Secure and Energy Efficient Prefetching Design for Smartphones

GAO Shang*, PENG Zhe*, XIAO Bin*, SONG Yubo*†,[1]
*Department of Computing, The Hong Kong Polytechnic University
†School of Information Science and Engineering, Southeast University
{cssgao, cszpeng, csbxiao}@comp.polyu.edu.hk, {songyubo}@seu.edu.cn

***Abstract*—Energy efficient prefetching systems for smartphones can greatly reduce energy consumption and data transmission, and maintain the timely response when information is prefetched. However, the proxy structure of the system can cause security problem to reveal private information to the third party. The end-to-end encryption (SSL) in traditional prefetching systems cannot solve the security problem in this new, complex energy efficient prefetching system. In this paper, we propose Secure and Energy Efficient Prefetching (SEEP) to meet the security requirement of HTTPS connections and to save smartphone's energy consumption and data transmission. The new design of SEEP includes two parts: the local proxy on the smartphone to verify the validity of prefetched responses, and the remote proxy (e.g. on the cloudlet) to store encrypted prefetched responses. SEEP is transparent to both smartphones and web servers, which does not need to change today's Browser/Server framework. Security analysis shows that SEEP protects the confidentiality of requests and responses, and is able to resist replay attack from malicious proxy. Experimental results show that the proposed system consumes 25% less energy and 95% less data when prefetching 10 outbound webpages than the traditional prefetching system in Wi-Fi networks.**

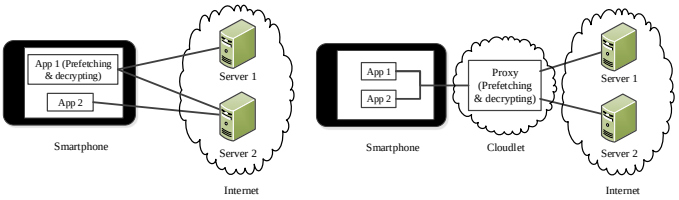
I. Introduction

The prefetching technique used on smartphones can reduce the delay of responses and provide good user experience [1]. For instance, web browser or online reader prefetches and stores subsequent webpages, and presents them immediately to users when requested. However, the prefetched responses can drain a smartphone's battery quickly and cause much wasted data transmission since the smartphone needs to receive and store all prefetched responses, as depicted in Fig. 1-a.

Prefetching technique can exploit a proxy-based architecture [2] to greatly reduce the energy consumption and delay. In HTTP connections, the proxy-based prefetching system is energy efficient by storing the prefetched responses on the proxy to reduce the energy and data cost, as depicted in Fig. 1-b. Therefore, the smartphone only needs to receive the prefetched responses when users request for them.

Such proxy-based energy efficient prefetching system will cause security problems in HTTPS (SSL) connections. HTTPS connection uses sequence numbers to resist replay attacks. However, in energy efficient prefetching system, the sequence number of prefetched responses are based on previous prefetching requests rather than the current user's requests.

[1]This paper is partially supported by HK PolyU G-YBAD and G-YBJV.



(a) Traditional prefetching system. Prefetched responses are stored and decrypted on the smartphone.

(b) Energy efficient prefetching system. Prefetched responses are stored and decrypted on the proxy.

Fig. 1. Security problems in energy efficient prefetching system.

Therefore, all prefetched responses will be regarded as replayed packets and dropped automatically. One simple solution is to build the proxy as a man-in-the-middle proxy (mitmproxy) [3]. Mitmproxy will decrypt the message from server/application, and send to application/server after encrypting, as depicted in Fig. 1-b. However, mitmproxy brings another security problem: the plaintext of the requests and responses will be revealed to the third party. This implementation offers no level of security when the proxy is untrusted. To the best of our knowledge, no previous work has addressed these security problems in energy efficient prefetching system.

We introduce SEEP system, a proxy-based framework to reduce energy consumption and data transmission, and meet the security requirements of HTTPS connections. The proposed framework involves two parts: the local proxy on the smartphone and the remote proxy (e.g. on the cloudlet). For instance, a user can set up the remote proxy in the Intranet of his company for all company users without any changes on today's Browser/Server framework. Originally, a smartphone needs to establish multiple connections to servers and receive multiple prefetched responses, or needs to reveal their private information to the proxy. In contrast, the local proxy in SEEP establishes only one connection with the remote proxy, and receives only one response. All requests and responses are protected by end-to-end encryption between the smartphone and servers in the SEEP system. The remote proxy is unable to decrypt them without the shared key. The two-proxy design in SEEP can efficiently reduce the energy and data cost, and meet the security requirements of HTTPS connections.

