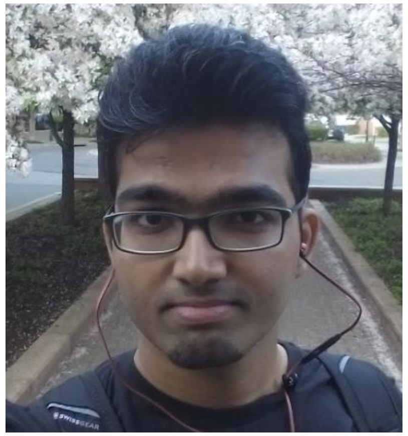
Real-Time Cyber Physical Systems Application on MobilityFirst

Winlab Summer Internship 2015

Karthikeyan Ganesan, Wuyang Zhang, Zihong Zheng

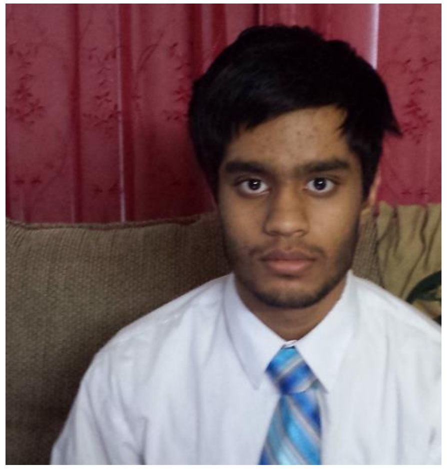
Shantanu Ghosh, Avi Cooper

TEAM MEMBERS


Karthikeyan Ganesan

Wuyang Zhang

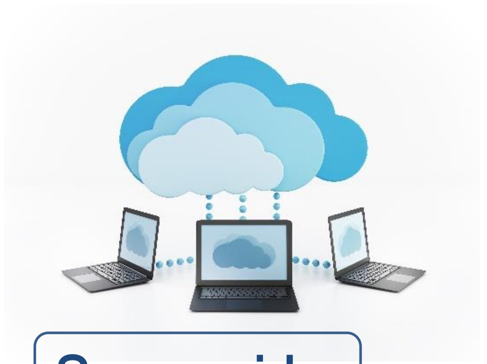
Zihong Zheng


Shantanu Ghosh

Avi Cooper

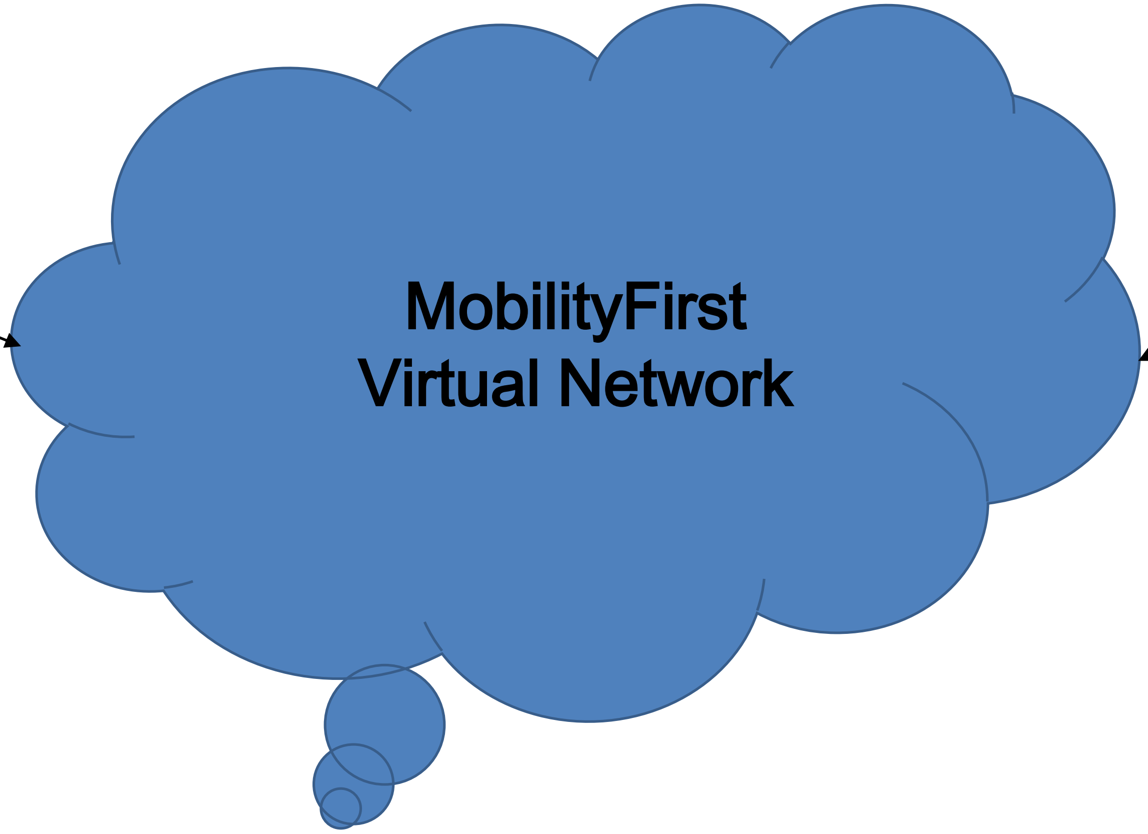
PRELIMINARY GOAL OF OUR PROJECT

CPS Application based on MF



Server side:

Implement server application for object recognition;
Return the result



