



# Automated Energy Optimization of HTTP Requests for Mobile Applications

# Autores

Ding Li, Yingjun Lyu  
Jiaping Gui  
William G. J. Halfond

University of Southern California  
Los Angeles, California, USA





# Introdução ao problema

O processo de fazer requisições HTTP consome uma grande quantidade de energia devido ao próprio processo de criar pacotes http, visto que o http é um protocolo de múltiplas camadas.

1. HTTP também introduz muitos dados em formato de header(entre 200B e 2KB de dados extra)
2. A criação de uma conexão HTTP requer um handshake de 3 vias e uma desconexão requer um handshake de 4 vias
3. Para evitar o overhead de abrir e fechar a conexão, o dispositivo pode gastar energia de “calda”, onde o dispositivo mantém a antena no modo ativo mesmo após a requisição HTTP acaba.



# Abordagem

O objetivo do estudo é diminuir o número de requisições HTTP feitas pelo dispositivo, o que os levou a seguinte abordagem

1. Identificar *Sequential HTTP Requests Session*(SHRSs)
2. Reescrever o código do cliente de modo a combinar esses SHRSs em um requests HTTP.



# Detecção de SHRS

Definiram um SHRS como um conjunto de invocações HTTP( $S = \{h_1, h_2, h_3\}$ ) em sequência que satisfazem :

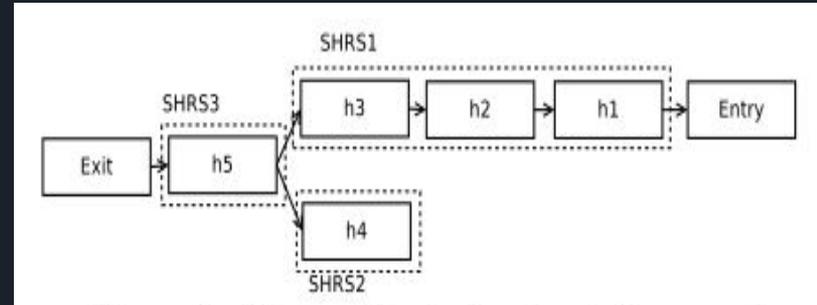
1. For any  $h_i$  and  $h_j$ , if  $i < j$ ,  $h_i$  is post dominated by  $h_j$  in the app's Control Flow Graph (CFG).
2. For any  $h_i$  and  $h_j$  where  $i < j$ , if there is another HTTP API invocation  $h'$  on a path from  $h_i$  to  $h_j$  in the CFG, then  $h' \in S$ .

```

14 public void print_html(String city)
15 {
16     URL url1, url2, url3, url4, url5;
17     URLConnection urlConnection1, urlConnection2,
        urlConnection3;
18     //query current weather
19     url1 = new URL("http://weather?city="+city);
20     urlConnection1 = url1.openConnection();
21     Parse(urlConnection1.getInputStream()); //h1
22     //query weather forecast
23     url2 = new URL("http://daily?city="+city);
24     urlConnection2 = url2.openConnection();
25     Parse(urlConnection2.getInputStream()); //h2
26     //query location info
27     url3 = new URL("http://location?city="+city);
28     urlConnection3 = url3.openConnection();
29     Parse(urlConnection3.getInputStream()); //h3
30     //query the information about the city
31     if(Cond())
32     {
33         url4 = new URL("http://information?city="+city);
34         urlConnection4 = url4.openConnection();
35         Parse(urlConnection4.getInputStream()); //h4
36     }
37     //get the rate of the city
38     url5 = new URL("http://rate?city="+city);
39     urlConnection5 = url5.openConnection();
40     Parse(urlConnection5.getInputStream()); //h5
41 }

```

## Post Dominator Tree





# Bundling de requests

Desenvolveram dois módulos para efetuar o *bundling* dos requests, **Tester** e **Operator**

- **Tester:** Conjunto de Regexes que representa a URL do primeiro request em uma SHR
- **Operator:** “Decide” qual chamada http deve ser empacotada, é chamado quando o Tester indentifica um match.



# Tester

Responsável por gerar regexes que representam as urls dos requests

O Tester é gerado a partir de dois passos:

1. Usando *Standard Alias Analysis*, é uma técnica que auxilia a procura de onde foi inicializada/alocada a String total do request.
2. Após encontrar a String total do Request, é gerada uma expressão regular com o objetivo de criar um padrão para detectar Requests parecidos.



# Operator

Responsável por descobrir os valores dos argumentos a serem passados nos requests.

