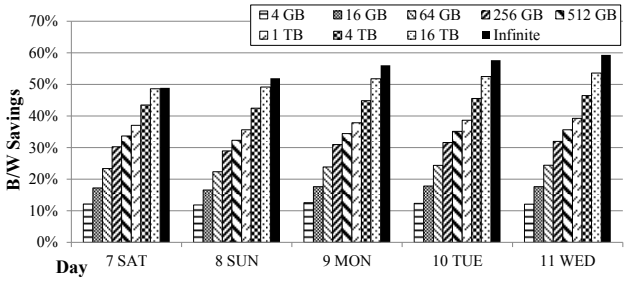
Redundancy at Peak Bandwidths. In general, the bandwidth savings shows a positive correlation with the bandwidth usage, but the peak times (10 - 11 pm) do not always yield higher redundancy. Figure 11 indicates that the periodic bandwidth peaks in the rush hour can be effectively suppressed by TCP-RE. The content types that drive the local peaks are audio and text, which implies that many users are listening to popular music or reading popular online news while commuting. In contrast, these periodic bandwidth peaks cannot be addressed by standard Web caching that saves 14.4∼18.2% less bandwidth than TCP-RE. Further investigation shows that many of those audio objects are from subscription-based music services, and are set uncacheable. Similar periodic bandwidth spikes during lunch time can also be effectively reduced by TCP-RE.

On the other hand, some bandwidth savings at 10 - 11 pm shows a slightly negative correlation with the bandwidth usage (*e.g.*, Mon., Tue., and Wed.) where a similar behavior is observed in enterprise traffic analysis [13]. Nevertheless, TCP-RE still promises 30.7∼36.9% bandwidth savings at the peak time in the 3G traffic. Figure 12 shows the estimated bandwidth usage for the seven days (called compressed bandwidth) after applying TCP-RE. Bandwidth fluctuation is much smaller with TCP-RE, which effectively curbs the usage below 5 Gbps.

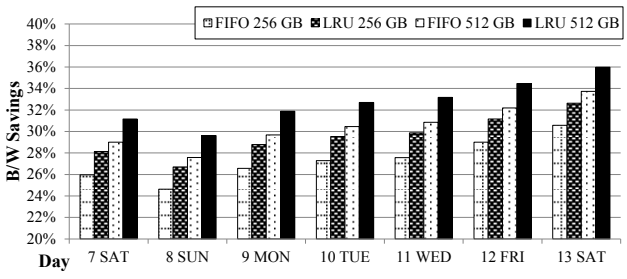
Cache Size and Replacement Policy. We further investigate the effect of different cache sizes and cache replacement algorithms on bandwidth savings in TCP-RE. Due to the large volume of our traces and the limitation in the available memory (144 GB) of our machine, we divide the traces into 24 disks by chunk hash values and calculate the bandwidth savings on one partition.

First, Figure 15(a) shows bandwidth savings for the first five days [4] where each bar represents bandwidth savings for each day. We increase the cache size from 4 GB to infinite, and use the LRU cache replacement policy. Except for the infinite cache case, the bandwidth savings stays more or less the same. The infinite cache can find 48.9∼59.3% of the redundancy while even 4 GB cache reduces the traffic by more than 10%. As expected, the bandwidth savings grows as the cache size increases, but we observe diminish-

[4]Due to the the memory limitation, we could not simulate infinite cache beyond the first five days.



