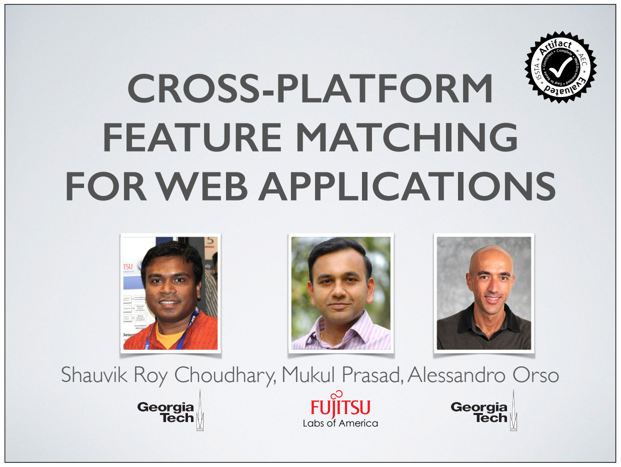
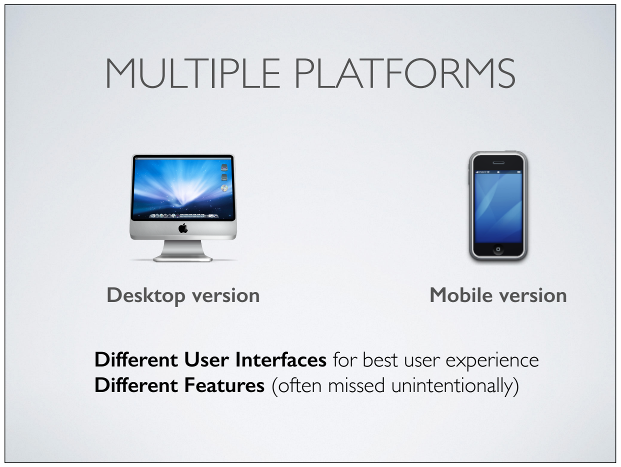


CROSS-PLATFORM FEATURE MATCHING FOR WEB APPLICATIONS

Shauvik Roy Choudhary, Mukul Prasad, Alessandro Orso
Georgia Tech · FUJITSU Labs of America · Georgia Tech

Thank you for your interest in my work.
This slide presents our work on web application feature matching across platforms.
This work was done at Georgia Tech in collaboration with Mukul Prasad from Fujitsu Labs and my advisor, Alex Orso.

Web applications have become an integral part of our life and we use them for our day-to-day personal and business activities on several different devices. But how do developers support different devices?



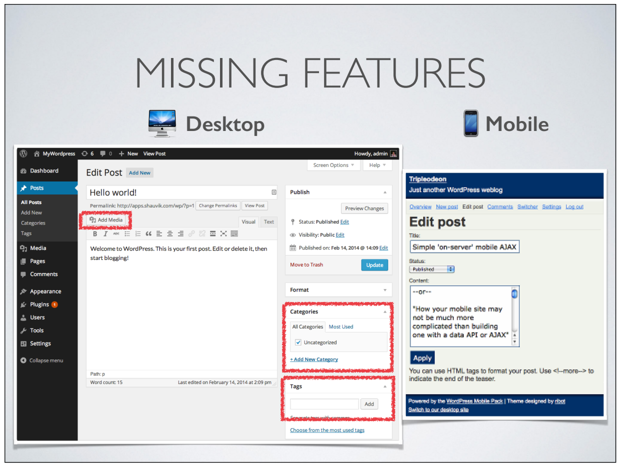
MULTIPLE PLATFORMS

Desktop version | Mobile version

Different User Interfaces for best user experience
Different Features (often missed unintentionally)

Developers typically build different versions of the web app. For example, a desktop version for large screen devices and mobile version for smaller screen devices.

- The UIs on different platforms are expected to be different and are tailored to offer the best user experience on that platform
- These versions differ not only in the UI but in the features supported, where a feature is a functionality offered by the web application.



MISSING FEATURES

Desktop

Mobile

Hello world!

Permalink: http://apps.shauvik.com/wp/?p=1

Welcome to WordPress. This is your first post. Edit or delete it, then start blogging!

Path: p

Word count: 15

Last edited on February 14, 2014 at 2:09 pm

Status: Published Edit

Visibility: Public Edit

Published on: Feb 14, 2014 @ 14:09 Edit

Tripleodeon

Just another WordPress weblog

Edit post

Title: Simple 'on-server' mobile AJAX

Status: Published

Content:

```
--or--

"How your mobile site may
not be much more
complicated than building
one with a data API or AJAX"
```

You can use HTML tags to format your post. Use <!--more--> to indicate the end of the teaser.

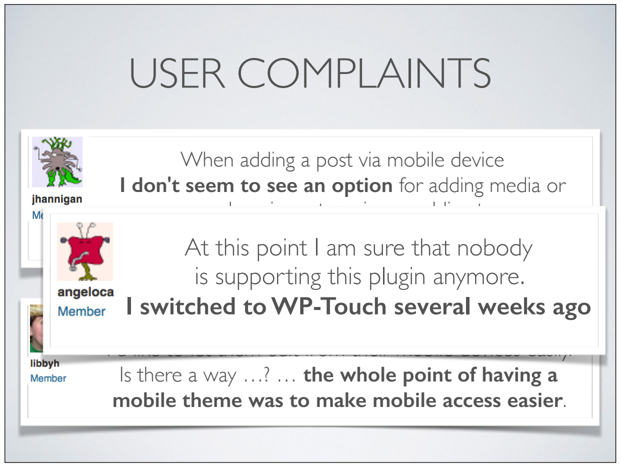
Powered by the WordPress Mobile Pack | Theme designed by ribot

Switch to our desktop site

Here are the desktop and mobile versions of the Wordpress web application for blogging. As shown,

desktop & mobile have significantly different UIs; and several features are missing on mobile. Example, while creating a blogpost, you can't add media or assign tags or categories to this post.

These missing features adversely affect the user.



USER COMPLAINTS

jhannigan

When adding a post via mobile device

I don't seem to see an option for adding media or

angeloca

Member

At this point I am sure that nobody is supporting this plugin anymore.

I switched to WP-Touch several weeks ago

libbyh

Member

Is there a way …? … **the whole point of having a mobile theme was to make mobile access easier**.

1) In fact this user complained about this same issue on the wordpress support forums.
2) The second user is a teacher and uses Wordpress to run her class blog. She is complaining that without specific features available on mobile, the whole point of having a mobile front-end is lost.
3) Finally, some people like this user, abandon the software and move on to other software.

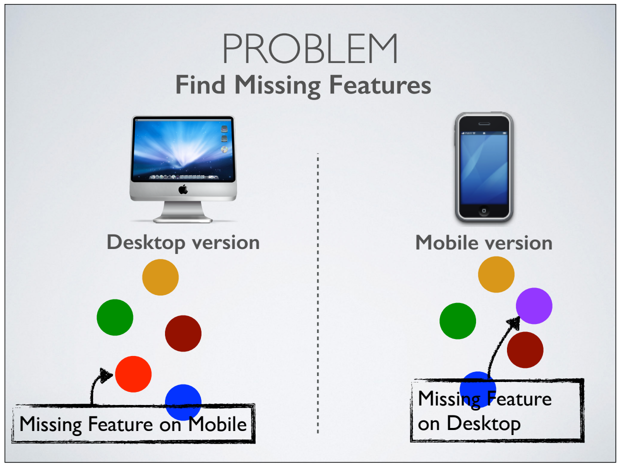
PROBLEM

Find Missing Features

Desktop version
Mobile version

Thus in this work, our goal is to find features present in one version of the web application but missing from another platform's version of it.

