Privacy-Preserving Offloading of Mobile App to the Public Cloud

Yue Duan, Mu Zhang, Heng Yin and Yuzhe Tang
Department of EECS Syracuse University

Overview

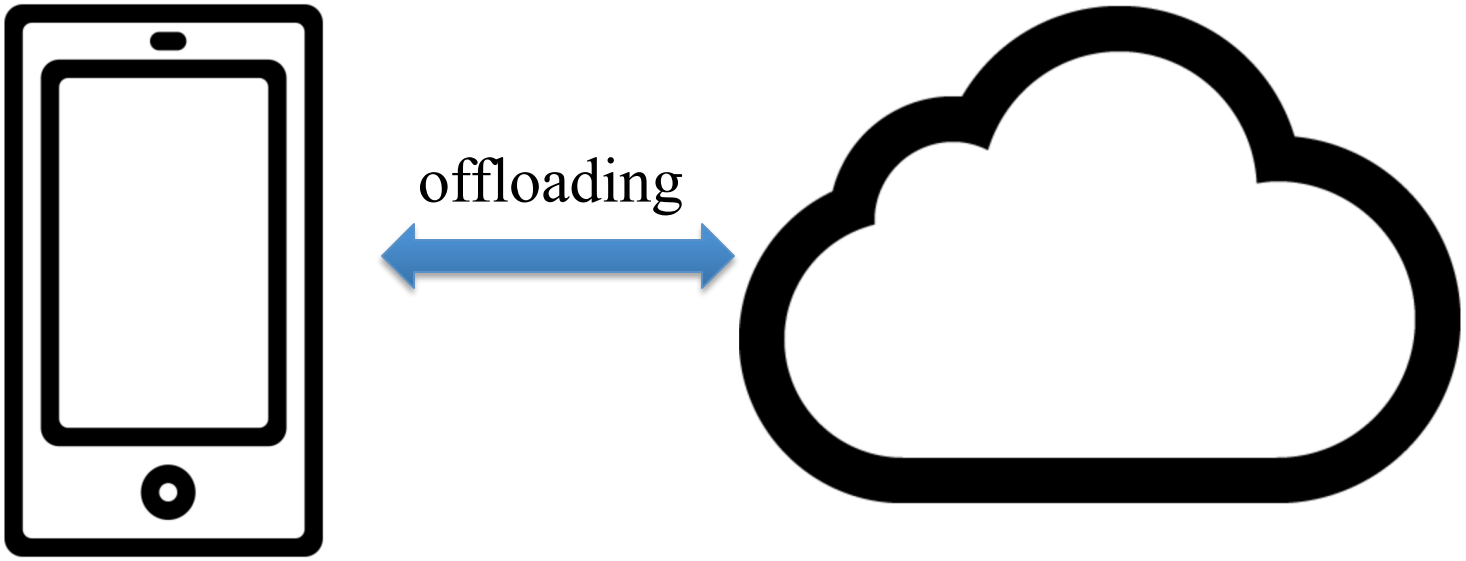
- **Introduction to offloading**
- Motivating example
- System design & goals
- Implementation details
- Preliminary evaluation
- Discussion

Introduction to offloading

- More and more complicated mobile apps are developed and installed.
- However, mobile devices have only limited hardware resources.

Introduction to offloading

- In order to preserve limited hardware resources (CPU, battery etc.), researchers develop techniques to execute part of the mobile apps to cloud server.



Introduction to offloading

- Most existing research work focuses on identifying computation-intensive portion of the program to offload.
  - MAUI[MobiSys'10]
  - CloneCloud[EuroSys'11]
  - COMET[OSDI'12]
- Mobile app offloading needs to send **code** and **data** to the public cloud to enable the program execution.
- There is little work that attempts to address the privacy issue during offloading.