(a) Bandwidth savings over various cache sizes



(b) Comparison of bandwidth savings between FIFO and LRU

Figure 15: TCP-RE bandwidth savings by cache size and cache replacement algorithms

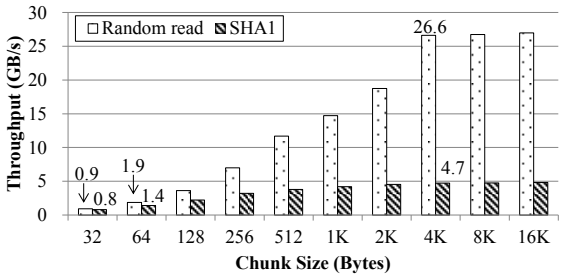


Figure 16: Memory I/O performance on Monbot

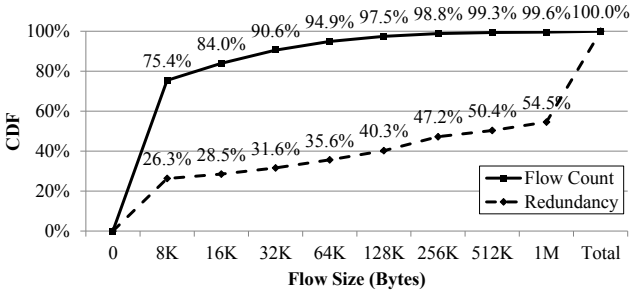


Figure 17: TCP-RE bandwidth savings by flow size

ing gains. This is because the popularity of the chunks follows the Zipf distribution, which we will discuss further in Section 6.5.

Next, we compare the impact of FIFO and LRU cache replacement algorithms on the bandwidth savings for two practical cache sizes that can be installed on currently available commodity servers. Figure 15(b) suggests that LRU is slightly better (2.1∼2.3%) than FIFO for the same cache size. It implies that the redundancy comes mostly from temporal locality.

6.4 TCP-based RE vs. Packet-based RE

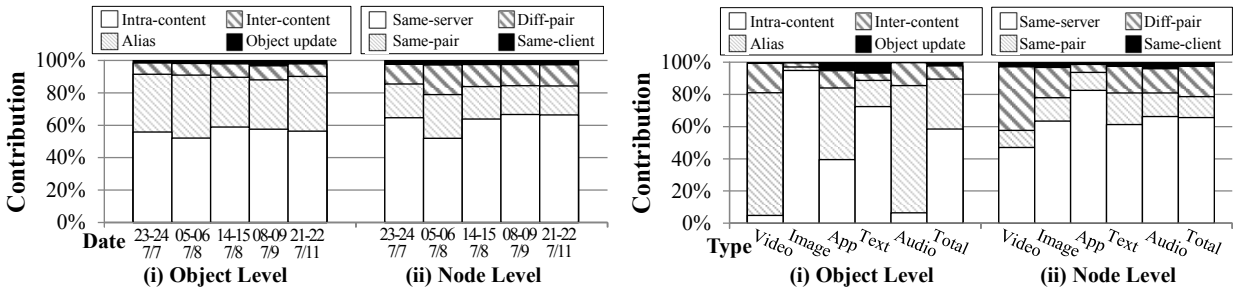
We discuss performance trade-offs of different RE system designs, and their suitability in high-speed cellular backbone networks. While we assume middlebox-based TCP-RE here, an alternative approach is packet-level RE [14, 42], which detects candidate fragments for RE within each IP packet, typically using content-based fingerprinting functions [11, 13, 39]. Packet-level RE is simpler because it does not manage TCP flows nor terminate the TCP connections at middleboxes. Also, it can potentially find more redundancy since it operates on smaller chunk sizes (*e.g.*, 32 to 64 bytes [11, 13, 42]).

However, there are a few drawbacks with packet-level RE in high-speed cellular backhaul networks. First, small chunks can easily explode the index sizes in a high-speed network. For example, a 64B average chunk size may require 64x more index entries than 4 KB chunks for the same cache size. Second, a small chunk size would incur a higher hashing and content reconstruction cost, and it stresses the memory system which is typically the performance bottleneck in memory-based RE. Figure 16 compares random memory read and SHA1 hashing performances over various chunk sizes when all 12 cores on Monbot do only the memory operations. We run the experiment five times and show the averages. Both random memory read and SHA1 performances increases as the chunk size grows and saturates at 4 KB. The 4 KB random read performance is 28.3x and 14.2x larger than those of 64B and 32B chunks, presumably due to much smaller TLB cache misses and page table operations. From the Figure, it looks very challenging to achieve 10 Gbps RE performance with 32B or 64B chunks on a single machine. Similarly, SHA1 performance of 4 KB chunks is 5.78x or 3.36x higher than those of 32B or 64B chunks. Some previous work [11] reduces the hashing overhead by maintaining a synchronized chunk cache, but it would be challenging to be applied to cellular networks since the packet routes can change abruptly due to user mobility. Finally, it is difficult to enforce flow-based policies with packet-level RE. Cellular ISPs may want to apply RE to only large file downloads or to specific content types that exhibit high redundancy with a small flow management overhead.