To achieve this task, we first find the different features of the web application on both platforms and..

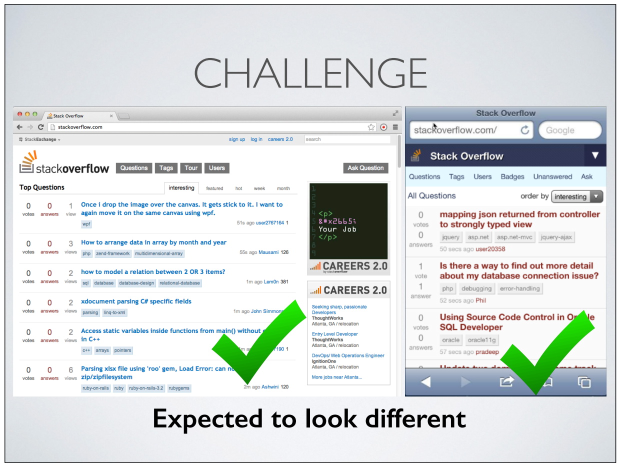


PROBLEM

Find Missing Features

.. establish a correspondence between matched features. Unmatched features are those, which are missing from the other platform.

This might seem easy at a high level; however there is a challenge while addressing this problem.



CHALLENGE

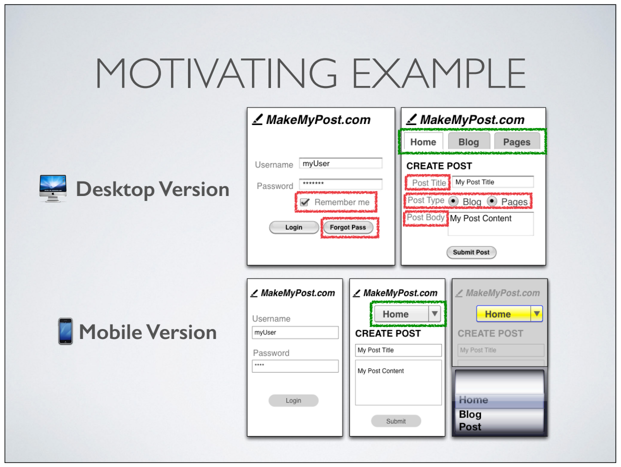
Expected to look different

stackoverflow
Top Questions
CAREERS 2.0
Stack Overflow
stackoverflow.com/
All Questions
mapping json returned from controller to strongly typed view
Is there a way to find out more detail about my database connection issue?
Using Source Code Control in Oracle SQL Developer

Web applications, like stackoverflow.com, shown here are expected to look different across platforms.

Hence,

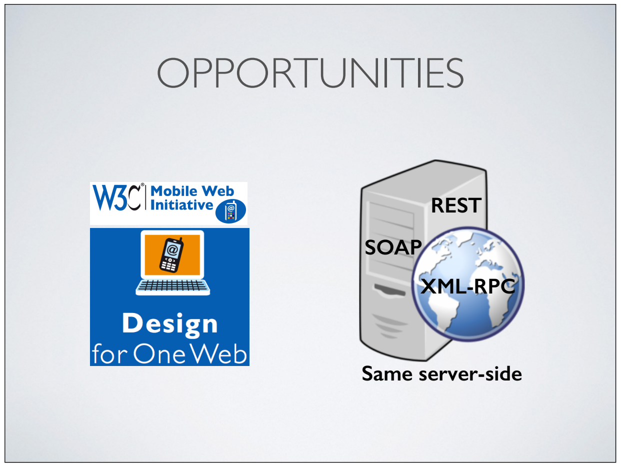
- UI comparison cannot be a basis for matching &
- any matching technique will need to work in spite of these expected differences and.



MOTIVATING EXAMPLE

Lets look at this fictitious website MakeMyPost.com, where the user can login and create a post. Although both versions support the post creation use-case, their user interfaces are significantly different. e.g., the red widgets are not present on mobile and different widgets highlighted in green are used for the same action.

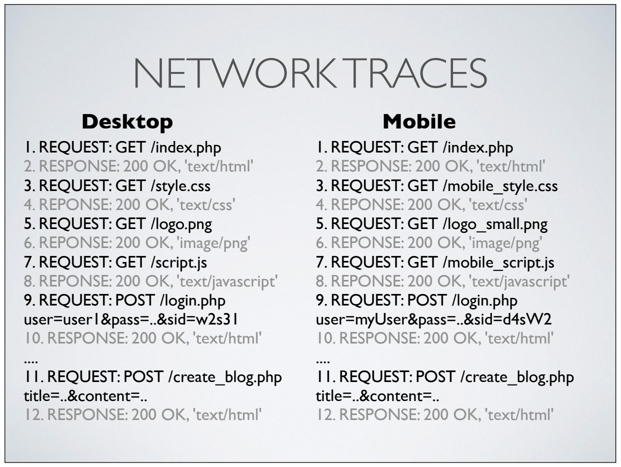
Thus, matching these features is very challenging given the different UIs. Now, lets look at some observations that provide us a basis for performing the feature matching…



OPPORTUNITIES

1) Web application versions need to offer similar features according to design guidelines (i.e., the W3C OneWeb principle which advocates that developers should make features available on each device albeit with different UIs.)
2) The next opportunity is the architectural similarity of web applications (i.e., both the platforms typically connect to a common backend.)

The combination of these provide us a basis (i.e. the client-server interface) which we can analyze and use to map the two versions.

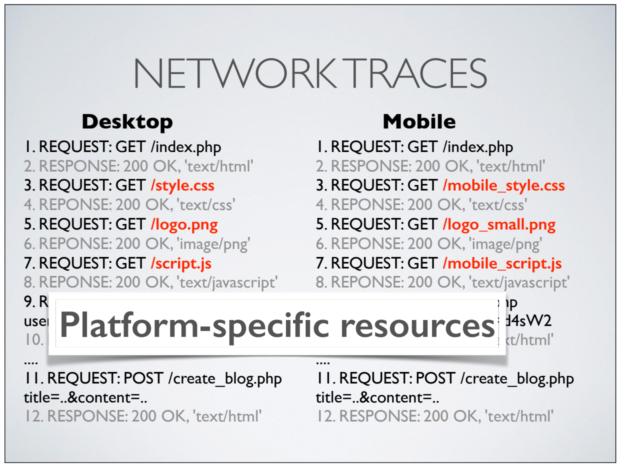


NETWORK TRACES

Desktop	Mobile
1. REQUEST: GET /index.php	1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'	2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET /style.css	3. REQUEST: GET /mobile_style.css
4. REPONSE: 200 OK, 'text/css'	4. REPONSE: 200 OK, 'text/css'
5. REQUEST: GET /logo.png	5. REQUEST: GET /logo_small.png
6. REPONSE: 200 OK, 'image/png'	6. REPONSE: 200 OK, 'image/png'
7. REQUEST: GET /script.js	7. REQUEST: GET /mobile_script.js
8. REPONSE: 200 OK, 'text/javascript'	8. REPONSE: 200 OK, 'text/javascript'
9. REQUEST: POST /login.php user=user1&pass=..&sid=w2s31	9. REQUEST: POST /login.php user=myUser&pass=..&sid=d4sW2
10. RESPONSE: 200 OK, 'text/html'	10. RESPONSE: 200 OK, 'text/html'
....	....
11. REQUEST: POST /create_blog.php title=..&content=..	11. REQUEST: POST /create_blog.php title=..&content=..
12. RESPONSE: 200 OK, 'text/html'	12. RESPONSE: 200 OK, 'text/html'

As shown here, the network level abstracts away most of the differences. In fact, our technique operates at this level for performing the feature matching.