Overview

- Introduction to offloading
- **Motivating example**
- System design & goals
- Implementation details
- Preliminary evaluation
- Discussion

Motivating example

Example: a simplified photo editor

```
public void OnTouchEvent(MotionEvent e) {
    if(e is the correct touch event){
      //First load picture to bitmap
      //then embed location info after editing
      Bitmap bm = LoadPicture(path);
      EditPicture(bm);
      ...
    }
}
Bitmap LoadPicture(String path) {
    ...
    Bitmap bm = BitmapFactory.decodeFile(file);
    return bm;
}
void EditPicture(Bitmap bm) {
    //heavy computational picture editing
    bm.getPixels(pix, 0, mPhotoWidth, 0, 0,
     mPhotoWidth, mPhotoHeight);
    for(every pixel) {
      ...
    }
    //embed location information into picture
    Location loc = getLastKnownLocation(Provider);
    double longitude = loc.getLongitude();
    double latitude = loc.getLatitude();
    EmbedLocation(bm, longitude, latitude);
}
```

Non-offloadable code

Computation-intensive code

Motivating example

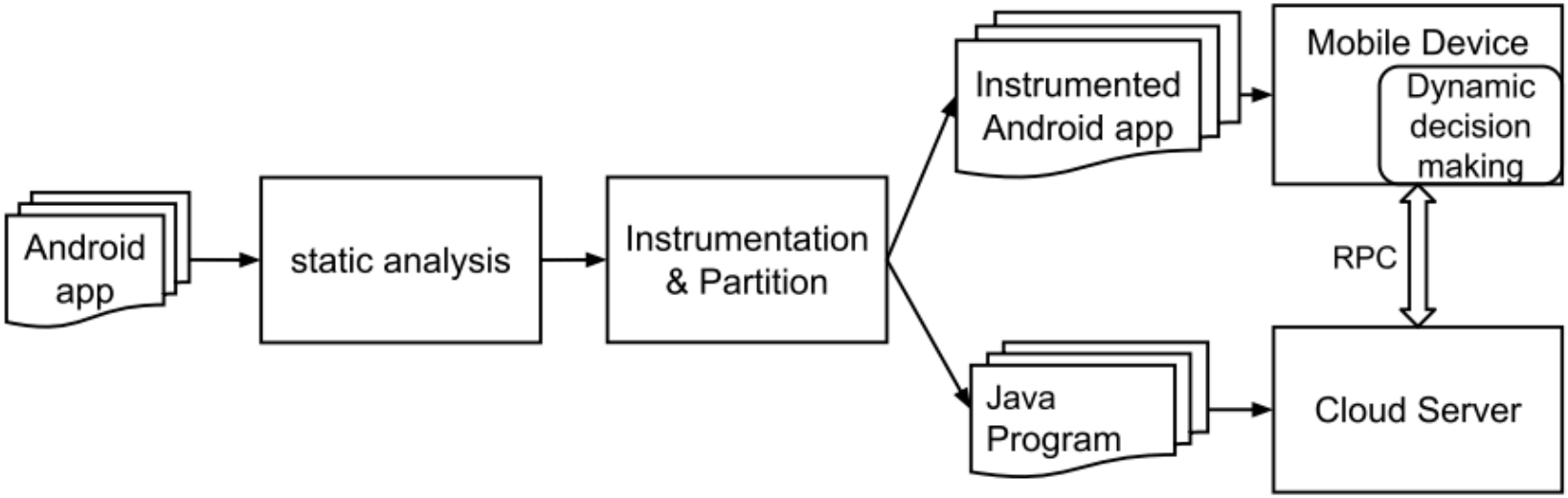
From the motivating example, we learn:

- Some data-flow analysis is needed to identify non-offloadable code
- Fine-grained app offloading is needed (method-level offloading is not enough)
- Computation-intensive code maybe interleaved with non-offloadable code (differentiate offloadable code from non-offloadable code)

Overview

- Introduction to offloading
- Motivating example
- **System design & goals**
- Implementation details
- Preliminary evaluation
- Discussion

System design & goals



The whole system is consisted of four major components:

1. **Static Analysis**: Identify and extract offloadable code regions
2. **Instrumentation & Partition**: Invoke dynamic decision-making component and perform offloading
3. **Cloud-side Deployment**: Use offloadable code regions to create a Java program on server side
4. **Dynamic decision Making**: Collect system information at runtime and make dynamic offloading decisions