Instead, middlebox-based TCP-RE with fixed 4 KB chunks strikes a balance between memory I/O costs and bandwidth savings, which we think is reasonable for cellular backbone networks. It could even reduce the flow management cost while still offering a reasonable level of bandwidth savings by selectively processing flows. Figure 17 shows the cumulative distribution functions (CDFs) of the redundancy and the number of flows for the flow size. It shows that the largest 10% of the flows bring about 70% of the bandwidth savings. That is, bypassing any HTTP responses smaller than 32 KB could eliminate the cost of managing 90.6% of the flows while still providing 31.6% of the bandwidth savings.

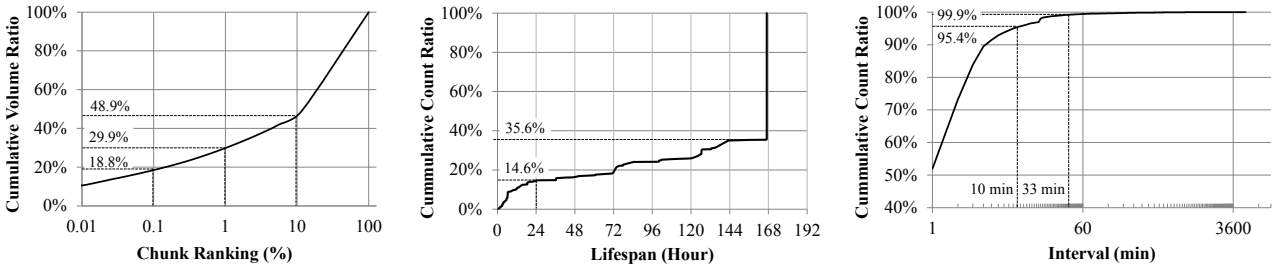
6.5 Source of Redundancy

We study the source of redundancy to help determine the effective caching strategies and deployment options. First, we analyze the source of the HTTP object-level redundancy. Specifically, we measure the fraction of redundancy from serving the same object with the same URL (`intra-content`) and with different URLs (`alias`), partial content overlap in the objects with the same URL (`object update`) and with different URLs (`inter-content`). Second, we measure the fraction of the redundancy in the communicating node pair. We mark it `same-pair` if the redundancy is found from the same client and server pair, `same-server` if it comes from the same server but different clients, `same-client` if it comes from the same client interacting with different servers, and `diff-pair` if the redundancy is from different client and server pairs.



(a) By time slot (b) By content type

Figure 18: Source of redundancy



(a) Volume distribution by chunk popularity (b) Cache-hit interval distribution (c) Chunk lifespan distribution

Figure 19: Temporal analysis of the most popular 1000 chunks

Figure 18(a) shows the results for five one-hour time slots that are carefully chosen based on bandwidth usage. Each bar represents an average for seven days. Interestingly, the behavior is similar across all time slots. The intra-content redundancy shows 52.1∼58.9% while alias contributes to 30.4∼38.9%. The redundancy from partial content overlap is less than 9% most of the time. This implies that Web caching is effective in addressing a large portion of the redundancy while prefix-based Web caching with a small prefix key size could suppress most of the redundancy if it can avoid false positives.