But let me show you somethings that make our matching non-trivial.

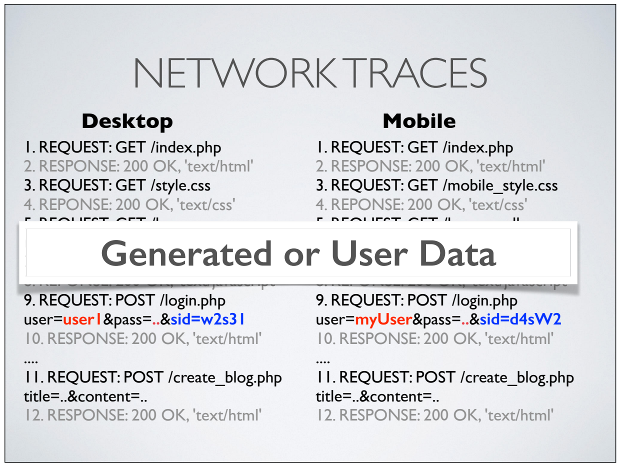


NETWORK TRACES

Desktop	Mobile
1. REQUEST: GET /index.php	1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'	2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET **/style.css**	3. REQUEST: GET **/mobile_style.css**
4. REPONSE: 200 OK, 'text/css'	4. REPONSE: 200 OK, 'text/css'
5. REQUEST: GET **/logo.png**	5. REQUEST: GET **/logo_small.png**
6. REPONSE: 200 OK, 'image/png'	6. REPONSE: 200 OK, 'image/png'
7. REQUEST: GET **/script.js**	7. REQUEST: GET **/mobile_script.js**
8. REPONSE: 200 OK, 'text/javascript'	8. REPONSE: 200 OK, 'text/javascript'
9. R use	ıp d4sW2
10.	t/html'
....	....
11. REQUEST: POST /create_blog.php title=..&content=..	11. REQUEST: POST /create_blog.php title=..&content=..
12. RESPONSE: 200 OK, 'text/html'	12. RESPONSE: 200 OK, 'text/html'

Platform-specific resources

Requests can contain platform specific resources requested from each client.



NETWORK TRACES

Desktop

1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET /style.css
4. REPONSE: 200 OK, 'text/css'

Generated or User Data

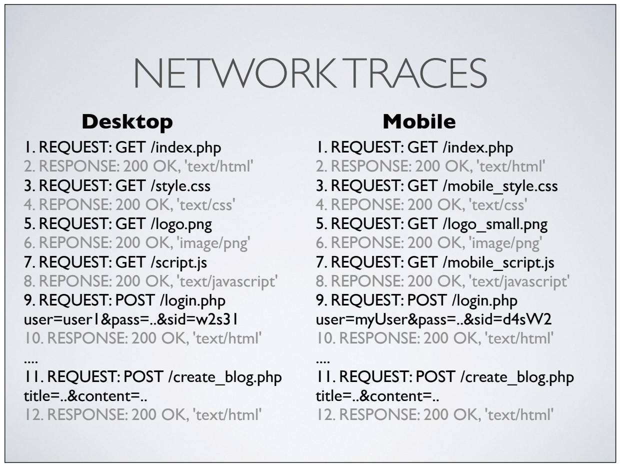
9. REQUEST: POST /login.php
user=user1&pass=..&sid=w2s31
10. RESPONSE: 200 OK, 'text/html'
....
11. REQUEST: POST /create_blog.php
title=..&content=..
12. RESPONSE: 200 OK, 'text/html'

Mobile

1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET /mobile_style.css
4. REPONSE: 200 OK, 'text/css'

9. REQUEST: POST /login.php
user=myUser&pass=..&sid=d4sW2
10. RESPONSE: 200 OK, 'text/html'
....
11. REQUEST: POST /create_blog.php
title=..&content=..
12. RESPONSE: 200 OK, 'text/html'

Requests can also contain system or user generated data such as IDs.



NETWORK TRACES

Desktop

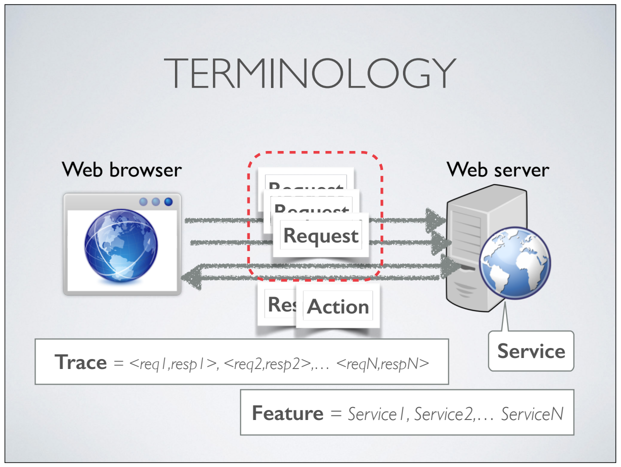
1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET /style.css
4. REPONSE: 200 OK, 'text/css'
5. REQUEST: GET /logo.png
6. REPONSE: 200 OK, 'image/png'
7. REQUEST: GET /script.js
8. REPONSE: 200 OK, 'text/javascript'
9. REQUEST: POST /login.php
user=user1&pass=..&sid=w2s31
10. RESPONSE: 200 OK, 'text/html'
....
11. REQUEST: POST /create_blog.php
title=..&content=..
12. RESPONSE: 200 OK, 'text/html'

Mobile

1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'
3. REQUEST: GET /mobile_style.css
4. REPONSE: 200 OK, 'text/css'
5. REQUEST: GET /logo_small.png
6. REPONSE: 200 OK, 'image/png'
7. REQUEST: GET /mobile_script.js
8. REPONSE: 200 OK, 'text/javascript'
9. REQUEST: POST /login.php
user=myUser&pass=..&sid=d4sW2
10. RESPONSE: 200 OK, 'text/html'
....
11. REQUEST: POST /create_blog.php
title=..&content=..
12. RESPONSE: 200 OK, 'text/html'

Our technique is designed to ignore such differences and match features across the platforms at the network level.

Before presenting our technique, let me walk you through some terminology used in our approach.

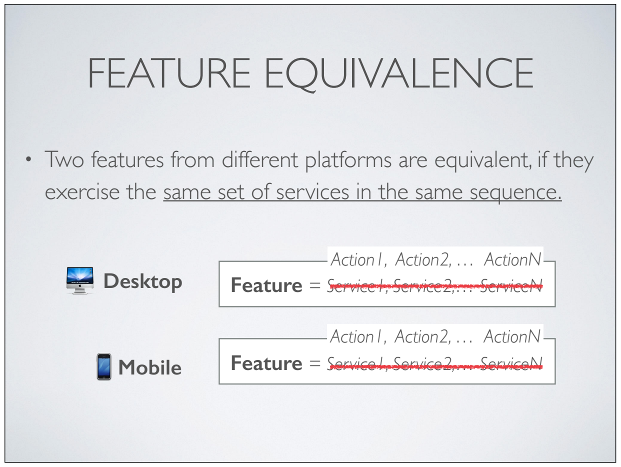


TERMINOLOGY

Web browser
Web server
Request
Request
Request
Res
Action
Service
Trace = <req1,resp1>, <req2,resp2>,… <reqN,respN>
Feature = Service1, Service2,… ServiceN

We will consider a typical client-server scenario as in web applications.