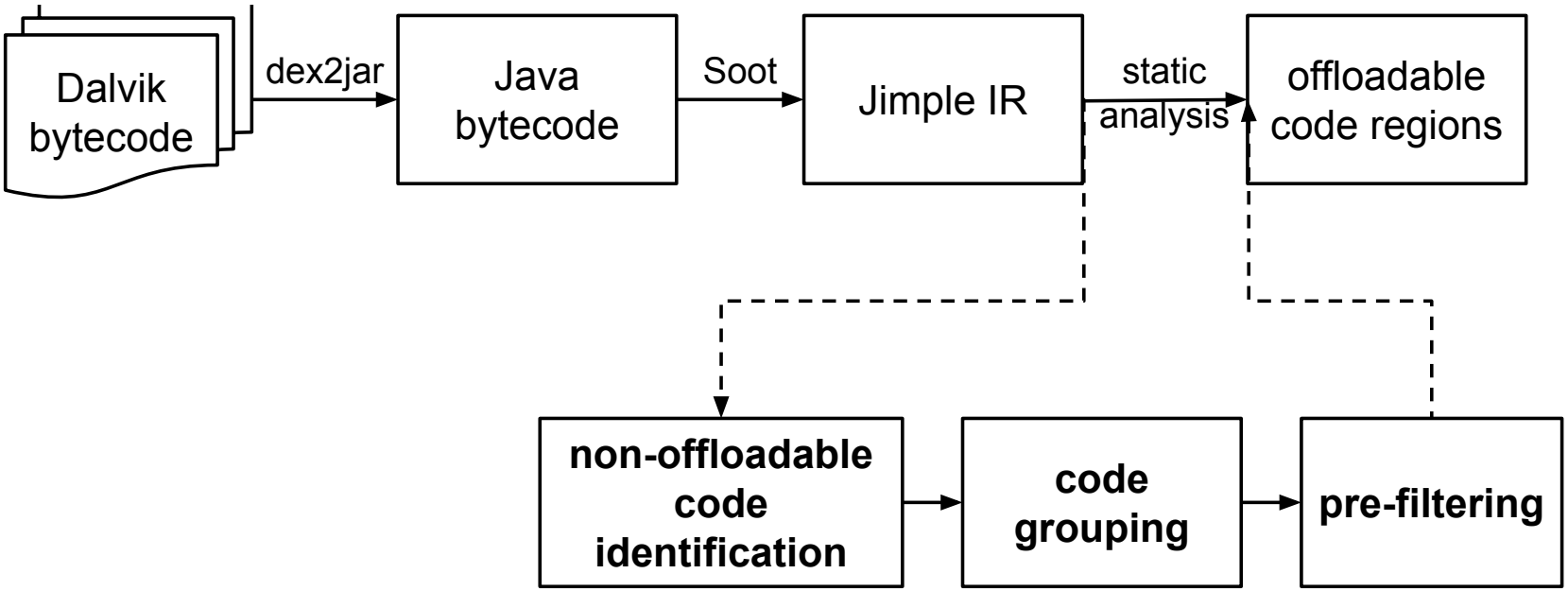
System design & goals

- From our system design, we aim at achieving the following design goals:
  - **Privacy preservation**: keep user private data within mobile devices during offloading
  - **Automatic offloading**: do not involve any human effort such as user annotation
  - **High performance**: self-explanatory but we face new challenges due to fine-grained app offloading

Overview

- Introduction to offloading
- Motivating example
- System design & goals
- **Implementation details**
- Preliminary evaluation
- Discussion