Client side:

Run an instance of camera system;
Transmits video in standard format;
Simple graphical interface to display results

CURRENT FRAME

Image Recognition:

Optimized the strategy for Descriptor Searching.

Application:

Finished debug the Android version MFstack and MFping programs.
Successfully set up the connection between phone and nodes.
Bluetooth transmitting part now works more smoothly and reliably.

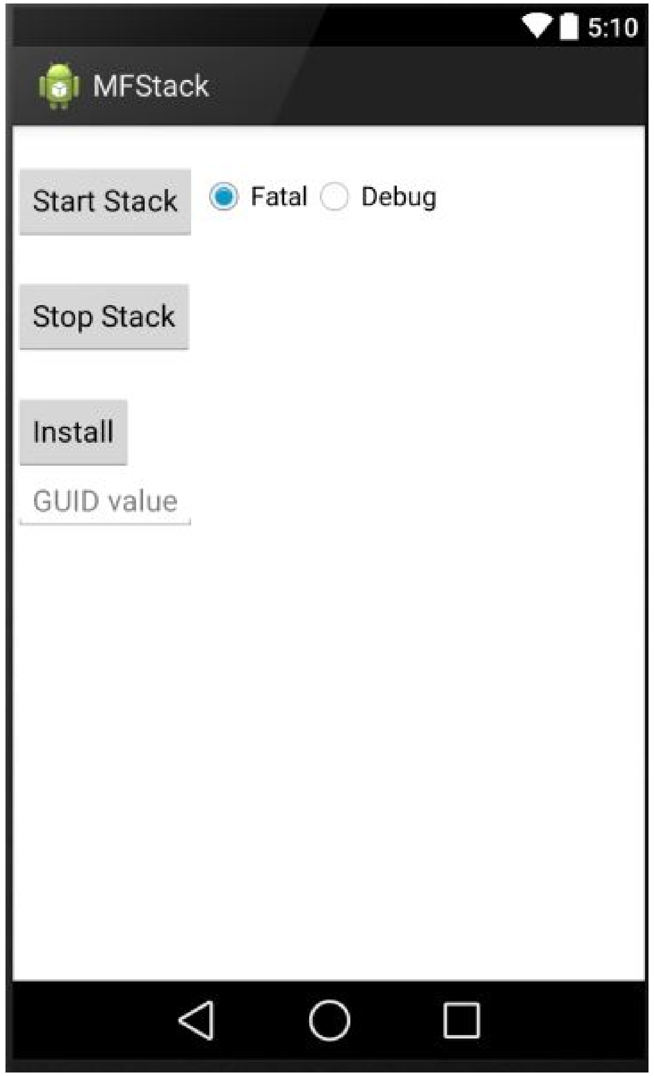
Cloud Computing:

Transformed the image recognition code to be compatible with STORM using Java Native Interface(JNI).

