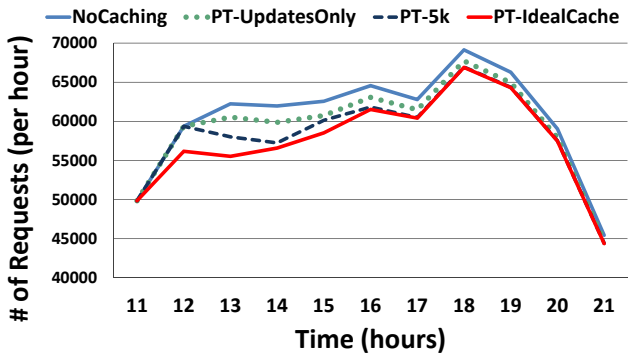


Figure 11: Search query volume reduction with PocketTrend (PT) for different update strategies (Boston marathon bomb).

ume lasts for several hours as the trending event evolves, and is progressively reduced over time as the trending event disappears.

Even more importantly, when passive updates are used (*PT-UpdatesOnly*), PocketTrend is still able to reduce the datacenter's query volume by 7–14%, depending on the trending event. Even though the benefit is lower compared to active updates, the fact that passive updates can achieve similar gains is important as they do not impose as high bandwidth requirements on the datacenter as active updates do (more about this in Section 4.4). However, there can be cases where active update strategies achieve significantly higher query volume reductions. For example, in the Boston marathon event (Figure 11), in the first two hours after the bombing (12pm – 2pm), *PT-5k* is almost twice more efficient than *PT-UpdatesOnly*. We believe that this has to do with the unpredictability of trending events. The Boston marathon bombing event was highly unexpected and dramatic for users, creating this instant spike in user query volume that only active updates can efficiently address initially. On the other hand, the U.S. president election was more expected, and it formed a trending event that slowly developed over time, providing enough time for passive updates to become almost as efficient as active updates[6].

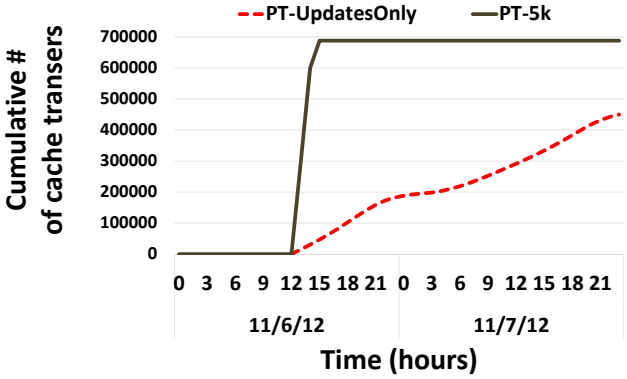
When compared to the ideal cache implementation, both active and passive updates perform remarkably well, with the active strategy having a small edge over the passive one. This is especially noticeable in Figure 10 where *PT-5k* is always within 3% of *PT-IdealCache*. More importantly, *PT-UpdatesOnly* can get at least 63% (usually more) of the benefits provided by *PT-IdealCache*.

Note PocketTrend's 17% reduction of query volume in the case of president elections is lower than the potential reduction of 30% shown in Figure 1. This is due to two reasons. First, it is nearly impossible for PocketTrend to timely predict every single user that will search for the trending event. For some users, predictions cannot be made; some users might also get updated after they have searched for the trending event. Second, Figure 1 presents the results when all trend-related queries are manually selected. In this section, trend-related queries are detected through the PocketTrend's two-step trending content identification algorithm, which might not include every single trend-related query.

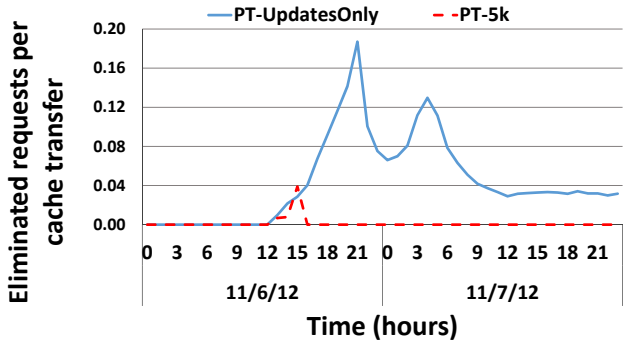
4.4 PocketTrend Overhead

Each time a search request is served locally at the client, one fewer request has to be processed by the datacenter. On the other hand, to avoid such requests the datacenter has to push trending

[6]When active and passive updates are combined, the reduction in the search query volume is only slightly higher than active updates.