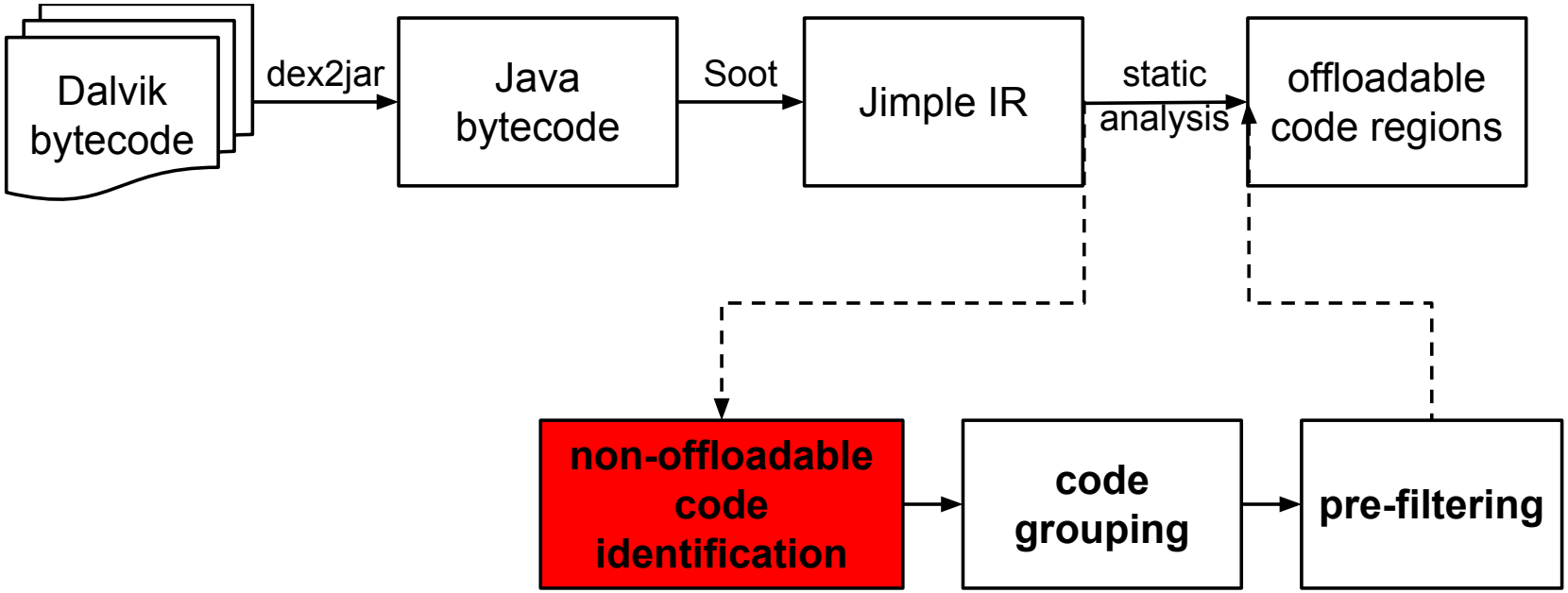
Implementation – Static Analysis



Static analysis has three major steps:

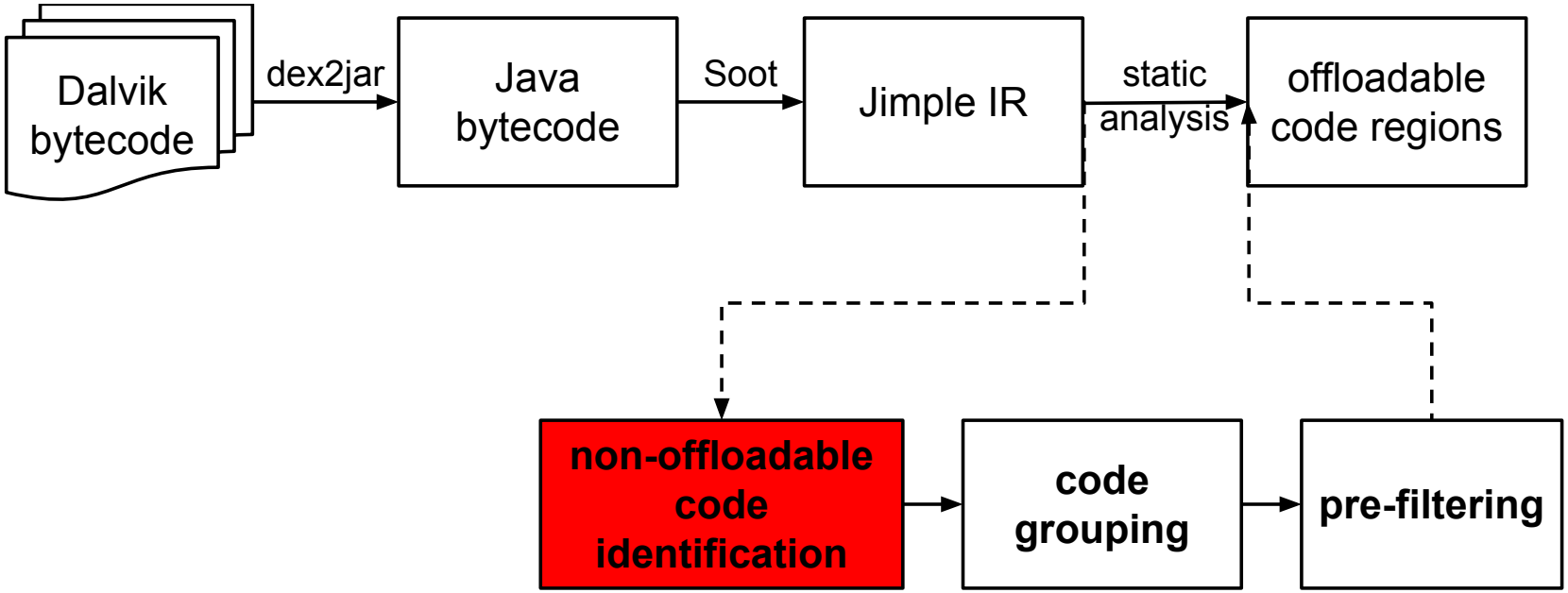
1. **non-offloadable code identification**
2. **offloadable code grouping**
3. **pre-filtering**