## 3 Casos:

1. Quando os requests são constantes.
2. Quando os requests são semi constantes (Combinações de constantes e variáveis desconhecidas)
  - a. Cria uma base e aguarda os dados das variáveis serem consolidados para gerar o request.
3. Outros casos
  - a. Guarda relações entre variáveis dos requests e criar regex para identificar partes que não mudam nos pacotes
  - b. Dados são apresentados ao desenvolvedor para que ele possa decidir se e como vai fazer bundling



# Desafios

1. Diferenciar Invocações HTTP com os mesmos dados mas que provém de fontes diferentes.
2. Garantir que os requests HTTP são executados em ordem



# The agent HTTP APIs (AHAs)

- Tem como papel fazer otimizações no lado do cliente.
- Antes de enviar os requests, o AHA verifica se as informações pedidas estão já disponível em cache.
- Cria e envia para o proxy Bundled Requests ( Request original + CSID )
- Recebe bundles responses e certifica-se que cada resposta vai ser direcionada para o request correto.



# Proxy

- Lado de otimização do lado do servidor
- Identifica as requisições que foram empacotadas juntas (Tester + Operator)
- Envia para o server as requisições na sequência do SHRS
- Retorna o bundle de respostas para o lado do cliente



# Desafios

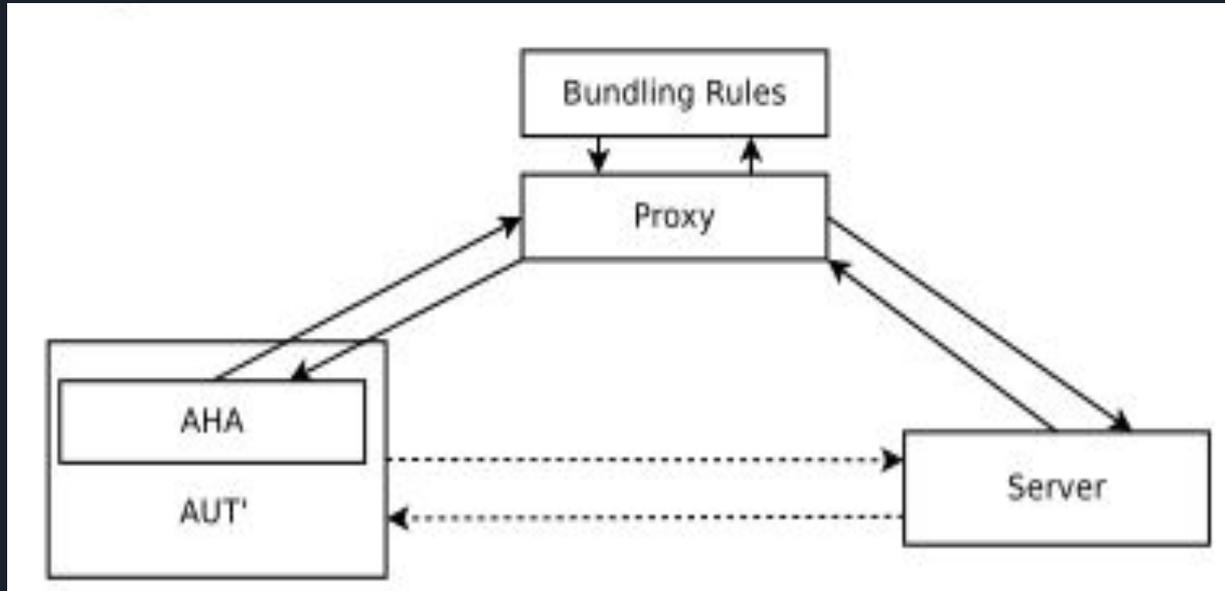
1. Diferenciar Invocações HTTP com os mesmos dados mas que provém de fontes diferentes.

**R: CSID**

2. Garantir que os requests HTTP são executados em ordem

**R: A definição de SHRSs garante que o conjunto vai ser executado em ordem**

# Runtime Workflow





# Análise

RQ 1: How much energy could be saved by using Bouquet?

RQ 2: How much manual effort is needed to augment the Bundler?

RQ 3: How long is the analysis time of Bouquet?

RQ 4: How much runtime overhead is introduced by Bouquet?

RQ 5: What is prevalence rate of SHRSs in marketplace apps?