(a) Cache transfers



(b) Eliminated datacenter requests per cache transfer

Figure 12: PocketTrend overhead measured as cumulative number of cache transfers (a) and number of eliminated requests to the datacenter per cache transfer (b) (U.S. President Elections).

content to the users, passively or actively. We quantify the overhead of PocketTrend on the datacenter with passive (*PT-UpdatesOnly*) or active (*PT-5k*) updates.

First, we measure the number of trending search content pushes performed by the datacenter. Figure 12(a) shows this for the U.S. president elections. For active updates, 5000 users are updated every minute with the trending search content. A large number of pushes takes place when the trending event is detected. After all users that submitted a search query within 2 hours before the trending event took place have been updated, no more pushes are initiated. With passive updates, pushes occur only for users that came to the search engine for other unrelated topics after the trending event was detected. Because of this, passive update pushes grow slowly over time, and at the end of the event's lifetime, they are half of the active update pushes.

Passive updates are not only fewer, but also more effective. For the same event, we compute the ratio between the number of search requests eliminated from the datacenter and the number of trending content pushes to end users. The higher this ratio is, the lower the overhead from the datacenter's point of view is. As Figure 12(b) shows, passive updates achieve a ratio of almost up to 20%, while active updates remain well below 4%.

Given that the passive and active update strategies provide comparable advantages to end users and to the datacenter, Figure 12 shows that passive updates can be a much more cost-effective approach for the datacenter and the end users. From the datacenter point of view, passive updates do not require any new requests to be

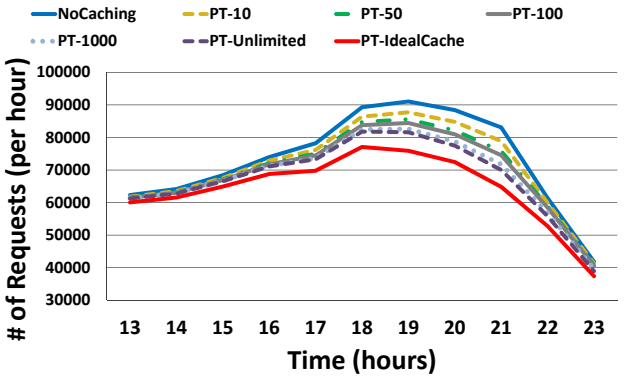


Figure 13: Effect of the size of the pushed trending search content on PocketTrend's query volume reduction (U.S. president elections).

submitted as trending search content is opportunistically pushed to end users, and the number of pushes is halved. From the user point of view, in contrast to active updates where a user's device needs to be explicitly activated to synchronize, passive updates only incur minor battery drain as the user's device radio is already powered up and connected to the search engine.

As we show in more detail in Section 4.5, each push shown in Figure 12(a) corresponds to the datacenter transferring approximately 1MB of additional data. Given the total number of pushes in Figure 12(a), active and passive updates require the datacenter to transfer an additional 680GB and 390GB, respectively. Note that the transfer of this data is distributed over the whole lifetime of a trending event. Given that trending events can last from several hours to tens of hours, this additional bandwidth does not stretch the bandwidth limits of the search backend.

4.5 Trending Content Size Effect

Figures 10 and 11 show the query volume reduction when every single trending search query and search result is pushed. To conclude our tradeoff analysis, we now analyze the PocketTrend's sensitivity on the size of the content pushed to users. We assume PocketTrend employs *UpdatesOnly*; similar results hold for active updates, but are not shown in the interest of space.