We formalize data exchange procedures in the SEEP system for security analysis, and build a model to evaluate the energy and data cost of a smartphone. The result of security analysis

shows that our SEEP system satisfies the confidentiality and robustness requirements of HTTPS connections at the same time. The energy and data cost model shows that SEEP system saves energy and data in two ways: reducing the cost of establishing connections and eliminating the cost of receiving prefetched responses.

We implement a prototype of the SEEP system, including an application on a smartphone as the local proxy, and a remote proxy on the cloudlet. We also evaluate the performance of the SEEP system in real-world experiments. The experimental results show that the SEEP system is much more energy saving and data reducing than the traditional prefetching system.

The rest of the paper is organized as follows. Section II introduces traditional prefetching model, energy efficient prefetching model and its security problems. In Section III, we present the detailed designs of the SEEP system, including its architecture, and security designs in the local and remote proxies. Section IV analyzes security requirements in the SEEP system and Section V analyzes its performance theoretically. The experimental evaluation is shown in Section VI. We summarize the related work in Section VII. Finally, we conclude this paper in Section VIII.

II. Problem Statement

A. Traditional Prefetching Model

In the traditional prefetching model, the browser application first establishes multiple connections with servers when browsing webpages. Second, the application generates and sends user's requests to the servers. Third, after receiving the responses from the servers, the application reconstructs the webpage, generates and sends prefetching requests for the outbound links to the servers. Forth, the application receives the prefetched responses and reconstructs the prefetched webpages. Finally, when a user requests for an outbound link, the application presents the according prefetched webpage to the user immediately.

B. Energy Efficient Prefetching Model

In the energy efficient prefetching model, when a user browses webpages, the browser application first establishes one connection with the proxy. Second, the application generates and sends user's requests to the proxy. Third, the proxy forwards these requests to the servers, and forwards the responses from the servers to the application. Forth, after receiving the responses, the application reconstructs the webpage, generates and sends prefetching requests for the outbound links to the proxy. Fifth, the proxy forwards these requests to the servers, receives and stores prefetched responses from the servers. Finally, when the user requests for an outbound link, the application fetches the according prefetched webpage from the proxy and presents it to the user immediately.

C. Security Problems in Energy Efficient Prefetching Model

Energy efficient prefetching model introduces two security problems. First, if we use this model in HTTPS connections, all prefetched responses will be regarded as replayed packets and dropped automatically, since the sequence numbers of them are based on previous prefetching request rather than current user's request. Second, if we apply mitmproxy [3] to solve the mentioned problem, the plaintext of the requests and responses will be revealed to the proxy. Since the proxy can be owned by untrusted third party, user's private information will be leaked.

Our goal is to enhance the security in the energy efficient prefetching model. This design should not only support HTTPS connections (distinguishing replayed packets from legal packets), but also maintain the confidentiality of the requests and responses (not reveling the plaintext of them to the proxy).

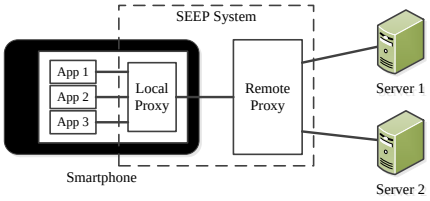


Fig. 2. The architecture of SEEP, including the local proxy on the smartphone and the remote proxy on the cloudlet.

III. System Design

A. SEEP Architecture

The framework of SEEP system consists of a local proxy and a remote proxy as depicted in Fig. 2. The local proxy is an application that runs on a smartphone, and the remote proxy is a proxy server that works on the cloudlet. The local proxy is in charge of interpreting contexts and generating prefetching requests, and the remote proxy is responsible for storing prefetched responses from servers.

The local proxy first establishes a connection with the remote proxy, and redirects the smartphone's network connections to it. When the application sends the request, the local proxy forwards this request to the remote proxy, and the remote proxy searches its response cache for the corresponding response. If there is no response for this request, the remote proxy forwards the request to the server. After the servers send responses to the application through the remote proxy and local proxy, the local proxy interprets the responses, generates and sends prefetching requests to the remote proxy. After receiving server's responses to these prefetching requests, the remote proxy stores these prefetched responses in its response cache, and forwards the corresponding responses to the local proxy when the user requests for them. SEEP is transparent to both applications and servers, which does not need any changes on today's Browser/Server framework.