- A service is an atomic functionality offered by the web application. e.g., user login is an example of a service
- A feature represents a set of services invoked in a certain sequence. e.g., user login followed by deleting a post is a feature
- To access a service, the browser sends a request, which is served with a response.
- A network trace is a set of such request-response pairs in a particular order.
- Requests from different traces, devices and other contexts can access the same service. To combine these traces, we introduce an action, which is an abstraction of requests where all non-essential information is removed… and corresponds to a service being invoked.



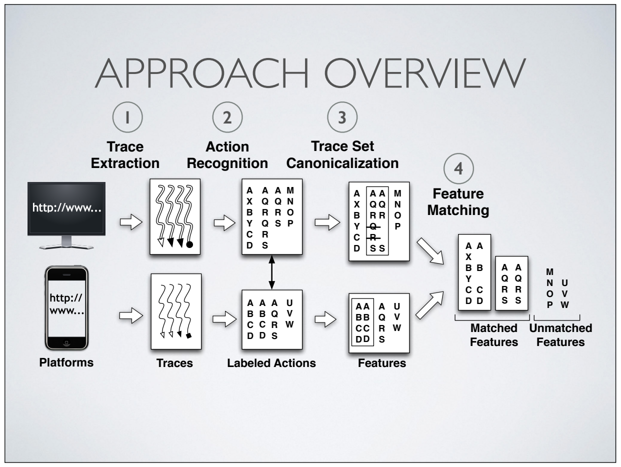
FEATURE EQUIVALENCE

- Two features from different platforms are equivalent, if they exercise the same set of services in the same sequence.

Desktop Feature = $\overset{\text{Action1, Action2, … ActionN}}{\text{~~Service1, Service2, … ServiceN~~}}$

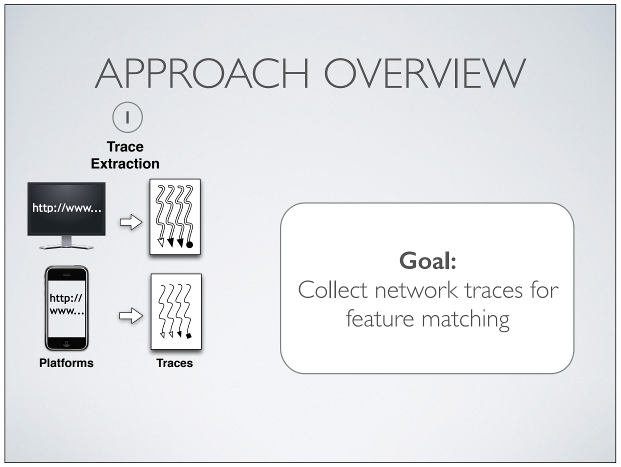
Mobile Feature = $\overset{\text{Action1, Action2, … ActionN}}{\text{~~Service1, Service2, … ServiceN~~}}$

The goal of the technique is to establish feature equivalence, where equivalent features exercise the same set of services in the same sequence.
However, since we are operating on the network level and the server side is a black box, our technique will use actions as a proxy for the services under consideration.
Now that I gave you the underlying formalism, let me walk you through our technique.



APPROACH OVERVIEW

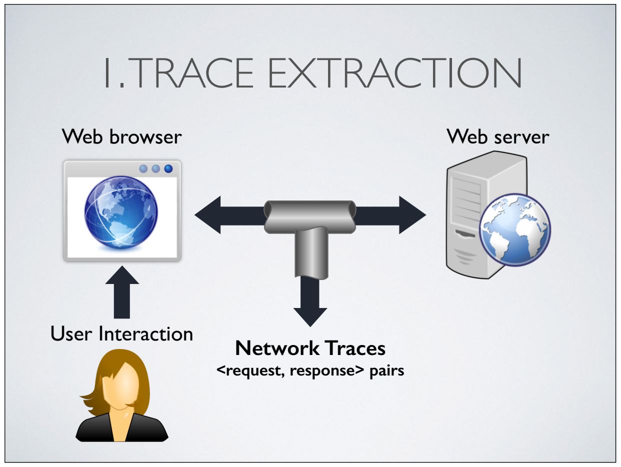
This is the high level overview of our technique, which will be explained in detail in upcoming slides.
Given an application running on desktop and mobile, the technique extracts network traces of the application.
Then, we process traces to recognize the different actions present in the traces
Then we combine similar traces into elemental traces — each of which correspond to a feature
The final feature matching step gives the matched and unmatched features; the latter being missing features on one of the platforms.



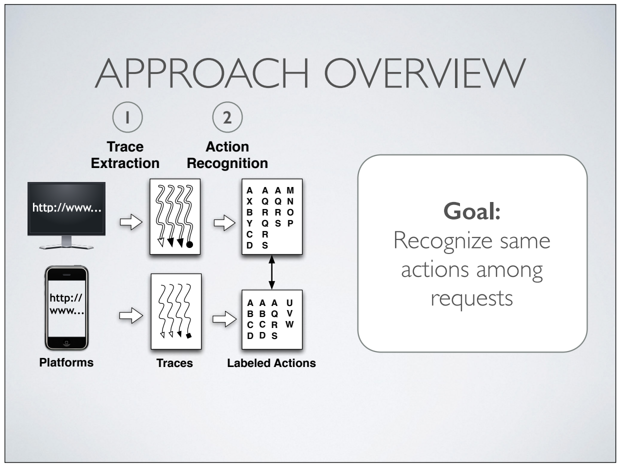
APPROACH OVERVIEW

Goal:
Collect network traces for feature matching

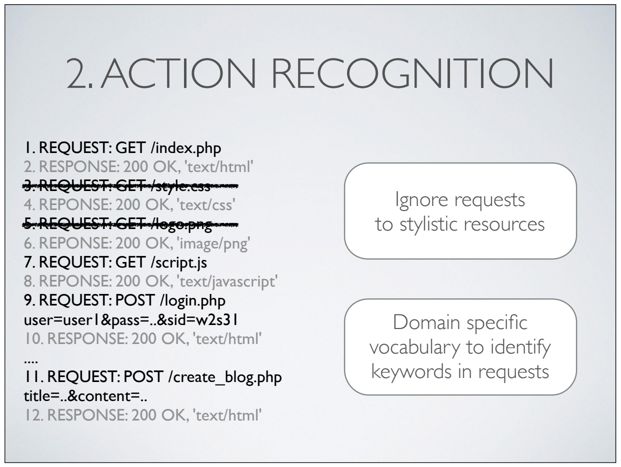
Let's look into detail by starting at trace extraction, where the goal is to collect network traces that are used for feature matching.


1. TRACE EXTRACTION

This step is mechanical, where as the user is accessing the web application, the network communication emanating from the browser is captured as network traces.