Implementation – Static Analysis



Definition of privacy: Our system allows user to configure what is considered as private information. Currently, we consider location, system setting (locale) and device ID as private by default.

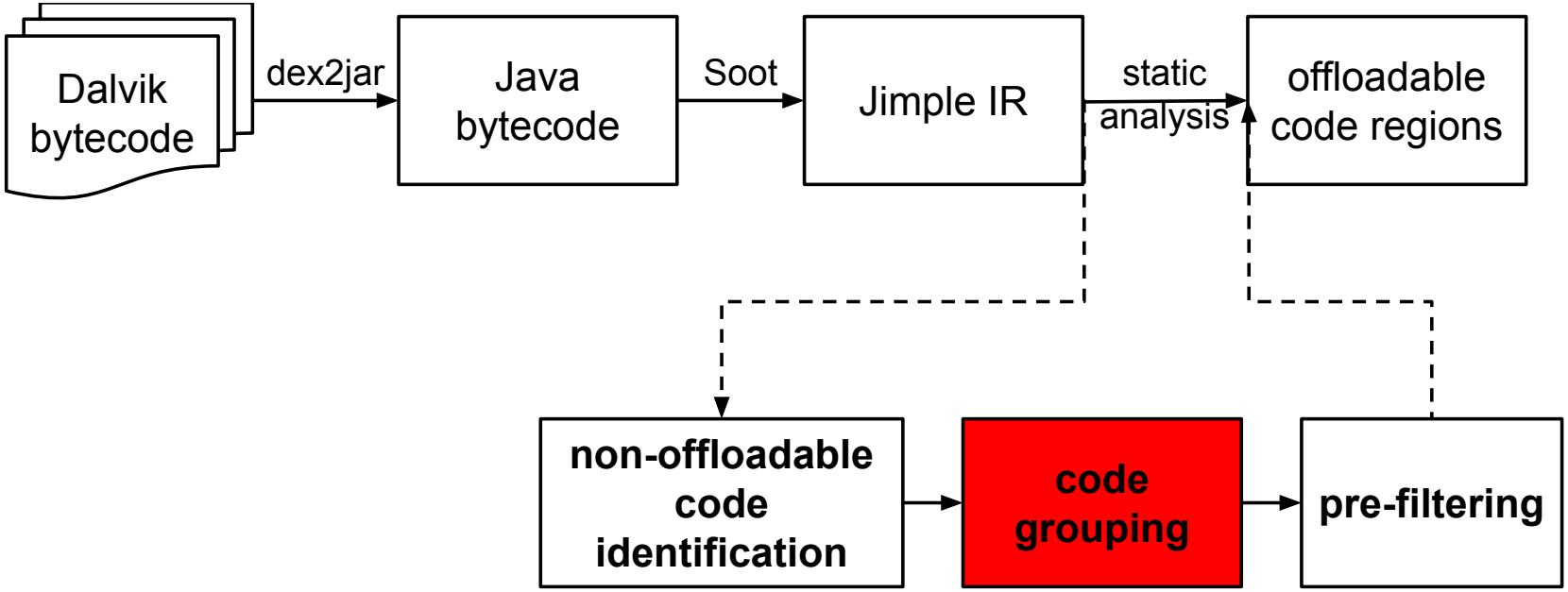
Implementation – Static Analysis



We statically identify those non-offloadable code:

1) private data manipulation statements (real privacy)
2) GUI components (touch event)
3) local resource access (file access)
4) APIs that rely on Android OS or physical device to execute (send text message)

Implementation – Static Analysis



After identifying the non-offloadable code, all other code should be offloadable. We then group those offloadable statements into a number of code regions.