B. Security Design in Local Proxy

The major functions of the local proxy are generating prefetching requests and verifying the validity of responses. To achieve these goals, the local proxy includes four major components, i.e. connection redirector, context interpreter, request generator, and message forwarding. Their relationships are depicted in Fig. 3.

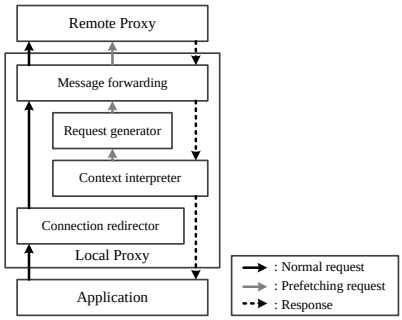


Fig. 3. Four major components in the local proxy.

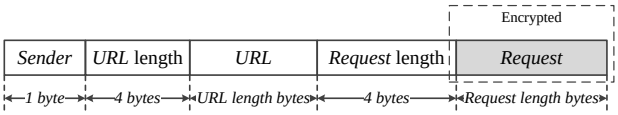


Fig. 4. The structure of encrypted prefetching request message.

1) Connection redirector: Since some applications may not provide interfaces to use proxy, it is challenging to make applications send messages to the local proxy first. Building the local proxy as a transparent proxy is impractical in this scenario, since no message can be sent out once the smartphone sets its gateway to 127.0.0.1. Therefore, the local proxy should not only be transparent to applications and servers, but also be able to set bypass IP addresses.

We fulfill the redirection function based on iptables. It redirects all connections to the local proxy except the connection with the remote proxy. Meanwhile, it also allows users to set up bypass IP address which they do not wish to use SEEP.

2) Context interpreter: In order to identify the information which needs to be prefetched, the local proxy should be able to interpret the context of responses and get all outbound links of the current webpage. One challenge is that the response messages are all encrypted under HTTPS connection. The local proxy is unable to get the plaintext without the session key. To solve this problem, we refer the code of mitmproxy [3] and build the local proxy as a man-in-the-middle proxy. The basic idea is pretending to be the server to the client, and pretending the client to the server [3]. Therefore, we are able to obtain the session keys with the application and servers, and decrypt response messages.

Man-in-the-middle proxy brings another challenge. Applications can reject SSL handshake when the local proxy fails to provide trusted certificates. The solution is to make the local proxy a trusted CA that generates interception certificates [3]. We register the local proxy as a trusted CA on the smartphone manually.

Another challenge is to resist replay attack from malicious remote proxy. Since the sequence numbers of the prefetched responses are based on the previous requests, we cannot distinguish replayed packets based on the current normal request. Context interpreter will preserve the sequence numbers of prefetching requests, and identify replay packets based on previous sequence numbers.

3) Request generator: Since the plaintext of the prefetching request should not be revealed to the remote proxy, the local proxy is in charge of generating prefetching requests. One challenge is that the remote proxy is unable to tell which response is to the corresponding request because the request and response are all encrypted. Therefore, we use a one-byte sender field to tell whether this request is sent by the application or the remote proxy, and add the URL of the requested webpage in front of the prefetching request. The structure is depicted in Fig. 4. In this way the remote proxy is able to identify the response to each request.

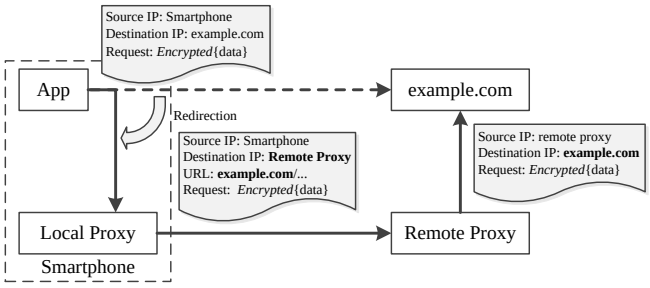


Fig. 5. SEEP forwards a request.

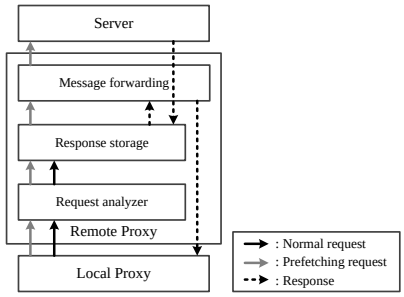


Fig. 6. Three major components in the remote proxy.