Image Processor Optimization

- Used a K-means tree for Descriptor search
  - Increased search speed by at least 2 times
  - Relatively large amount of time spent building tree
- Still working on GPU implementation of SURF feature detector and extractor

ANDROID PHONE

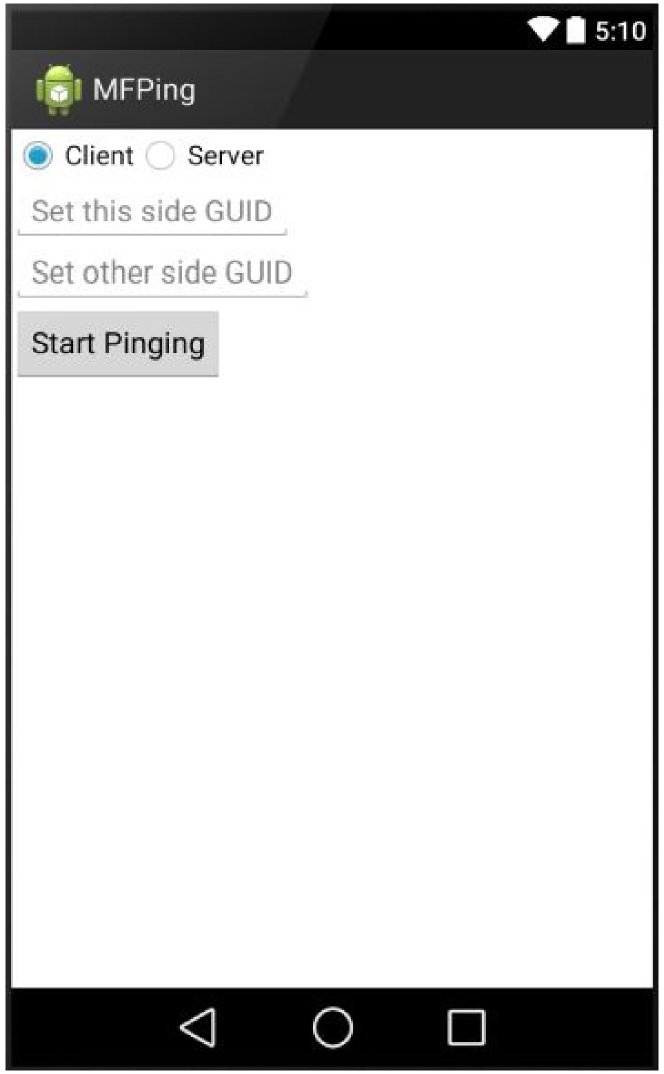


[✓ Fatal] Debug

MFStack is used to install the MF stack on phone.

Also a launcher to start and stop the mf stack service.

Devices under MF network is actually communicating through the MF stack.