# Testes

## Especificações:

- DELL XPS 8100, running linux mint 14. (Server + Proxy)
- Galaxy S5
- Dentre 7878 apps analisados, foram escolhidos 5 Apps que tenham SHRSs

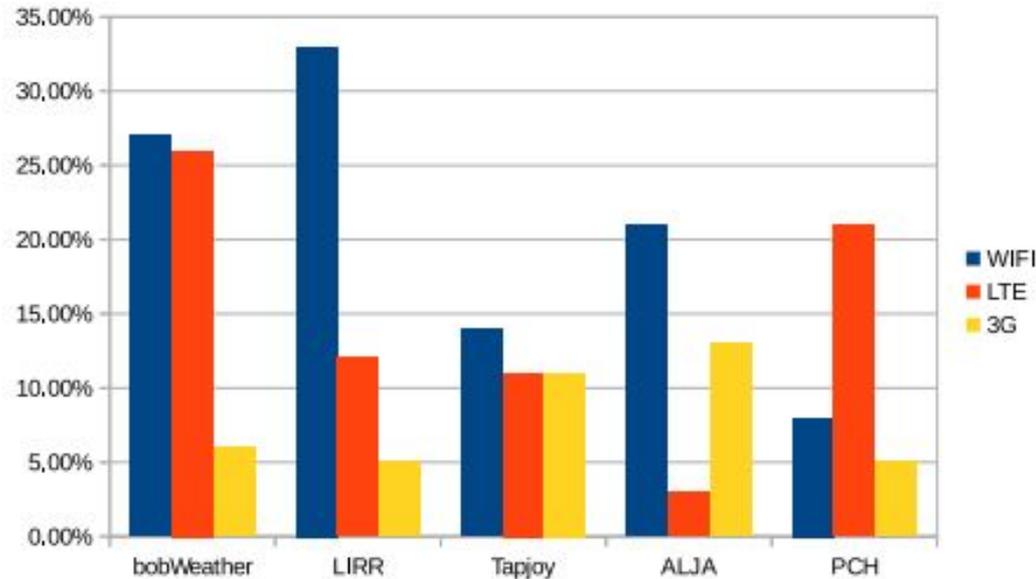


# RQ1: How much energy could be saved by using Bouquet?

**Table 1:** Description of subject apps

App	Description	#Bytecode	API	Method	Generated	Provided
bobWeather	Weather Forecasting	22,517	URLConnection	GET	88	20
LIRR	Train Schedule Checker	4,408	HttpClient	POST	88	0
Tapjoy	Rewards Tracker	84,963	URLConnection	GET	28	0
ALJA	News Portal	279,114	HttpClient	GET	24	0
PCH	Lottery	216,842	URLConnection	GET	28	0

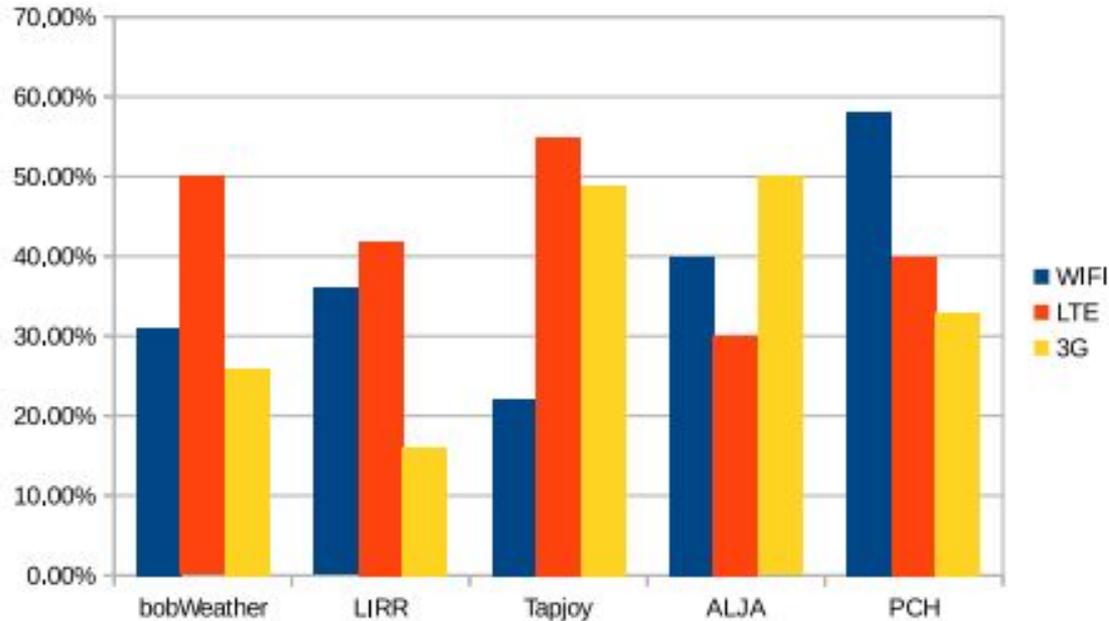
# RQ1: How much energy could be saved by using Bouquet?