4) Message forwarding: One challenge is the server's information may be lost because the IP address of the destination is changed to the remote proxy's after forwarding. The remote proxy is unable to get the server's IP address based on the destination IP address. Therefore, the remote proxy uses the URL field to get the server's IP address, as depicted in Fig. 5.

C. Security Design in Remote Proxy

To support HTTPS connections, the major function of the remote proxy is to identify the according prefetched responses to the user's request. The remote proxy includes three major components, e.g. request analyzer, response storage, and message forwarding, as depicted in Fig. 6. Since the procedures of message forwarding on the remote proxy are similar to those on the local proxy, we skip this part due to space constraints.

1) Request analyzer: When the request arrives, the remote proxy first identifies the sender of request based on the sender field. If the request is sent by applications, the response should be forwarded to the local proxy immediately. Otherwise the response would be stored in the response storage to save the energy and data of the smartphone. Second, the request analyzer splits the URL and the request of each message and forwards them to the response storage for further processing.

2) Response storage: The main job for response storage is to identify the corresponding response to each request. We

cannot only use URL to match up the responses to each request, since some applications may request for the same URL. Therefore, we maintain a cache table to identify the corresponding response to each request in response storage. The key of this table is the combination of sender ID and URL, and the value is the encrypted response.

IV. Security Analysis

A. Data Exchange

The data exchange in SEEP system involves four parties, i.e. the application, the local proxy, the remote proxy and the server. The SEEP system should not reveal the plaintext of the requests and responses to the remote proxy, and be able to resist replay attack since the remote proxy can be owned by untrusted third party. The notations used in data exchange in depicted in Table I

TABLE I. Notations used in SEEP system

Symbol	Description
APP	Application
LP	Local proxy
RP	Remote proxy
SER	Server
req_i	ith request
res_i	Response to ith request
ID_A	Identity of A
$K_{\text{A-B}}$	Symmetric session key shared between A and B
$K_{\text{A-B}}(m)$	Encrypting message m with key $K_{\text{A-B}}$
$Seq_{\text{A-B}(m)}$	Sequence number of message m under A-B connection

After establishing an HTTPS connection, the secret session keys, $K_{\text{APP-LP}}$ between the application and the local proxy and $K_{\text{LP-SER}}$ between the local proxy and server, will be generated and processed by both sides. The procedures of key exchange are protected by SSL protocol. Since we are not focus on the security analysis of SSL, we skip the procedures of key exchange and regard nobody is able to obtain the session keys except the key possessors.

We analyze data exchange procedures in the SEEP system. Here we ignore some information, e.g. URL, since we focus on the security of requests and responses. First, the application generates the request to get the webpage from server. After the local and remote proxies forward this request, the server sends the response to the remote proxy, and the remote proxy stores this response in its cache table:

Step 1 *APP* $\rightarrow$ *LP* : $K_{APP\text{-}LP}(Seq_{APP\text{-}LP}(req_1), req_1)$

Step 2 *LP* $\rightarrow$ *RP* : $ID_{APP}, K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_1), req_1)$

Step 3 *RP* $\rightarrow$ *SER* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_1), req_1)$

Step 4 *SER* $\rightarrow$ *RP* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_1), res_1)$

The remote proxy forwards this response to the local proxy immediately. The local proxy verifies the sequence number of this response and forwards it to the application.

Step 5 *RP* $\rightarrow$ *LP* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_1), res_1)$

Step 6 *LP* $\rightarrow$ *APP* : $K_{APP\text{-}LP}(Seq_{APP\text{-}LP}(res_1), res_1)$

After step 5 and 6, the application is able to get the corresponding response to its request. Meanwhile, the local proxy interprets the response and generates prefetching requests.

Step 7 *LP* $\rightarrow$ *RP* : $ID_{LP}, K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_2), req_2)$

Step 8 *RP* $\rightarrow$ *SER* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_2), req_2)$

Step 9 *SER* $\rightarrow$ *RP* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_2), res_2)$

We repeat step 7 to 9 to send multiple prefetching requests. Since the prefetching requests are not generated by the application, the remote proxy stores these responses and forwards to the local proxy when the application requests for them.