Implementation – Static Analysis

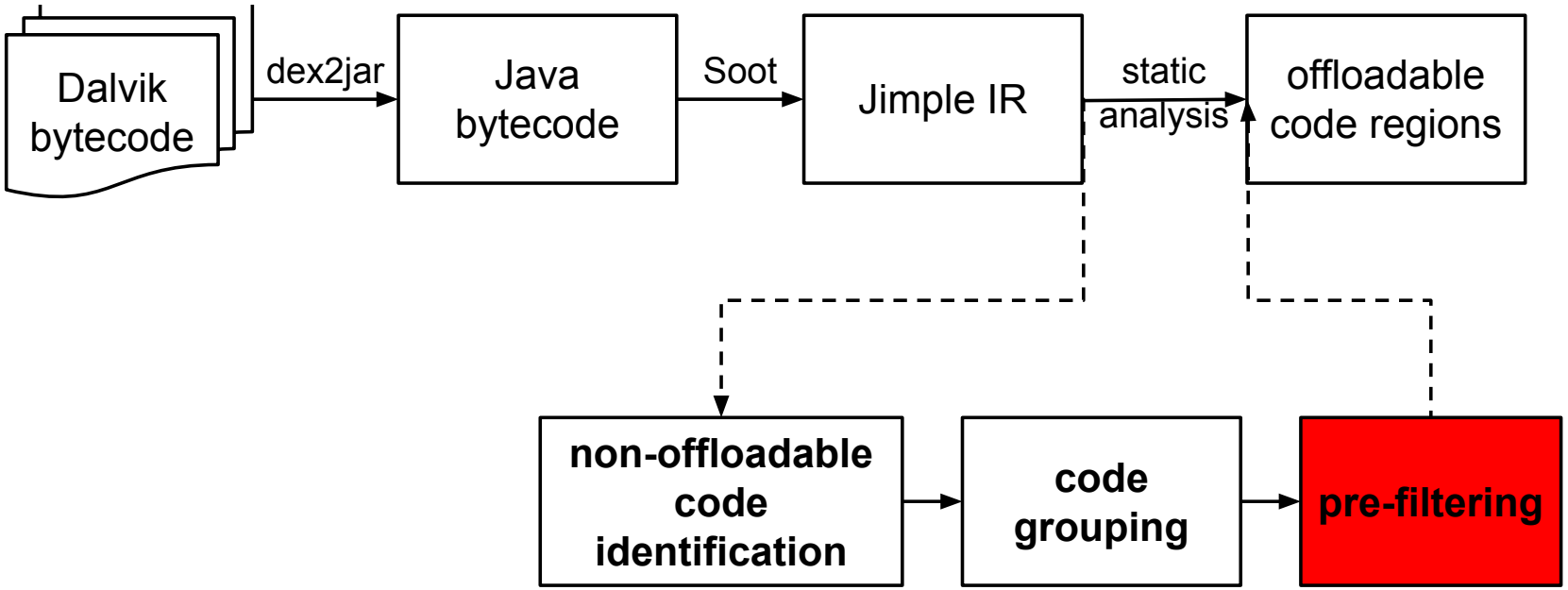
Code grouping has two goals:
1. Keep the original program logic
2. Maximize the code regions to minimize instrumentation and communication overhead

Algorithm 1 achieves these two goals. Please refer to the paper for the details.

Algorithm 1 Offloadable Code Grouping

$Set_{NO} \leftarrow$ all the non-offloadable statements in app
$Set_{method} \leftarrow$ all the methods in app
$Set_{intersplitBranch}, Set_{intersplitTarget} \leftarrow$ null
$Set_O \leftarrow$ null
for $m \in Set_{method}$ **do**
  $cfg_m \leftarrow$ GetControlFlowGraph(m)
  $Set_{cfg'} \leftarrow$ SplitCFG(cfg_m, Set_{NO})
  for $cfg' \in Set_{cfg'}$ **do**
    $Set_{branch} \leftarrow$ GetBranches(cfg')
    if any b in Set_{branch} has a target t out of cfg' **then**
      $Set_{intersplitBranch} \leftarrow Set_{intersplitBranch} \cup b$
      $Set_{intersplitTarget} \leftarrow Set_{intersplitTarget} \cup t$
    end if
  end for
end for
for $m \in Set_{method}$ **do**
  $cfg_m \leftarrow$ GetControlFlowGraph(m)
  $Set_{cfg''} \leftarrow$ SplitCFG(cfg_m, $Set_{NO} \cup Set_{intersplitTarget}$)
  $Set_{cfg''} \leftarrow$ DeleteNode($Set_{cfg''}$, Set_{NO})
  $Set_{cfg''} \leftarrow$ SetAsReturn($Set_{cfg''}$, $Set_{intersplitBranch}$)
  $Set_O \leftarrow Set_O \cup Set_{cfg''}$
end for
output Set_O as a set of offloadable code regions

Implementation – Static Analysis



Pre-filtering is performed to filter out small code regions that are not computation-intensive.