For the node level analysis, we assume that it is the same client if it uses the same IP, the same mobile device model, and the same OS version in the user-agent field. If we detect the change of the device model or the OS version, we assume that the IP is reassigned and consider it as a different client. While this can bloat the actual redundancy from a client, we find that the redundancy from the same client (e.g., same-pair, same-client) is less than 21%, which implies the client-side RE would produce limited bandwidth savings. This is in contrast to the enterprise traffic where as much as 75% of the redundancy is attributed to the intra-host traffic [11]. The majority of the redundancy in our 3G traffic comes from downloading popular contents from the same server.

Figure 18(b) shows the source of redundancy by the content type. Overall, the major source of redundancy is from the same contents (intra-content, alias) and from the same server (same-server, same-pair). However, the contribution by aliases is much higher for media contents such as video and audio than other contents since they are usually served by multiple servers for load balancing. On the other hand, we find that the contribution by partial content overlap (inter-content, object update) and client-side RE (same pair, same-client) is limited.

The redundancy in the non-HTTP traffic is 2.8∼12.9% in the time slots, which is lower than that of the HTTP traffic. The top traffic contributor is P2P video streaming that uses port 17600 [10] and the SSL traffic, taking up 3.6% and 3.4% of the downlink traffic each.

Data Type	10-11 pm 7/7	11 pm-12 am 7/7	5-6 am 7/8	8-9 am 7/8
Chunks	0.7441	0.7512	0.7177	0.7490
Web Objects	0.8820	0.8799	0.8798	0.8755

Table 6: Zipf's α values for one-hour time bins

We see that the P2P traffic with port 17600 significantly increases in early morning (up to 8.8% in 4-6 am), which partly explains why the bandwidth savings decreases during the time.

6.6 Temporality of Redundancy

We find that both HTTP objects and chunks follow the Zipf distribution. Table 6 shows that the Zipf α values of Web objects (0.88) are relatively larger than those of chunks (0.72∼0.75), but we actually find that TCP-RE is more effective even with a smaller cache size. This implies that the cache duration and the cacheability of the HTTP objects set by the content providers may be suboptimal in terms of bandwidth savings.

We also analyze the temporal locality of the chunks in the first one hour of our measurement more closely. Overall, we find strong temporal locality of the chunks. During the first hour, the most popular 1% chunks represent 29.9% of the total downlink bytes in the hour and the top 10% chunks represent 48.9% of the hourly volume as shown in Figure 19(a). Figure 19(b) shows the distribution of hit intervals of the most popular 1,000 chunks in the first hour. 95.4% of all hits occur within 10 minutes while 99.0% of them are within 33 minutes. Lastly, Figure 19(c) shows the distribution of the lifespans (the time between the first and the last hits of the same chunk) of the most popular 1,000 chunks in the first hour for the remaining 7 days. We find that 64.4% of the chunks are accessed throughout the seven days while 14.6% of them are used only on the first day. We also find that there is no strong correlation between the lifespan and the hit-count rank.