Step 10 *APP* $\rightarrow$ *LP* : $K_{APP\text{-}LP}(Seq_{APP\text{-}LP}(req_{2'}), req_{2'})$

Step 11 *LP* $\rightarrow$ *RP* : $ID_{APP}, K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_{2'}), req_{2'})$

Step 12 *RP* $\rightarrow$ *LP* : $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_2), res_2)$

Step 13 *LP* $\rightarrow$ *APP* : $K_{APP\text{-}LP}(Seq_{APP\text{-}LP}(res_2), res_2)$

When the local proxy receives the response in step 12, it should verify $Seq_{LP\text{-}SER}(res_2)$. Notice that $Seq_{LP\text{-}SER}(req_{2'})$ is different from $Seq_{LP\text{-}SER}(req_2)$, and the local proxy should verify $Seq_{LP\text{-}SER}(res_2)$ based on $Seq_{LP\text{-}SER}(req_2)$ since $Seq_{LP\text{-}SER}(res_2)$ is generated by $Seq_{LP\text{-}SER}(req_2)$. After step 10 to 13, the local proxy is able to interpret the new response (res_2) and generate new prefetching requests (repeating step 7 to 9).

B. Security Analysis

Our proposed SEEP system should satisfy two security requirements: the plaintext of requests and responses should not be revealed to the remote proxy (confidentiality requirement), and the local proxy should be able to distinguish replayed responses from legal responses (robustness requirement).

1) Confidentiality: We first analyze the confidentiality requirement. During the procedures of data exchange in the SEEP system, the remote proxy is able to obtain ID_{APP}, ID_{LP}, $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_i), req_i)$, $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_i), res_i)$, $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_j), req_j)$, $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(res_j), res_j)$, and $K_{LP\text{-}SER}(Seq_{LP\text{-}SER}(req_{j'}), req_{j'})$. Since req_i, res_i, req_j, $req_{j'}$, res_j are protected by end-to-end encryption, the local proxy can only get the plaintext of ID_{APP} and ID_{LP}. The remote proxy cannot decrypt requests and responses without $K_{LP\text{-}SER}$.

2) Robustness: We analyze the robustness of SEEP system against replay attack. The remote proxy is able to replay previous request messages to the server, and replay previous response messages to the local proxy. However, by verifying the sequence number field in SSL protocol, the server can easily identify replayed requests and reject them because all requests are real-time. Meanwhile, on smartphone side, the local proxy verifies $Seq_{LP\text{-}SER}(res_i)$ based on $Seq_{LP\text{-}SER}(req_i)$. Messages which fail in this verification will be regarded as replayed packets and dropped automatically. In this way, our SEEP system meet the robustness requirement.

V. Energy and Data Cost Analysis

We consider a set of servers $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ that the browser needs to connect to, a set of request messages $\mathcal{MS} = \{ms_1, ms_2, ..., ms_n\}$ sent by the smartphone, and a set of response messages $\mathcal{MR} = \{mr_1, mr_2, ..., mr_n\}$ received from servers or the remote proxy during webpage browsing. We denote the energy cost of establishing a connection with server s_i by $E_{conn}(s_i)$, the energy cost of sending a request message ms_j by $E_{req}(ms_j)$, and the energy cost of receiving a response message mr_k by $E_{res}(mr_k)$. The data cost is similar to the energy cost. We denote the data cost of establishing connections with server s_i by $D_{conn}(s_i)$, the data cost of a request message ms_j by $D_{req}(ms_j)$, and the data cost of a response message mr_k by $D_{res}(mr_k)$. When browsing a webpage, the application sends p requests and receives q responses.

We first calculate the energy consumption and data transmission under the traditional prefetching system. A smartphone will establish multiple connections and receive all prefetched responses from servers. The total energy consumption E_{TP} of during webpage browsing is the sum of three parts energy cost: establishing m connections, sending p requests, and receives q responses.

$$E_{TP} = \sum_{i=1}^{m} E_{conn}(s_i) + \sum_{i=1}^{p} E_{req}(ms_i) + \sum_{i=1}^{q} E_{res}(mr_i) \quad (1)$$

The calculation of the data transmission D_{TP} under traditional prefetching system is similar to that of the energy consumption.

$$D_{TP} = \sum_{i=1}^{m} D_{conn}(s_i) + \sum_{i=1}^{p} D_{req}(ms_i) + \sum_{i=1}^{q} D_{res}(mr_i) \quad (2)$$