Do not contain loops and have a small number of statements. Because potentially, offloading those small code regions will not bring benefit.

Implementation - Instrumentation

- Make a copy of offloadable code regions in the app
- Each copied code region becomes one RPC method
- Instrument the original program by inserting decision making code and remote procedural calls.

```
void EditPicture(Bitmap bm) {
    boolean callRemote = makeDecision();
    if(callRemote){
        callRPCEditPicture1(bm);
    }else{
      //heavy computational picture editing
      bm.getPixels(pix, 0, mPhotoWidth, 0, 0,
     mPhotoWidth, mPhotoHeight);
      ...
    }
    //embed location information into picture
    Location loc = getLastKnownLocation(Provider);
    ...
    EmbedLocation(bm, longitude, latitude);
}
```

Implementation – Runtime decision

- Create an Android service to make offloading decisions based on runtime information (network)
- **Service component** runs in the background and returns true/false to allow/disallow offloading.

```
void EditPicture(Bitmap bm) {
    boolean callRemote = makeDecision();
    if(callRemote){
        callRPCEditPicture1(bm);
    }else{
      //heavy computational picture editing
      bm.getPixels(pix, 0, mPhotoWidth, 0, 0,
     mPhotoWidth, mPhotoHeight);
      ...
    }
    //embed location information into picture
    Location loc = getLastKnownLocation(Provider);
    ...
    EmbedLocation(bm, longitude, latitude);
}
```

Overview

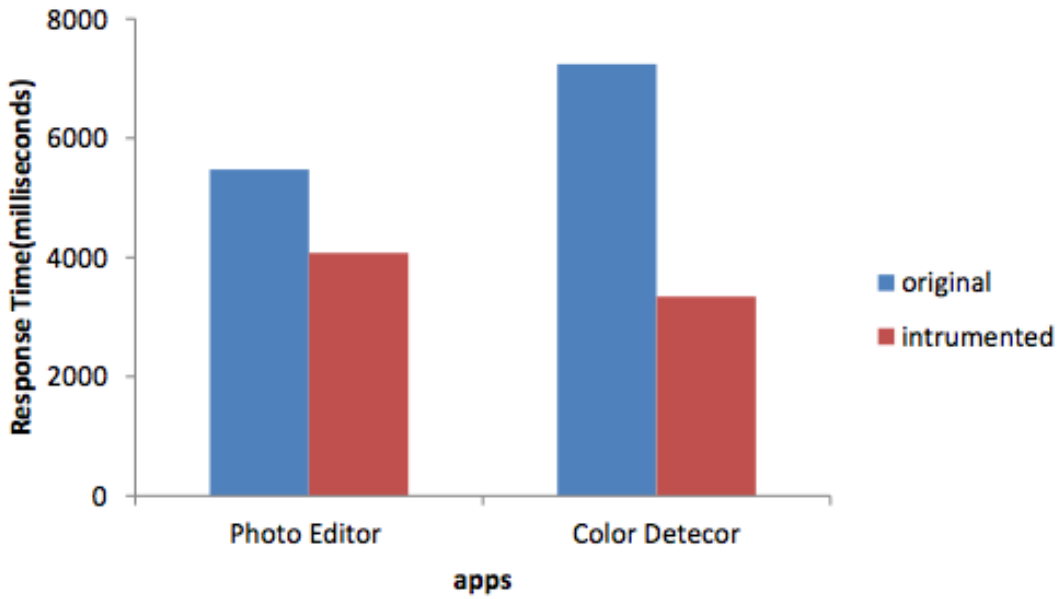
- Introduction to offloading
- Motivating example
- System design & goals
- Implementation details
- **Preliminary evaluation**
- Discussion

Preliminary evaluation

- We conduct our preliminary evaluation on a Nexus S model device running Android 4.0.4 in local network environment.
- We evaluate our system with two Android applications.
  - Photo editor that embeds location information into the picture
  - Color detector that shows text in different languages based on current locale setting