APPROACH OVERVIEW

In the next step, we process these traces to recognize similar actions among requests in different traces

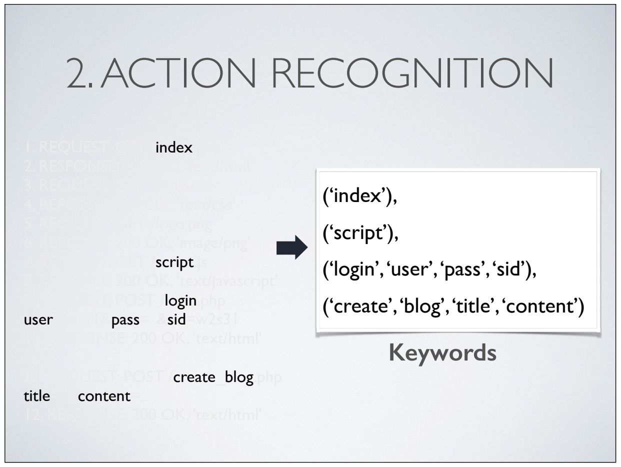

2. ACTION RECOGNITION

1. REQUEST: GET /index.php
2. RESPONSE: 200 OK, 'text/html'
~~3. REQUEST: GET /style.css~~
4. REPONSE: 200 OK, 'text/css'
~~5. REQUEST: GET /logo.png~~
6. REPONSE: 200 OK, 'image/png'
7. REQUEST: GET /script.js
8. REPONSE: 200 OK, 'text/javascript'
9. REQUEST: POST /login.php
user=user1&pass=..&sid=w2s31
10. RESPONSE: 200 OK, 'text/html'
....
11. REQUEST: POST /create_blog.php
title=..&content=..
12. RESPONSE: 200 OK, 'text/html'

Ignore requests to stylistic resources

Domain specific vocabulary to identify keywords in requests

Specifically, given the traces, we first ignore stylistic resources such as images/stylesheets. Then we use a domain specific vocabulary to extract keywords from the requests.

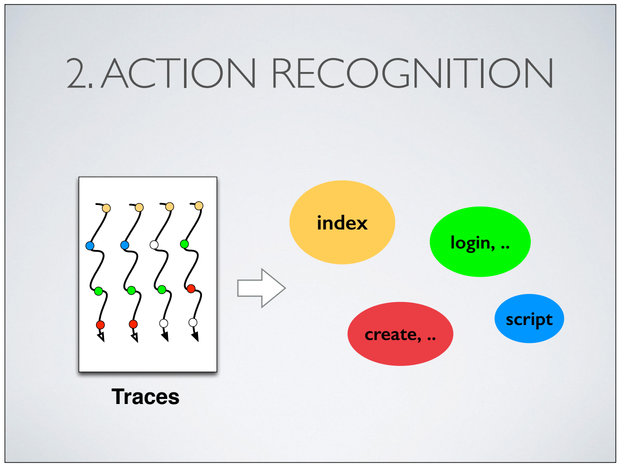


2. ACTION RECOGNITION

('index'),
('script'),
('login', 'user', 'pass', 'sid'),
('create', 'blog', 'title', 'content')

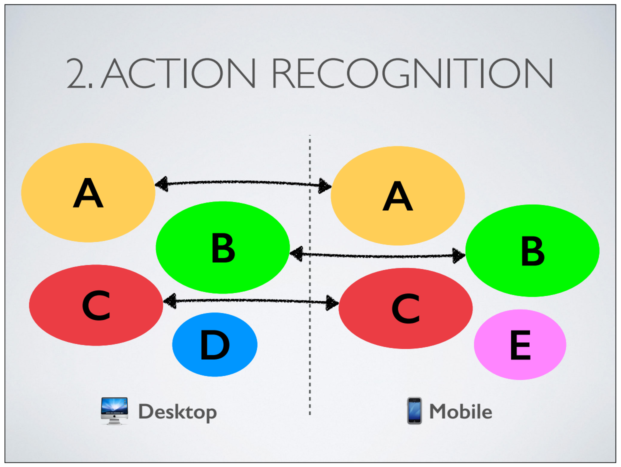
Keywords

For our example, we get the following keyword sets.
These keywords gives us a basis to abstract requests across traces to identify same actions.



2. ACTION RECOGNITION

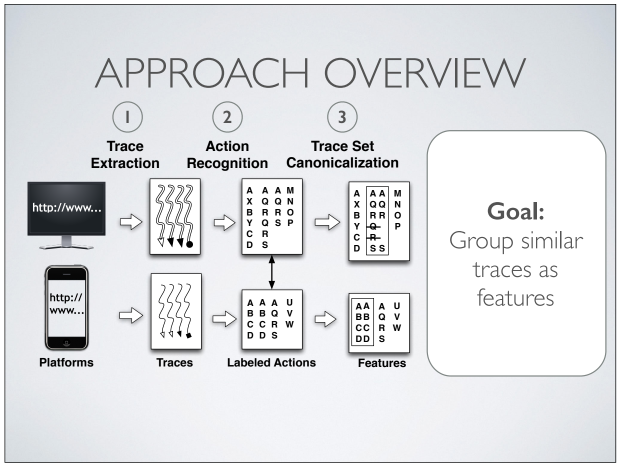
In particular, through the similarity of keywords, we identify requests potentially corresponding to the same action, and **group them into clusters**. For instance, all the yellow requests correspond to the index cluster.



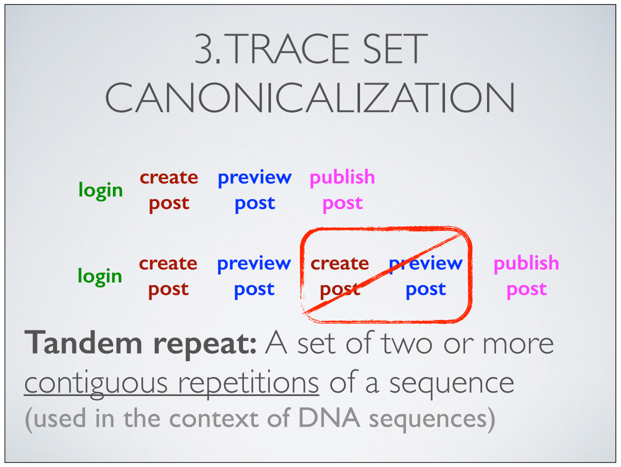
2. ACTION RECOGNITION

Then, we look at clusters across platforms and establish a correspondence by assigning each matching clusters the same label across platforms.
For the clusters in our example, index is assigned the similar label A, and so is B & C.
D and E are platform specific actions and are not present on both sides.

Now that we have same actions across different platforms, our next goal is to identify features.



For identifying features, we identify and merge slight variations of traces that represent the same feature.



3. TRACE SET CANONICALIZATION

login create post preview post publish post

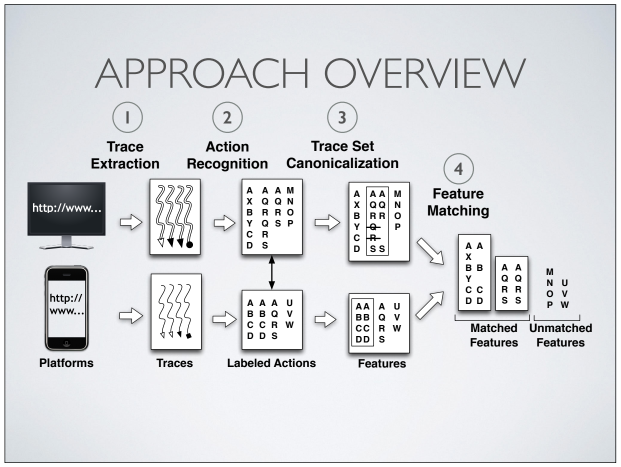
login create post preview post ~~create post preview post~~ publish post

Tandem repeat: A set of two or more contiguous repetitions of a sequence (used in the context of DNA sequences)

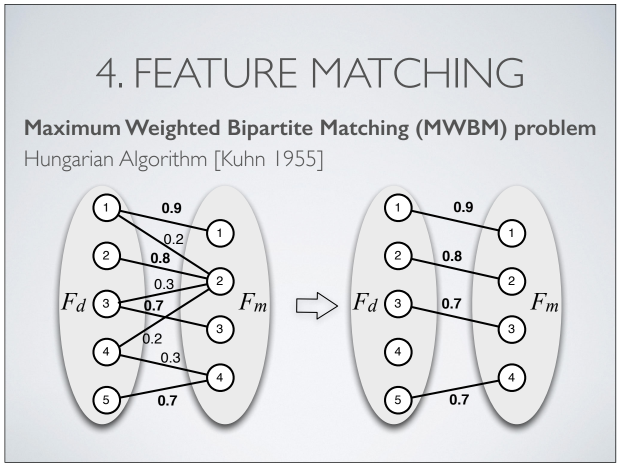
An example of this is is highlighted here. In the first trace, the user logs in, creates a post, previews it and finally publishes the post. In the second trace, the user previews the post two times by navigating back to the create post action. Both of these represent the same post creation feature.