Figure 13 shows the overall query volume when the 10, 50, 100, and 1000 most popular trending search queries along with the top 10 most clicked search results for each trending query are pushed (labeled as *PT-** in the figure). As a reference, the results when all trending search queries (Table 3) and their corresponding search results are pushed is also shown (*PT-Unlimited*). Surprisingly, caching 1000 queries (*PT-1000*) provides most of the benefits of pushing every single trending query and search result (*PT-Unlimited*). As a result, PocketTrend can achieve its top performance while pushing a limited number of trending search queries ($\approx$ 1000) and results. Reducing the size of the pushed content to 100 entries significantly reduces the performance gain, while pushing the top 50 entries can halve PocketTrend's performance gain.

Considering that an average search result requires roughly 500 bytes (i.e., URL and data snippet), storing the top 10 search results (i.e., the first page of search results) for a trending query, requires about 5 kB. Hence, pushing 1000 trending search queries corresponds to pushing roughly 5 MB. Note that if PocketTrend were to push all trending search queries and search results (Table 3), more than 135 MB would be used. The size of the trending search content can be further reduced by leveraging data compression techniques. Given that the set of queries and search results pushed to end users relates to the same event, there is significant overlap across the 1000 entries. We experimentally verified that a simple compression algorithm based on bzip2 [27, 31] can achieve a significant compression ratio ranging from 4.5x to 5x. This means that the trending search content can be compressed from 5 MB to just 1 MB. This data size is less than the size of a simple application update on a smartphone today, and it could take place over WiFi links when the cost of cellular bandwidth is a concern.

5. RELATED WORK

Discovering trends in search is a well-studied problem [9]. Commercial search engines already offer products such as Google Trends [6]. Our trend detection approach leverages well-known techniques for trend analysis [20, 24], and is not the main contribution of this work. Instead, the contributions of this work lie on the findings of the search log analysis that show how mobile users search for trending topics. Specifically, the way these findings are leveraged to enable PocketTrend to automatically detect search content related to a trending event, and more importantly to efficiently push this content to users without significantly increasing the datacenter workload is the core contribution of this work.

Previous work on search engine optimization has focused primarily on server-side and client-side caching. In server-side caching [2, 29, 16, 7, 11], search results for popular queries are cached in the search backend. Incoming queries that are in the cache can be answered faster than other queries as there is no need to access the index, and perform any ranking in real-time. The reduction in user response time can be significant when the delay due to search backend's computation is the bottleneck between the user and the datacenter. This tends to be the case when users access search engines through very high speed links (e.g., desktops). Yet, as more and more people use their mobile devices to access search engines through slow cellular links that have high setup times [8], the bottleneck shifts from the datacenter to the cellular links. In this case, the benefit of server-side caching techniques is reduced.

Researchers have been exploiting client-side caching techniques to enable faster user experience in the case of web browsing [19, 30, 5, 17, 28], ad delivery [21], and more recently search result delivery [15]. Koukoumidis et al. [15] showed that a small set of queries and search results represents a significant fraction of the mobile query volume. By analyzing mobile search logs on a daily basis, authors identify the part of the search index that is most often accessed, and use it to create a search engine that lives on the mobile device, and it is able to instantly answer about 60% of the queries an individual user submits. The search results stored on the user's mobile device are only updated nightly when the phone is charging and connected to WiFi. Even though quite effective, this work is only limited to relatively static search content that can be predicted through periodic search log mining. This system cannot address unpredicted query volume spikes due to trending events taking place throughout the day as the necessary trending search content will not be available on the users' mobile devices. Moreover, the authors blindly update every user assuming that the updates take place during off-peak times (e.g., overnight), and over WiFi connections. This approach, in the case of trending events, would require the datacenter to simultaneously update hundreds of millions of users during peak time, creating a larger problem for the search backend than the one we solve.

Conversely, PocketTrend automatically detects trending events in real time, and identifies the search content associated to these events. The trending search content is then intelligently pushed either actively to a subset of users that with high probability will

search for the trending event, or passively to any user that reaches the search engine after a trending event has been detected and is still active. In that way, PocketTrend addresses trending query volume spikes, without stressing even further the search engine's backend.

Unpredicted workload spikes is a reality for most web services [24], and are not limited to search engines [26]. As a result, there have been a lot of efforts to address these challenges directly at the networking level within datacenters [10, 22, 1]. These techniques are orthogonal to our work.