Motivated by the high temporal locality of chunks, we simulate

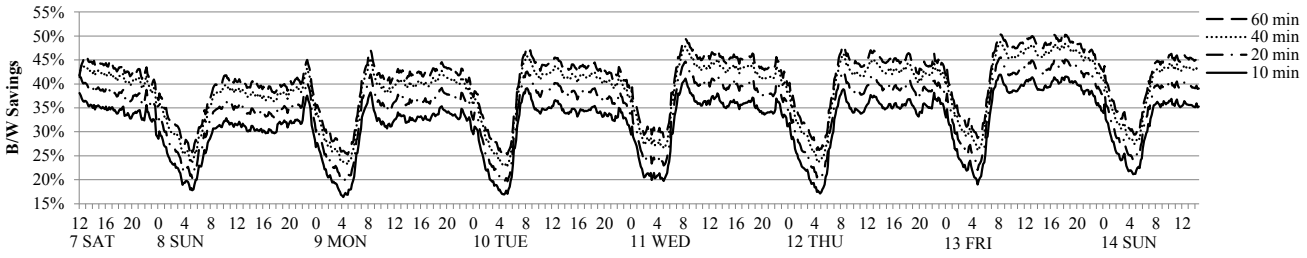


Figure 20: TCP-RE bandwidth savings for caching for the most recent N minutes

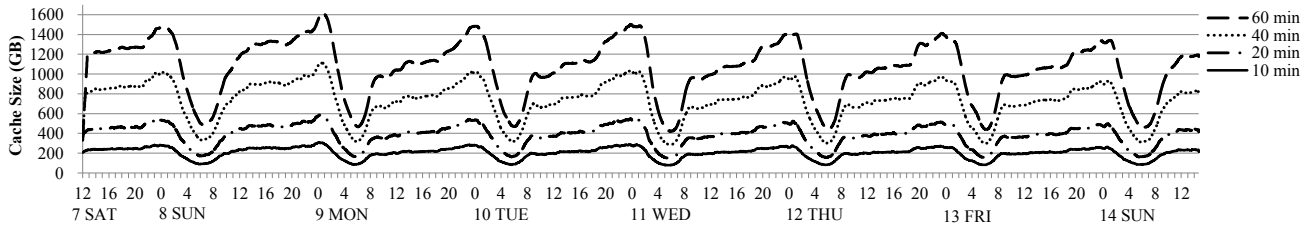


Figure 21: Required TCP-RE cache size for caching for the most recent N minutes

the effect of limited but realistic TCP-RE cache sizes. Figure 20 and Figure 21 show the bandwidth savings and required cache sizes when caching chunks only for the most recent 10 to 60 minutes. We achieve as much as 30∼40% of the bandwidth savings by caching chunks for the most recent 10 minutes while caching for 60 minutes increases the bandwidth savings by only 10% at the cost of linear increase in the required cache size. That is, the high temporality of the chunks greatly helps suppress the redundancy even with small cache size but larger cache size would yield a marginal increase in bandwidth savings.

One interesting finding is that the top two chunks consist of a stream of 0x00 (chunk with all 0's) and 0xFF (chunk with all 1s), and they are used for various content types like image, app, and even text. We suspect these are due to padding generated by commonly-used libraries.

7. IMPLICATIONS FOR CACHING MIDDLEBOXES

This section summarizes our key findings that could shape the design of caching middleboxes for high-speed cellular backhaul networks.

- Caching middleboxes should be able to efficiently manage a large number of concurrent flows in high speed links. In our data set, we observe up to 270K concurrent flows, 1.3M new flows per minute, and about 7.5 Gbps peak per-minute bandwidth usage.
- Unlike wired network traffic, our mobile network traffic incurs several periodic spikes in bandwidth usage related to user mobility during the day (*e.g.*, rush hour, lunch time, and midnight).
- Among different caching strategies, we find TCP-RE with 512 GB memory cache outperforms other approaches, and it is even effective for suppressing peak bandwidth usage. While prefix-based Web caching offers higher bandwidth savings than standard Web caching, it comes with false positives that severely limit its utility. Due to the Zipf-like distribution of chunk popularity, LRU slightly outperforms FIFO.
- Packet-based RE may be unsuitable for high-speed cellular backhaul links because of its high overhead on indexing and memory I/Os. We also find that an end-host based caching approach provides only limited bandwidth savings.
- Most flows are short-lived, but a small fraction of large flows (*e.g.*, video) dominate the entire traffic volume. This offers an optimization opportunity for RE systems to reduce the flow management cost while still providing a reasonable level of bandwidth savings.

8. RELATED WORK

Our work is built upon previous research in many areas. In this section, we relate our work to this previous work.