**Figure 4:** Energy savings at the whole-app level.

- Média de 15% de energia reduzida em nível de aplicação

# RQ1: How much energy could be saved by using Bouquet?



**Figure 5:** Energy savings at the SHRS level

- Média de 38% de energia reduzida nas requisições

## RQ 2: How much manual effort is needed to augment the Bundler?

**Table 1:** Description of subject apps

App	Description	#Bytecode	API	Method	Generated	Provided
bobWeather	Weather Forecasting	22,517	URLConnection	GET	88	20
LIRR	Train Schedule Checker	4,408	HttpClient	POST	88	0
Tapjoy	Rewards Tracker	84,963	URLConnection	GET	28	0
ALJA	News Portal	279,114	HttpClient	GET	24	0
PCH	Lottery	216,842	URLConnection	GET	28	0

**Generated:** Comentários gerados para análise

**Provided:** Linhas escritas manualmente para criar bundles

## RQ 3: How long is the analysis time of Bouquet?

**Table 2: Analysis Time (s)**

App	Loading	Analysis	Convert	Rewrite	Total
bobWeather	1.4	4.7	29.4	2.5	38.0
LIRR	1.1	1.2	6.7	1.0	10.0
Tapjoy	2.2	5.3	20.1	7.5	35.1
ALJA	25.1	4.4	43.3	12.4	85.2
PCH	10.4	7.0	36.9	12.2	66.5

- Load android app
- Analysis to detect SHRS and generate Bundles
- Convert to Java bytecodes
- Rewrite replacing the http requests

# RQ 4: How much runtime overhead is introduced by Bouquet?

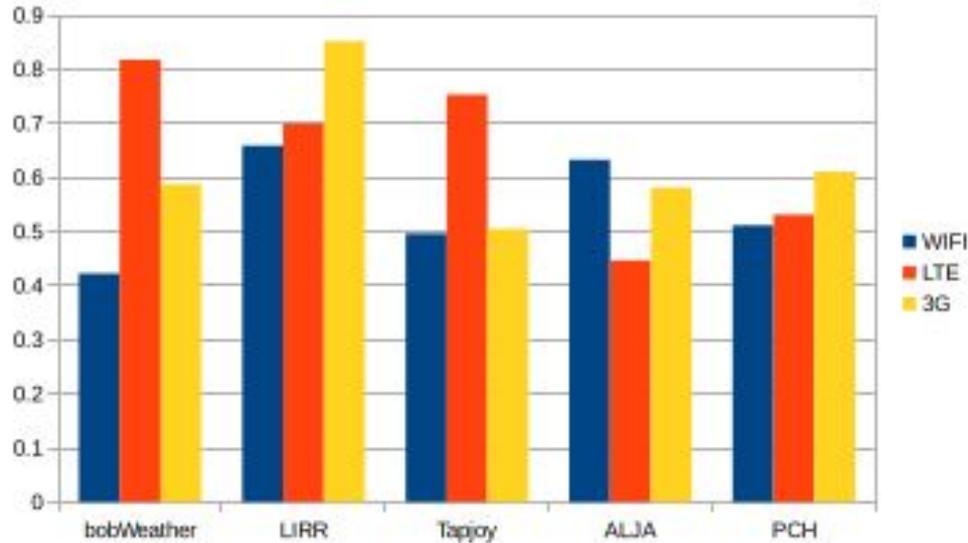


Figure 6: The runtime overhead introduced by Bouquet

- Não se teve overhead.
- Em média o custo foi 61% em relação ao tempo sem otimização.



## RQ 5: What is prevalence rate of SHRSs in marketplace apps?

- Foram coletados 7878 aplicativos da Play Store
- 206 aplicativos continham SHRSs ( 2.6% )
- A Play Store possui 1.6 milhões de aplicativos
- 2.6% representa 40 mil aplicativos



# Conclusão

- Consumo de energia é crítico para aparelhos móveis
- Abordagens recentes não levam em conta requisições https
- Redução em média de 38% de energia em requisições
- Redução em média de 15% de energia em nível de aplicação
- Possibilitando a otimização

Jailson Gomes

João Amaro

Obrigado!

An abstract graphic on the right side of the slide. It features several overlapping, dark grey, 3D-style rectangular blocks arranged in a diagonal pattern from the top right towards the bottom left. Two of these blocks are highlighted with color: one is light green and the other is blue. The background is a dark, solid color.