Thus, in this step, we canonicalize such traces into their most elemental form.

We use the concept of tandem repeat finder, which has been used heavily in the context of DNA sequences. This allows us to combine these two traces into the same feature..



The final step in our approach matches features across the different platforms and reports unmatched features as missing. Let me explain how we do this in more detail.



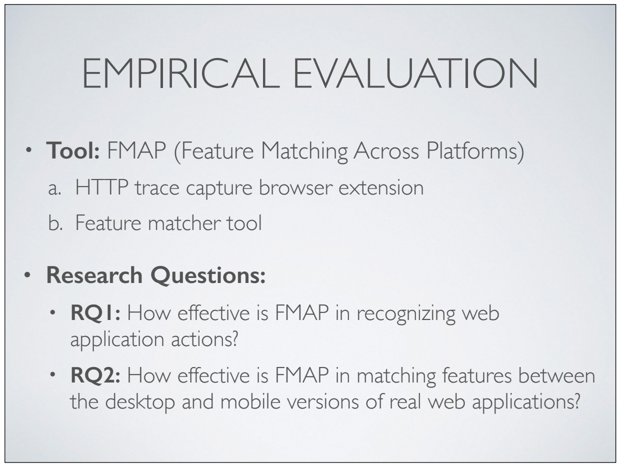
4. FEATURE MATCHING

Maximum Weighted Bipartite Matching (MWBM) problem

Hungarian Algorithm [Kuhn 1955]

We formulate feature matching as a maximum weighted bipartite graph matching problem, and leverage the Hungarian algorithm, which is an **efficient and polynomial time algorithm**. The following is an example of our bipartite graph, where vertices are features on each platform - desktop and mobile.. and edges are a possible similarity between these features, with the edge weight as the similarity metric. We used the HCS as our similarity metric .. details can be found in the paper.

Now, the MWBM problem is to find a 1-1 bipartite matching such that the additive weight of chosen edges is maximized. For this particular graph, we show the solution on the right side, where 1-1, 2-2, 3-3, 5-4.. and 4 in the desktop. So, this how our algorithm works.

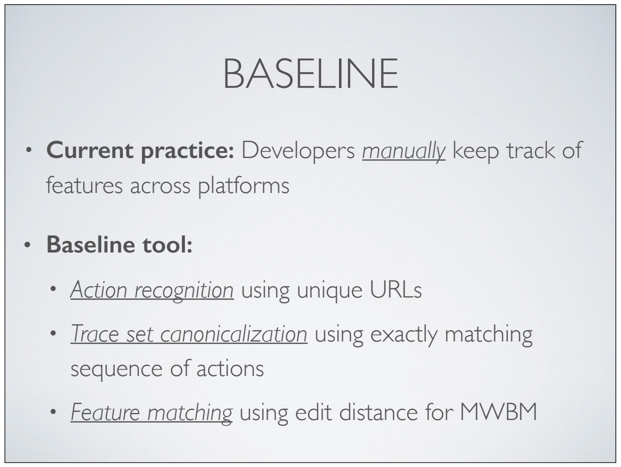


EMPIRICAL EVALUATION

- **Tool:** FMAP (Feature Matching Across Platforms)
  a. HTTP trace capture browser extension
  b. Feature matcher tool
- **Research Questions:**
  - **RQ1:** How effective is FMAP in recognizing web application actions?
  - **RQ2:** How effective is FMAP in matching features between the desktop and mobile versions of real web applications?

To evaluate our technique, we implemented it in a tool called FMAP, which stands for Feature Matching across Platforms. Our tool consists of two components — 1) Browser extension for the Chromium web browser to capture network traces 2) The feature matcher tool implements the rest of the steps and provides a matching of features identified in the traces.

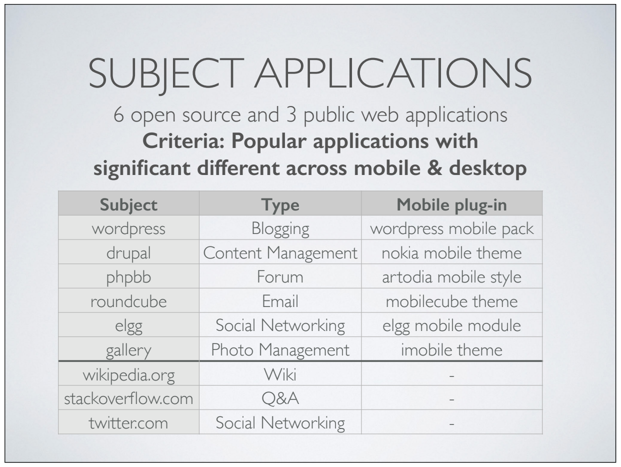
We used our tool to answer the following research questions:
1) How effective is FMAP in the action recognition step..
2) What is FMAPs effective in the overall feature matching.
In the interest of time, I will explain RQ2. Details of RQ1 are available in the paper.



BASELINE

- **Current practice:** Developers *manually* keep track of features across platforms
- **Baseline tool:**
  - *Action recognition* using unique URLs
  - *Trace set canonicalization* using exactly matching sequence of actions
  - *Feature matching* using edit distance for MWBM

For our experiments, we wanted a baseline. However, current practice among developers is to manually keep track of features across platforms. Thus, we implemented a baseline by changing key steps of our technique with simpler ones. for example, in the trace set canonicalizing step, we use exact action sequence matching and did not perform tandem repeat elimination.



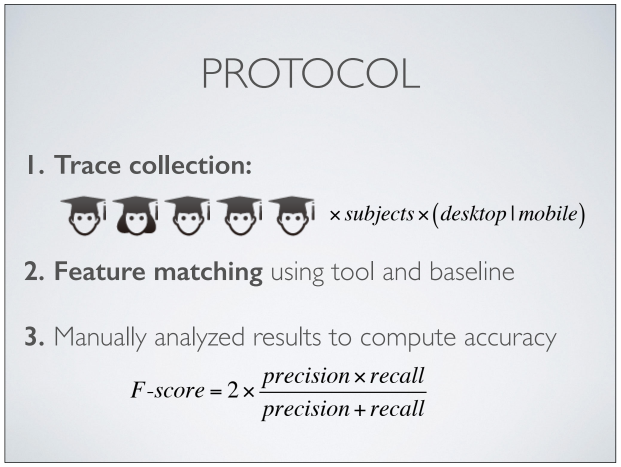
SUBJECT APPLICATIONS

6 open source and 3 public web applications

Criteria: Popular applications with significant different across mobile & desktop

Subject	Type	Mobile plug-in
wordpress	Blogging	wordpress mobile pack
drupal	Content Management	nokia mobile theme
phpbb	Forum	artodia mobile style
roundcube	Email	mobilecube theme
elgg	Social Networking	elgg mobile module
gallery	Photo Management	imobile theme
wikipedia.org	Wiki	-
stackoverflow.com	Q&A	-
twitter.com	Social Networking	-

For our study, we chose 6 open source and 3 public web applications, as shown in this table. We picked these from a list of top web applications, such that their user interface was significantly different between the platforms.