6. CONCLUSION

We have analyzed 21 million mobile search queries to understand how trending search topics are formed, and how they evolve over time. Based on our findings, we have designed and evaluated PocketTrend, a new architecture that is capable of servicing user queries related to trending events locally, improving both the user experience and the datacenter query load. Our evaluation using real mobile search logs, showed that in the presence of a trending event, up to 13%–17% of the overall traffic can be eliminated from the datacenter, impacting as many as 19% of all users.

Acknowledgements

We thank the reviewers for their valuable suggestions. Gennady Pekhimenko is supported by a Microsoft Research Fellowship and a Qualcomm Innovation Fellowship.

7. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proc. of SIGCOMM*, pages 63–74, 2010.

[2] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. of SIGIR*, pages 183–190, 2007.

[3] R. Baeza-yates, F. Junqueira, V. Plachouras, and H. F. Witschel. Admission policies for caches of search engine results. In *Proc. of the 14th String Processing and Information Retrieval Symposium*, volume 4726 of *LNCS*, pages 74–85, 2007.

[4] R. Baeza-Yates and F. Saint-Jean. A three level search engine index based in query log distribution. In *Proc. of the 10th String Processing and Information Retrieval Symposium*, volume 2857 of *LNCS*, pages 56–65, 2003.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of IMC*, pages 280–293. ACM, 2009.

[6] H. Choi and H. Varian. Predicting the present with google trends. *Economic Record*, 88:2–9, 2012.

[7] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, Jan. 2006.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proc. of MobiSys*, 2010.

[9] N. G. Golbandi, L. K. Katzir, Y. K. Koren, and R. L. Lempel. Expediting Search Trend Detection via Prediction of Query Counts. In *Proc. of WSDM*, 2013.

[10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proc. of SIGCOMM*, pages 51–62, 2009.

[11] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. Prefetching Query Results and Its Impact on Search Engines. In *Proc. of SIGIR*, 2012.

[12] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[13] M. Kamvar and S. Baluja. A large scale study of wireless search behavior: Google mobile search. In *CHI*, 2006.

[14] M. Kamvar, M. Kellar, R. Patel, and Y. Xu. Computers and iphones and mobile phones, oh my! In *WWW*, 2009.

[15] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger. Pocket Cloudlets. In *ASPLOS*, 2011.

[16] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proc. of WWW*, pages 19–28, 2003.

[17] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. PocketWeb: instant web browsing for mobile devices. In *ASPLOS*, pages 1–12, 2012.

[18] H. Ma and B. Wang. User-aware Caching and Prefetching Query Results in Web Search Engines. In *Proc. of SIGIR*, 2012.

[19] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the web. In *Proc. of INET*, 1998.

[20] M. Mathioudakis and N. Koudas. TwitterMonitor: trend detection over the twitter stream. In *Proc. of SIGMOD*, pages 1155–1158, 2010.

[21] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: can advertising systems afford it? In *Proc. of EuroSys*, pages 267–280, 2013.

[22] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. I. T. Rowstron. Everest: Scaling Down Peak Loads Through I/O Off-Loading. In *OSDI*, pages 15–28, 2008.

[23] Pyevolve. Machine Learning - Text feature extraction (tf-idf). http://pyevolve.sourceforge.net/wordpress/?p=1589, 2014.

[24] K. Radinsky and E. Horvitz. Mining the Web to Predict Future Events. In *Proc. of WSDM*, 2013.

[25] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[26] A. Saha and V. Sindhwani. Learning Evolving and Emerging Topics in Social Media: A Dynamic Nmf Approach with Temporal Regularization. In *Proc. of WSDM*, 2012.

[27] J. Seward. Bzip2 V. 1.0.6. http://www.bzip.org/, 2010.

[28] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proc. of WWW*, pages 31–40, 2012.

[29] Y. Xie and D. R. O'Hallaron. Locality in search engine queries and its implications for caching. In *INFOCOM*, 2002.

[30] L. Yin and G. Cao. Adaptive power-aware prefetch in wireless networks. *IEEE Trans. on Wireless Comms*, 2004.

[31] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inf. Theory*, 1977.