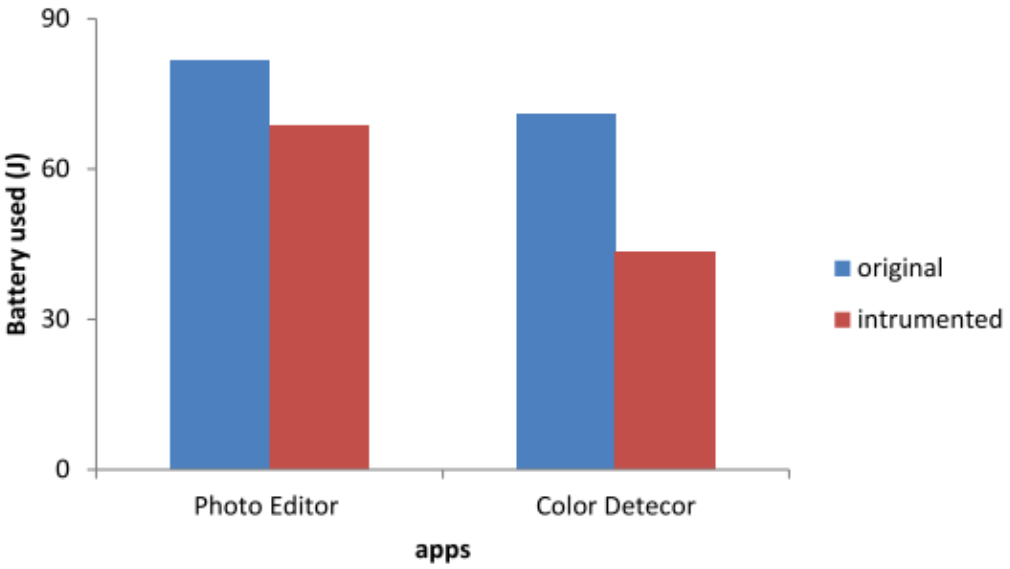
Preliminary evaluation

Runtime performance: speed up by 25.5% and 53.8% respectively for the two apps.



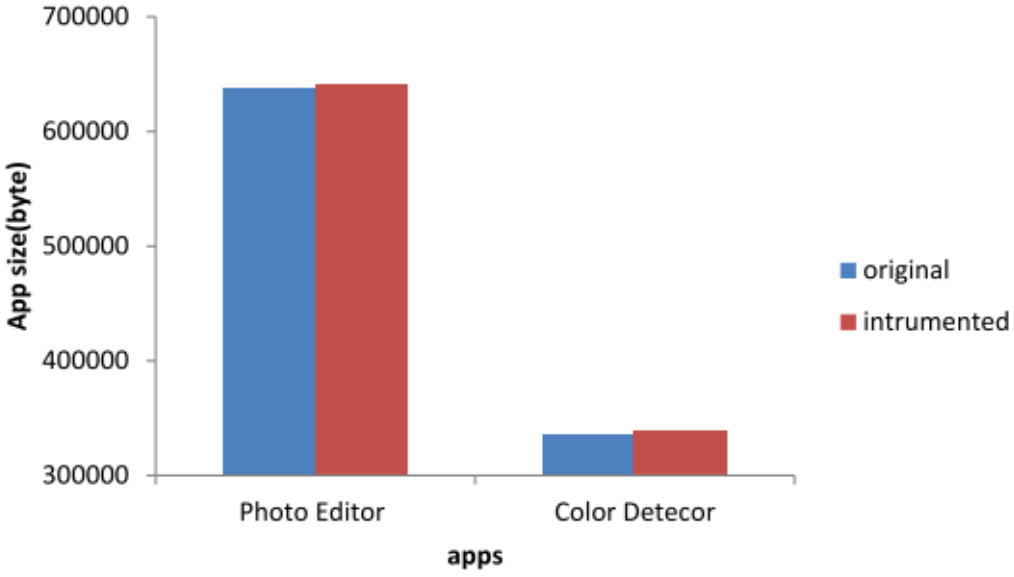
Preliminary evaluation

Power consumption: reduced by 15.9% and 38.7% respectively.



Preliminary evaluation

App size: negligibly increased by 0.4% and 0.8%.



Discussion

- **Dynamic privacy policy configuration**

So far, our privacy configuration has been static. We would like to extend it to a dynamic fashion. This requirement might completely change the system design.

- **Communication Versus Computation**

How to quantify the benefits of offloading using various of system and program information?