Scalable Traffic Analysis on Multicore Systems. Various efforts exploited processing parallelism in multicore systems for scalable packet/flow management [25, 36, 41]. Sommer *et al.* build a custom FPGA-based NIC that manages a large table indexed by five tuples of a packet header, and dispatches packets to core-pinned CPU threads based on the routing decision in the table entry [41]. For high scalability, they ensure the core affinity of packet processing but allow event queues to exchange information between threads. Fusco *et al.* present virtual capture devices for scalable packet analysis on a commodity NIC [25]. They use RSS to distribute incoming packets to multiple RX queues, and virtualize a physical NIC to provide multiple virtual devices for packet capture threads. Affinity-accept focuses on core-scalable `accept()` on an end node by removing lock contention and ensuring core affinity in incoming TCP packet processing [36].

In comparison, Monbot achieves scalable TCP flow processing in a middlebox using a commodity NIC. Monbot extends PacketShader I/O engine [26] to process all packets in the same TCP connection in the same core by symmetric RSS without lock contention or any sharing of flow context across different cores.

Network Redundancy Elimination. There have been many works that suggest redundancy suppression for WAN acceleration. Spring *et al.* first present packet-level RE based on Rabin's fingerprinting, and showed that 39% of the uncacheable Web traffic can be saved [42]. More recently, Anand *et al.* report 59% and 31% of potential bandwidth savings for small and large enterprise traffic [13]. They also find that over 75% of the savings are from intra-host matches. Drawing from this fact, EndRE suggests pushing the RE cache to clients [11]. Other proposals include deploying network-

wide RE by carefully coordinating the router cache [14], and supporting RE as a primitive service in the IP-layer [12]. Furthermore, PACK [47] shows the power of chunk prediction in reducing the redundant traffic for cloud traffic. Ihm *et al.* find 42.0∼50.6% of the redundancy in the CoDeeN content distribution network Web traffic [28].

We find that the cellular network traffic could also benefit from significant bandwidth savings by TCP-RE. Unlike the enterprise traffic, our measurements show that the redundancy in the intra-host traffic is rather limited, and the middlebox-based approach would bring higher redundancy suppression in the cellular backhaul networks. Furthermore, our work deals with one or two orders of magnitude larger traffic than previous works [13, 23], showing the daily redundancy pattern, flow dynamics, and detailed comparison of effectiveness of various caching strategies.

Web Caching in Cellular Networks. Erman *et al.* report 68.7% of the HTTP objects are cacheable with 33% of the cache hit ratio in their two-day measurements in a 3G cellular network in 2010 [22]. While the fraction of cacheable objects is smaller (53.9%) in our measurement, the cache hit ratio is similar (38.3%). Another work finds that 36% of the 3G traffic is attributed to video streaming, and 24% of the bytes can be served from HTTP cache [23]. Our numbers match the volume of the traffic (33.8%), but the video bandwidth savings in Web cache is smaller in our trace (18.7%). However, we show that TCP-RE could increase the bandwidth savings further by 9.7% for video. Qian *et al.* report that Web caching on smartphones (*e.g.*, browser or application cache) could be ineffective due to the lack of support for Web caching or library bugs [38]. More recently, Qian *et al.* measure the effectiveness of Web caching, file compression, delta encoding, and packet-level RE using packet traces collected from 20 mobilephone users for 5 months [37]. They find that combining all of the four methods could reduce the cellular traffic by 30%. In comparison, our middlebox-based TCP-RE provides the similar bandwidth savings without any modification of the server and the client. In addition, we believe packet-level RE that they use is difficult to deploy in practice since it is challenging to synchronize the RE cache due to device mobility.

Breslau *et al.* report that the popularity of Web objects follows the Zipf distribution [18]. We see the same trend in our 3G Web traffic with $\alpha = 0.88$. This suggests that even a small cache reduces a large amount of Web traffic, which is implied in Figure 9(a). Chunk-level popularity shows smaller α values, but it is more effective than Web caching even with a smaller cache size.