PROTOCOL

1. Trace collection:

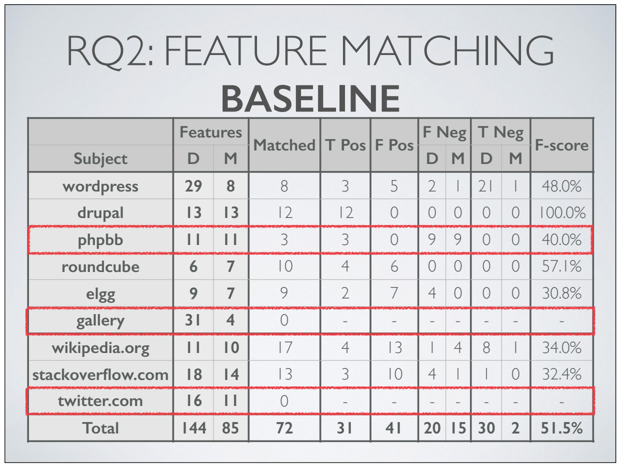
$\times subjects \times (desktop \mid mobile)$

2. Feature matching using tool and baseline

3. Manually analyzed results to compute accuracy

$$F\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}$$

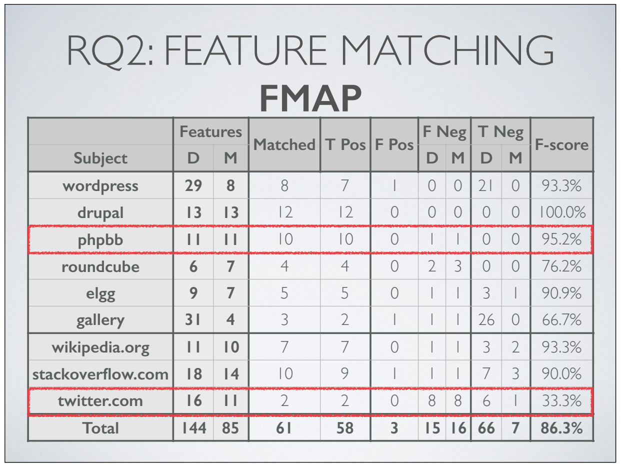
- For conducting our experiments, we recruited 5 graduate students for trace collection and gave them one or more subjects and asked them to study the user interface of the web application, define as many use cases as they could and then exercise these use cases on either the desktop or mobile version of the subject.
- Next, we performed feature matching using our tool and the baseline, and, manually analyzed the results to compute accuracy. For computing accuracy, we calculated both precision and recall, and used it to compute the F-score metric, which is reported in our results.



RQ2: FEATURE MATCHING BASELINE

	Features		Matched	T Pos	F Pos	F Neg		T Neg		F-score
Subject	D	M				D	M	D	M	
wordpress	29	8	8	3	5	2	1	21	1	48.0%
drupal	13	13	12	12	0	0	0	0	0	100.0%
phpbb	11	11	3	3	0	9	9	0	0	40.0%
roundcube	6	7	10	4	6	0	0	0	0	57.1%
elgg	9	7	9	2	7	4	0	0	0	30.8%
gallery	31	4	0	-	-	-	-	-	-	-
wikipedia.org	11	10	17	4	13	1	4	8	1	34.0%
stackoverflow.com	18	14	13	3	10	4	1	1	0	32.4%
twitter.com	16	11	0	-	-	-	-	-	-	-
Total	144	85	72	31	41	20	15	30	2	51.5%

This table shows the results of the Baseline tool in feature matching. The table shows for each subject on desktop and mobile platforms, the reported feature matchings, true positives, false positives, false negatives, true negatives and the overall F-score of the matching.
For instance, for PHPBB, out of 11 features, it could match 3 features, which were all true positives but the rest 9 were false negatives, and the F-score was 40%
Note that for two subjects, gallery and twitter, it did not report any results.. and the overall F-score of the baseline tool was 51.5%

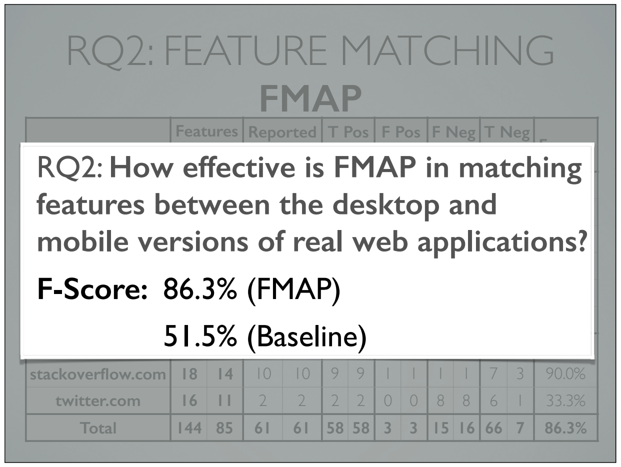


RQ2: FEATURE MATCHING

FMAP

Subject	Features		Matched	T Pos	F Pos	F Neg		T Neg		F-score
	D	M				D	M	D	M	
wordpress	**29**	**8**	8	7	1	0	0	21	0	93.3%
drupal	**13**	**13**	12	12	0	0	0	0	0	100.0%
phpbb	**11**	**11**	10	10	0	1	1	0	0	95.2%
roundcube	**6**	**7**	4	4	0	2	3	0	0	76.2%
elgg	**9**	**7**	5	5	0	1	1	3	1	90.9%
gallery	**31**	**4**	3	2	1	1	1	26	0	66.7%
wikipedia.org	**11**	**10**	7	7	0	1	1	3	2	93.3%
stackoverflow.com	**18**	**14**	10	9	1	1	1	7	3	90.0%
twitter.com	**16**	**11**	2	2	0	8	8	6	1	33.3%
Total	**144**	**85**	**61**	**58**	**3**	**15**	**16**	**66**	**7**	**86.3%**

Now we see the results of the FMAP tool.
As shown, for the same subject PHPBB, FMAP could match 10 features instead of 3 and the F-score was 95.2%.
Let me highlight the most challenging subject for FMAP — twitter. From the traces, we found that the desktop and mobile versions were interacting with different servers, and on contacting a twitter developer, we found that the mobile version was a legacy application. We think that this is an exception, and that server-side is typically the same, because it favors code re-use and eases maintenance.



RQ2: FEATURE MATCHING

FMAP

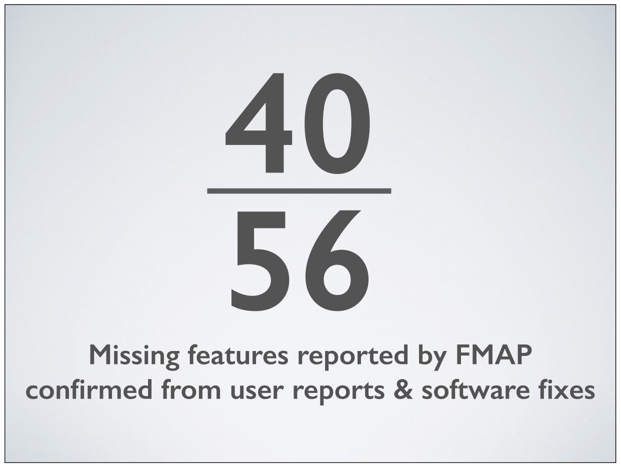
	Features		Reported		T Pos		F Pos		F Neg		T Neg		F
stackoverflow.com	18	14	10	10	9	9	1	1	1	1	7	3	90.0%
twitter.com	16	11	2	2	2	2	0	0	8	8	6	1	33.3%
Total	144	85	61	61	58	58	3	3	15	16	66	7	86.3%