Moreover, we evaluate the energy and data cost of the smartphone under the SEEP system when performing prefetching operation. In the SEEP system, a smartphone will establish only one connection with the remote proxy, and receive only one response (the response to the user's request). The $\sum_{i=1}^{p} E_{req}(ms_i)$ part is almost the same as traditional prefetching system, since the volume of the sender and URL fields is pretty small comparing to the whole request. Therefore, the energy cost under SEEP system E_{SEEP} can be calculated as follows:

$$E_{SEEP} = E_{conn}(s_1) + \sum_{i=1}^{p} E_{req}(ms_i) + E_{res}(mr_1) \quad (3)$$

The data transmission under the SEEP system D_{SEEP} can be calculated as follows:

$$D_{SEEP} = D_{conn}(s_1) + \sum_{i=1}^{p} D_{req}(ms_i) + D_{res}(mr_1) \quad (4)$$

Generally speaking, the SEEP system saves energy consumption and data transmission in two ways: reducing the cost of establishing multiple connections to servers by maintaining one connection with the remote proxy, and reducing the wasted data transmission by prefetching and storing responses on the remote proxy.

VI. Experiment

A. Implementation

We implement a prototype of our proposed SEEP system, including both the local proxy on the smartphone device and the remote proxy on the cloudlet.

The local proxy is implemented on Android OS 4.2 based on JAVA. Since modifying iptables requires root permission, we give root permission to the local proxy before using the SEEP system. The local proxy first responds with "200 Connection Established" once it receives "Connect" request. After the application initiates SSL handshake with server name indication (SNI), the local proxy establishes an SSL connection with the server using the SNI hostname, and the server responds with the original certificate. Finally, the local proxy generates the interception certificate with CN field and SAN field and establishes SSL connection with the application.

The remote proxy is implemented on Windows 7 based on C++. We use a dynamic-sized hashtable to ensure the efficiency of the cache table, and keep its load factor under 0.75. The key is the combination of "Sender ID" and "URL". When a new request from the application arrives, the remote proxy deletes all records with the same sender to save space.

B. Setup

We evaluate the performance of the SEEP under WLAN on a smartphone equipped with Android OS 4.2, and place the remote proxy under the same AP with the smartphone. To get a fine-grained measurement of energy consumption on the smartphone device, we use a power meter to measure the real-time current of the smartphone. For the data consumption, we use the Data Usage function of the smartphone to get the size of data consumed by each application.

We first evaluate the energy saving performance of the SEEP. We compare the energy cost of prefetching 1 to 10 outbound webpages in 40 seconds between the SEEP and traditional prefetching system. Here we connect Google News [2] with the Browser, and uses the ten outbound webpages in Top Stories and World sections to measure their performances. The size of the news webpage is 0.83MB, and the sizes of ten outbound webpages are 3.22MB, 3.54MB, 1.37MB, 1.62MB, 3.32MB, 1.39MB, 1.03MB, 1.19MB, 0.66MB, and 1.19MB.

Moreover, we evaluate the data saving performance of the SEEP. Similar to the experiment of energy comparison, we also use 1 to 10 outbound webpages of Google News to compare the data consumption between the SEEP and traditional prefetching system.

C. Experimental Result

Energy cost. Fig. 7 shows that energy comparison of the SEEP does not increase dramatically with the number of prefetching webpages. It is because in the SEEP system, the

[2] https://news.google.com/news/

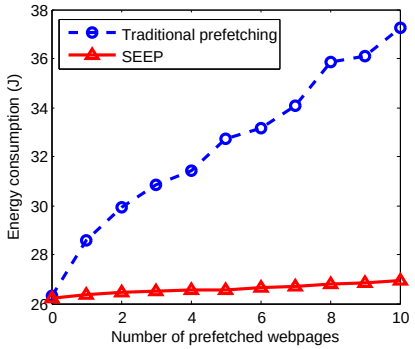


Fig. 7. The energy cost comparison between SEEP and the traditional prefetching system in 40 seconds.