TCP Performance in Cellular Networks. The TCP performance of the live cellular networks has been measured in previous research. Chesterfield *et al.* report an average throughput of 350 Kbps and 200 ms of delays in early 3G UMTS networks in 2004 [19]. Later, Jurvansuu *et al.* report an effective bandwidth of 1 Mbps and 61 ms of delays in HSDPA networks in 2007 [29].

In contrast, our measurements show that the average throughput is only 111.92 Kbps because the majority of the TCP flows are small and they do not leave the slow start phase. For flows larger than 1 MB, however, the average bandwidth is 0.95 Mbps, and the maximum throughput we observe reaches 13.6 Mbps.

9. CONCLUSION

Caching in the cellular networks is an attractive approach that can address the backhaul congestion in core networks. By monitoring 8.3 billion flows of 3G backhaul traffic for one week, we have compared the benefits and trade-offs of promising caching solutions. We have found that TCP-RE is the most effective in terms of bandwidth savings and cache size overhead. It gives 14% additional bandwidth savings compared to standard Web cache, with only 512 GB memory. Prefix-based caching is effective with a small key size, but we find that it produces false positives even with a large prefix size such as 1 MB. We have also shown the flow-level characteristics of modern 3G traffic, which should be useful for provisioning the capacity of the networks or for designing various middleboxes for cellular networks. Finally, we have presented Monbot, a high-performance DFI system on commodity hardware that scalably manages hundreds of thousands of concurrent flows by balancing the load across CPU cores with symmetric RSS. We believe Monbot and symmetric RSS could serve as a basis for flow managing middleboxes in high-speed networks.

10. ACKNOWLEDGMENTS

We appreciate many valuable comments from the anonymous reviewers of ACM MobiSys 2013. We also thank our shepherd, Morley Mao, for her help in improving the paper, and Vivek S. Pai for providing comments and insights on the early draft of the paper. We thank Kyoungjun Lee who greatly supported setting up the measurement environment in the ISP. We thank Marc Fiuczynski who helped catch numerous editorial errors in the final version. This project is supported in part by the grant from SK Telecom and by the National Research Foundation of Korea (NRF) grant #2012R1A1A1015222.

References

[1] 3GPP LTE. http://www.3gpp.org/LTE, 2012.

[2] ComWorth SwiftWing SIRIUS. http://www.comworth.com.sg/products/swiftwing-sirius-line-rate-capture-storage-system, 2012.

[3] Endace DAG. http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html, 2012.

[4] Endace Intelligent Network Recoder. http://www.endace.com/endace-high-speed-packet-capture-probes.html, 2012.

[5] Libzero for DNA: Zero-copy flexible packet processing on top of DNA. http://www.ntop.org/products/pf_ring/libzero-for-dna/, 2012.

[6] Mobile operators risk backhaul gap in LTE networks. http://www.telecomstechnews.com/blog-hub/2013/feb/13/mobile-operators-risk-backhaul-gap-in-lte-networks/, 2012.

[7] Napatech NT40/NT20. http://www.napatech.com/products/capture_adapters/1x40g_pcie_nt40e2-1_capture.html, 2012.

[8] NPulse HammerHead. http://www.npulsetech.com/Products/HammerHead-Flow-Packet-Capture.aspx, 2012.

[9] Perfecting policy control. http://www.ericsson.com/res/docs/whitepapers/WP-e2e-policy-control.pdf, 2012.

[10] AfreecaTV, Inc. http://www.afreeca.com/.

[11] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese. EndRE: An end-system redundancy elimination service for enterprises. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[12] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: The implications of universal

redundant traffic elimination. In *Proceedings of ACM SIGCOMM*, 2008.

[13] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: Findings and implications. In *Proceedings of the ACM SIGMETRICS*, 2009.

[14] A. Anand, V. Sekar, and A. Akella. SmartRE: An architecture for coordinated network-wide redundancy elimination. In *Proceedings of ACM SIGCOMM*, 2009.

[15] ARA Networks, Inc. http://www.aranetworks.com/.