RQ2: **How effective is FMAP in matching features between the desktop and mobile versions of real web applications?**

F-Score: 86.3% (FMAP)

51.5% (Baseline)

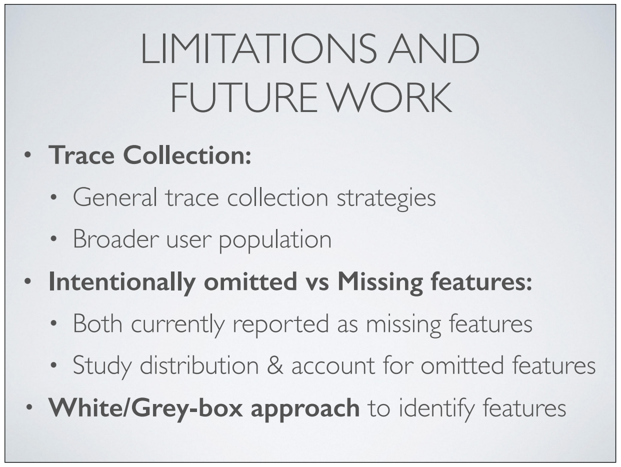
In summary, the answer to RQ2 is YES, FMAP is effective in matching features, and thus finding missing features. Its F-score is 86.3% as compared to the baseline's 51.5%



40
56

Missing features reported by FMAP confirmed from user reports & software fixes

Moreover, 40 out of 56 features reported missing by FMAP were confirmed from user reports and fixes to the web application later by the developers themselves.
This increases our confidence in FMAP's ability to find missing features.

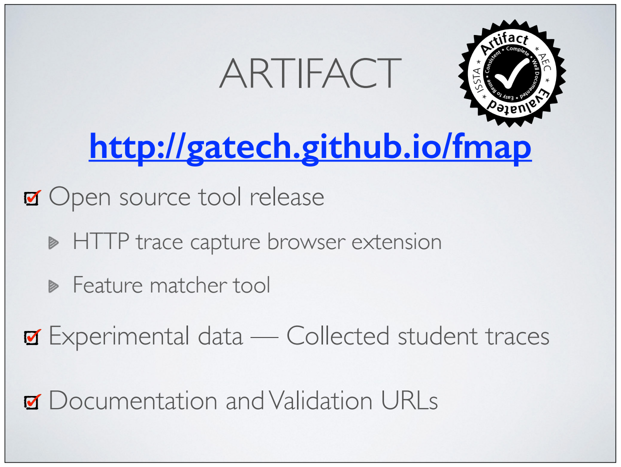


LIMITATIONS AND FUTURE WORK

- **Trace Collection:**
  - General trace collection strategies
  - Broader user population
- **Intentionally omitted vs Missing features:**
  - Both currently reported as missing features
  - Study distribution & account for omitted features
- **White/Grey-box approach** to identify features

While our approach is a good first attempt at this problem, it has certain limitations which needs to be addressed by future work.

1) Our traces were collected by students. We could improve coverage of features in our traces by collecting traces from a more realistic deployment and possibly from a broader user population such as those obtained via crowd sourcing.
2) Secondly, features are sometimes omitted intentionally on certain platforms. Any automated technique such as ours cannot distinguish between missing vs omitted features. Future work should study the distribution of omitted vs missing features and account for that in the technique.
3) Finally, our approach is black box in nature and the first attempt towards solving this



ARTIFACT

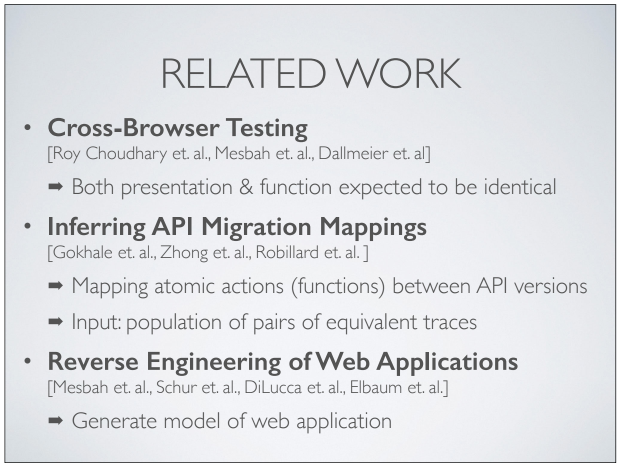
http://gatech.github.io/fmap

☑ Open source tool release
- HTTP trace capture browser extension
- Feature matcher tool

☑ Experimental data — Collected student traces

☑ Documentation and Validation URLs

Our artifact has been approved by the AEC committee and is available on github.
It includes, our tool, the experimental data of student traces, documentation and URLs used to validate the missing features.

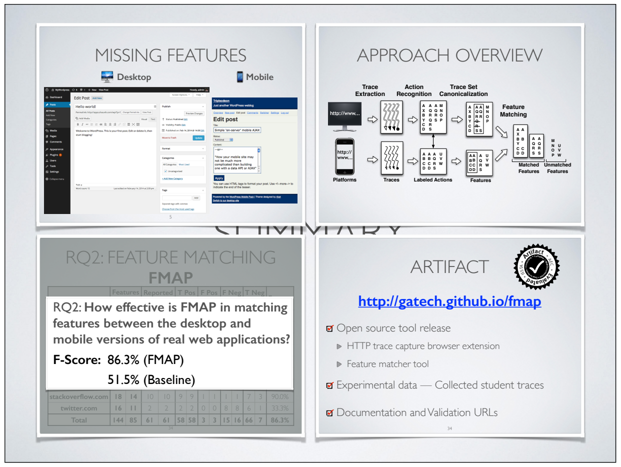


RELATED WORK

- **Cross-Browser Testing**
  [Roy Choudhary et. al., Mesbah et. al., Dallmeier et. al]
  ➡ Both presentation & function expected to be identical
- **Inferring API Migration Mappings**
  [Gokhale et. al., Zhong et. al., Robillard et. al.]
  ➡ Mapping atomic actions (functions) between API versions
  ➡ Input: population of pairs of equivalent traces
- **Reverse Engineering of Web Applications**
  [Mesbah et. al., Schur et. al., DiLucca et. al., Elbaum et. al.]
  ➡ Generate model of web application

There are 3 sets of related work.

1) There is work on Cross-browser testing, where both presentation & function are expected to be identical and any differences are reported as issues. However, in our case these are expected to be different.
2) There is also a line of work is to infer API migration mappings, which maps atomic actions (i.e., functions) between different API versions. These techniques take as an input a population of pairs of equivalent traces. However, our technique needs to find this equivalence.
3) The final set of related work is to reverse engineer web applications to generate a model of a web application. This is an orthogonal problem and we believe that feature matching


MISSING FEATURES

Desktop

Mobile

APPROACH OVERVIEW

RQ2: FEATURE MATCHING

FMAP

RQ2: **How effective is FMAP in matching features between the desktop and mobile versions of real web applications?**

F-Score: 86.3% (FMAP)

51.5% (Baseline)

ARTIFACT

http://gatech.github.io/fmap

- Open source tool release
  - HTTP trace capture browser extension
  - Feature matcher tool
- Experimental data — Collected student traces
- Documentation and Validation URLs

In summary, I motivated the problem of finding missing features. I presented our technique for matching features with its evaluation and open source release of our artifact.

Thank you for your time! If you have any questions or comments please reach out to me at shauvik [at] gmail.com