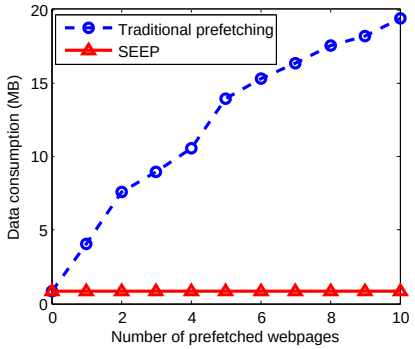


Fig. 8. The data cost comparison between SEEP and the traditional prefetching system.

local proxy only establishes one connection ($E_{conn}(s_1)$ part) and receives one response ($E_{res}(mr_1)$). The energy cost of sending requests is trivial to the total energy cost. Generally speaking, the SEEP system can reduce 25% energy when prefetching 10 outbound webpages.

Data cost. Fig. 8 shows that the data cost of SEEP almost remains the same when the number of prefetching webpages increases. It is because the local proxy only send multiple requests ($\sum_{i=1}^{p} D_{req}(ms_i)$ part in Equation (4)), and these requests are trivial to the total data cost (about 2.34 KB pre request). Generally speaking, SEEP only consumes the data of the first webpage, and reduces 95% data usage when prefetching 10 outbound webpages.

VII. Related Work

Prefetching techniques have been used in many areas of computer system. Here we focus on previous approaches in network prefetching [4], [5], [6], [7].

Informed Mobile Prefetching (IMP) [4] is a system where developers use an API to specify a method to fetch a network object, or cancel the prefetching operation. The IMP system can optimize for data, power, and performance based on a hint of the network object and the later action for this object.

Besides reducing network consumption, previous approaches improve web caching in applications [8], [9]. Recent study [8] investigates caching efficiencies from the perspective of individual handsets, and another study [9] investigates the potential for caching video content in cellular networks. They indicate that the redundant traffic during data transferring can be saved.

There are also previous approaches to reduce both energy consumption and delay of mobile browser [2], [10]. PARCEL [2] introduces a proxy to reduce the cost of establishing connections while saving the time of loading webpages. Though PARCEL does not use prefetching techniques, it splits functionality between the mobile device and the proxy on their respective strengths.

VIII. Conclusion

Prefetching techniques require large amounts of date interacting to get outbound webpages, which can drain a smartphone's battery quickly and consume great data usage. Meanwhile, some private information can be revealed to the third party when a proxy is involved in the energy efficient prefetching system. We propose the SEEP system to reduce the energy and data cost on a smartphone, and meet the security requirements. The SEEP can successfully reduce the energy and data cost of establishing connections and receiving responses. Moreover, it can ensure the confidentiality of information and resist replay attack. We analyze the confidentiality and robustness requirements of the SEEP and build models for energy consumption and data transmission. We also evaluate the performance of the SEEP under today's network servers. The experimental results show that the SEEP reduces great energy and data than the traditional prefetching system.

References

[1] "prefetching." https://en.wikipedia.org/wiki/Instruction_prefetch.

[2] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, and S. Sen, "PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction," in *Proc. of the ACM 10th international conference on emerging networking experiments and technologies 2014 (CoNEXT)*, pp. 325–336, 2014.

[3] A. Cortesi, "mitmproxy." https://mitmproxy.org/.

[4] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetching," in *Proc. of the ACM 10th international conference on Mobile systems, applications, and services 2012 (MobiSys)*, pp. 155–168, 2012.

[5] L. Ravindranath, S. Agarwal, J. Padhye, and C. Riederer, "Procrastinator: pacing mobile apps' usage of the network," in *Proc. of the ACM 12th international conference on Mobile systems, applications, and services 2014 (MobiSys)*, pp. 232–244, 2014.

[6] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *Proc. of the ACM international joint conference on Pervasive and ubiquitous computing 2013 (UbiComp)*, pp. 275–284, 2013.

[7] S. Sundaresan, N. Magharei, N. Feamster, and R. Teixeira, "Accelerating last-mile web performance with popularity-based prefetching," pp. 303–304, 2012.

[8] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," in *Proc. of the ACM 10th international conference on Mobile systems, applications, and services 2012 (MobiSys)*, pp. 127–140, 2012.

[9] J. Erman, A. Gerber, K. Ramadrishnan, S. Sen, and O. Spatscheck, "Over the top video: the gorilla in cellular networks," in *Proc. of the ACM SIGCOMM conference on Internet measurement conference 2011 (IMC)*, pp. 127–136, 2011.

[10] J. Li, Z. Peng, B. Xiao, and Y. Hua, "Make smartphones last a day: Pre-processing based computer vision application offloading," in *Proc. of the IEEE SECON 2015 conference*, pp. 462–470, 2015.