[16] ARS Technica. The State of 4G: It's All about Congestion, not Speed. http://arstechnica.com/tech-policy/2010/03/faster-mobile-broadband-driven-by-congestion-not-speed/, 2010.

[17] A. Badam, K. Park, V. S. Pai, and L. Peterson. HashCache: Cache storage for the next billion. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[18] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM*, 1999.

[19] J. Chesterfield, R. Chakravorty, J. Corwcroft, P. Rodriguez, and S. Banerjee. Experiences with multimedia streaming over 2.5G and 3G networks. In *Proceedings of the International Conference on Broadband Communications, Networks, and Systems (BROADNETS)*, 2004.

[20] Cisco, Inc. Cisco visual networking index: Global mobile data traffic for 2011-2016. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html, 2012.

[21] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835 – 846, 1997.

[22] J. Erman, A. Gerber, M. T. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck. To cache or not to cache: The 3G case. *IEEE Internet Computing*, 15(2):27–34, Mar. 2011.

[23] J. Erman, A. Gerber, K. Ramakrishnan, S. Sen, and O. Spatscheck. Over the top video: The gorilla in cellular networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, 2011.

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[25] F. Fusco and L. Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, 2010.

[26] S. Han, K. Jang, K. Park, and S. B. Moon. PacketShader: a GPU-accelerated software router. In *Proceedings of ACM SIGCOMM*, 2010.

[27] S. Han, S. Marshall, B.-G. Chun, and S. Ratnasamy. MegaPipe: a new programming interface for scalable network I/O. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2012.

[28] S. Ihm and V. Pai. Towards understanding modern web traffic. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, 2011.

[29] M. Jurvansuu, J. Prokkola, M. Hanski, and P. Perala. HSDPA performance in live networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, 2007.

[30] H. Krawczyk. LFSR-based hashing and authentication. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 1994.

[31] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband Internet traffic. In *Proceedings of the ACM SIGCOMM Conference on Internet measurement Conference (IMC)*, 2009.

[32] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Proceedings of the Passive and Active Measurement (PAM)*, 2010.

[33] I. Microsoft. MSDN: Introduction to receive-side scaling. http://msdn.microsoft.com/en-us/library/windows/hardware/ff556942%28v=vs.85%29.aspx, 2012.

[34] Oversi, Inc. http://www.oversi.com/.

[35] PeerApp, Inc. http://www.peerapp.com/.

[36] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2012.

[37] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck. How to reduce smartphone traffic volume by 30%? In *Proceedings of the Passive and Active Measurement (PAM)*, 2013.

[38] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones: Ideal vs. reality. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[39] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, 1981.

[40] E. Seidel and E. Saad. LTE home node BS and its enhancements in release 9. http://www.nomor.de/uploads/fc/lp/fclpAIhtNJQ9zwyD957atQ/2010-05_LTE_HomeNB_Rel9_Overview.pdf, 2012.

[41] R. Sommer, V. Paxson, and N. Weaver. An architecture for exploiting multi-core processors to parallelize network intrusion prevention. *Concurrency and Computation: Practice and Experience*, 21(10):1255–1279, 2009.

[42] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, 2000.

[43] The Cuttting Edge. U.S. could see $53 billion in 4G network investments in five years. http://www.thecuttingedgenews.com/index.php?article=52617&pageid=28&pagename=Sci-Tech, 2011.

[44] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. MIDeA: A multi-parallel intrusion detection architecture. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.

[45] I. VerizonWireless. Verizon optimization deployment-terms & conditions. http://support.verizonwireless.com/terms/network_optimization.html, 2012.

[46] S. Woo and K. Park. Scalable TCP session monitoring with symmetric receive-side scaling. http://www.ndsl.kaist.ac.kr/~shinae/papers/TR-symRSS.pdf, 2012.

[47] E. Zohar, I. Cidon, and O. Mokryn. The power of prediction: Cloud bandwidth and cost reduction. In *Proceedings of ACM SIGCOMM*, 2011.