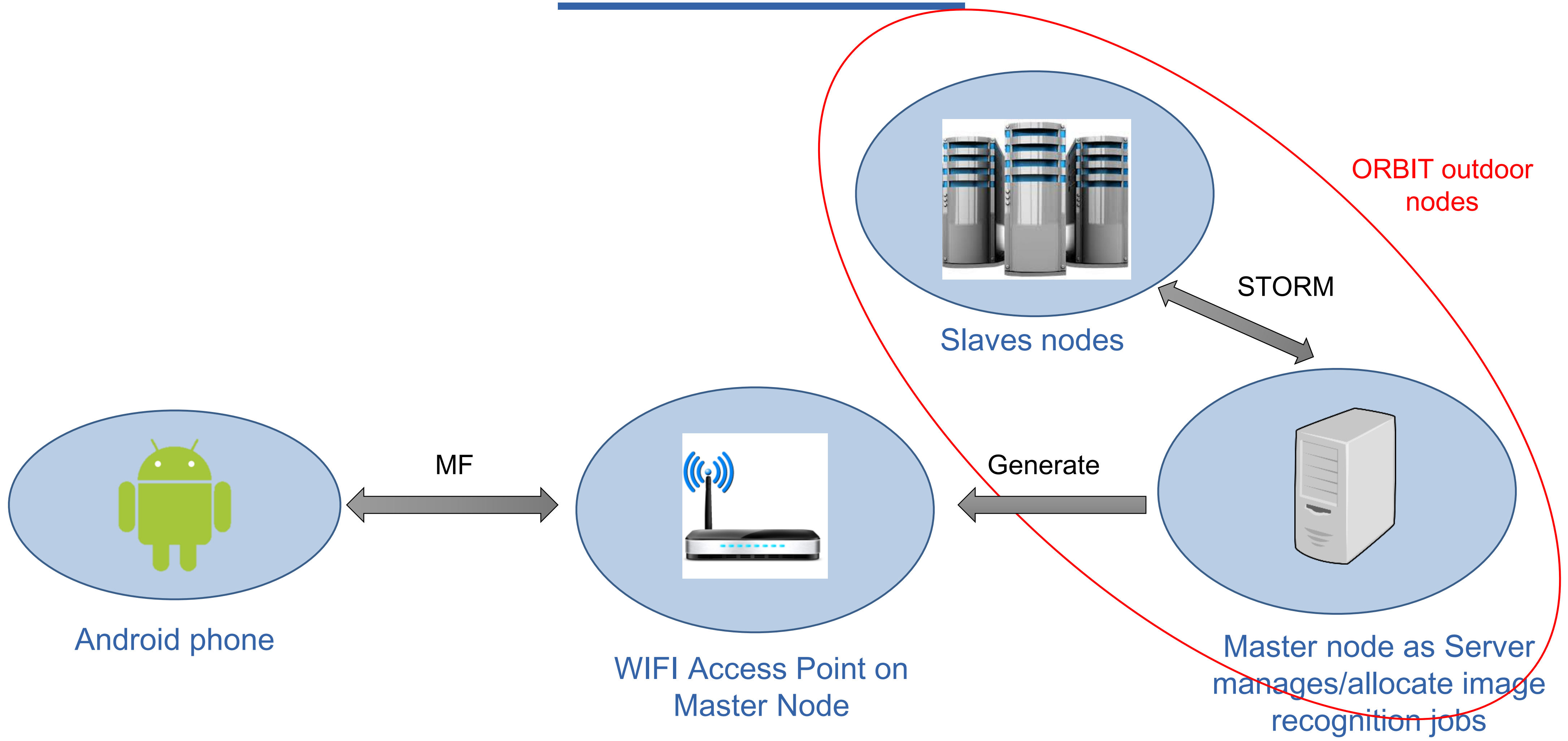


[✓ Client] Server

MFPing achieves the basic ping function for MF such as the ping for TCP/IP.

We used it to test the MF connection after we set up the access point on node.

SET UP THE CONNECTION


ORBIT outdoor nodes
STORM
Slaves nodes
MF
Generate
Android phone
WIFI Access Point on Master Node
Master node as Server manages/allocate image recognition jobs

SET UP THE CONNECTION

Solution:

Create an access point on one of the node using hostapd(an application). Make the Android Phone connect to the AP and hence make the node and phone both in the same subnet.

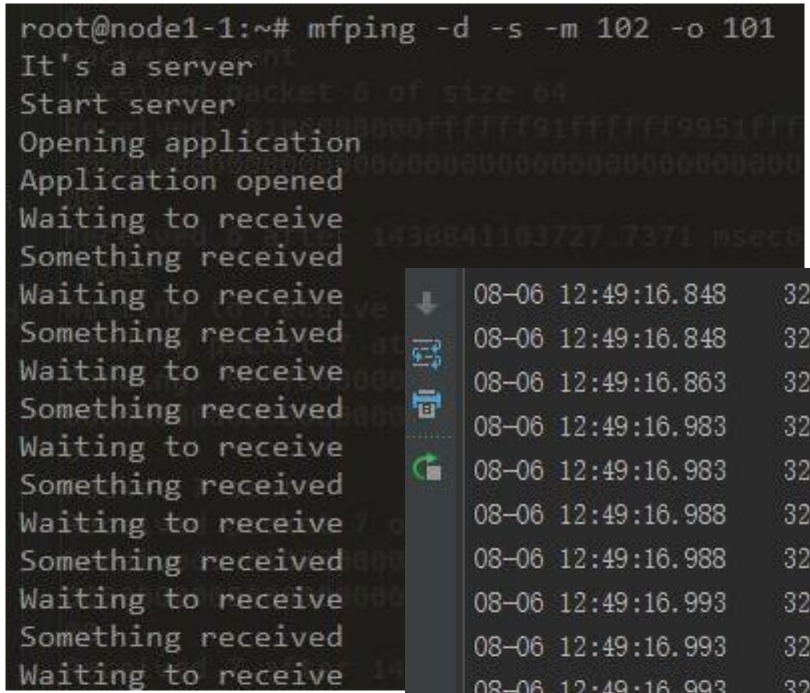
Enable the MF direct communication between these two machines (without set up MF router to simplify the process).

Progress:

Successfully create an access point on SB5 node1-1, finished configure the MFstack on the machines. Both Android phone and Laptop are now able to communicate with nodes using MF network.

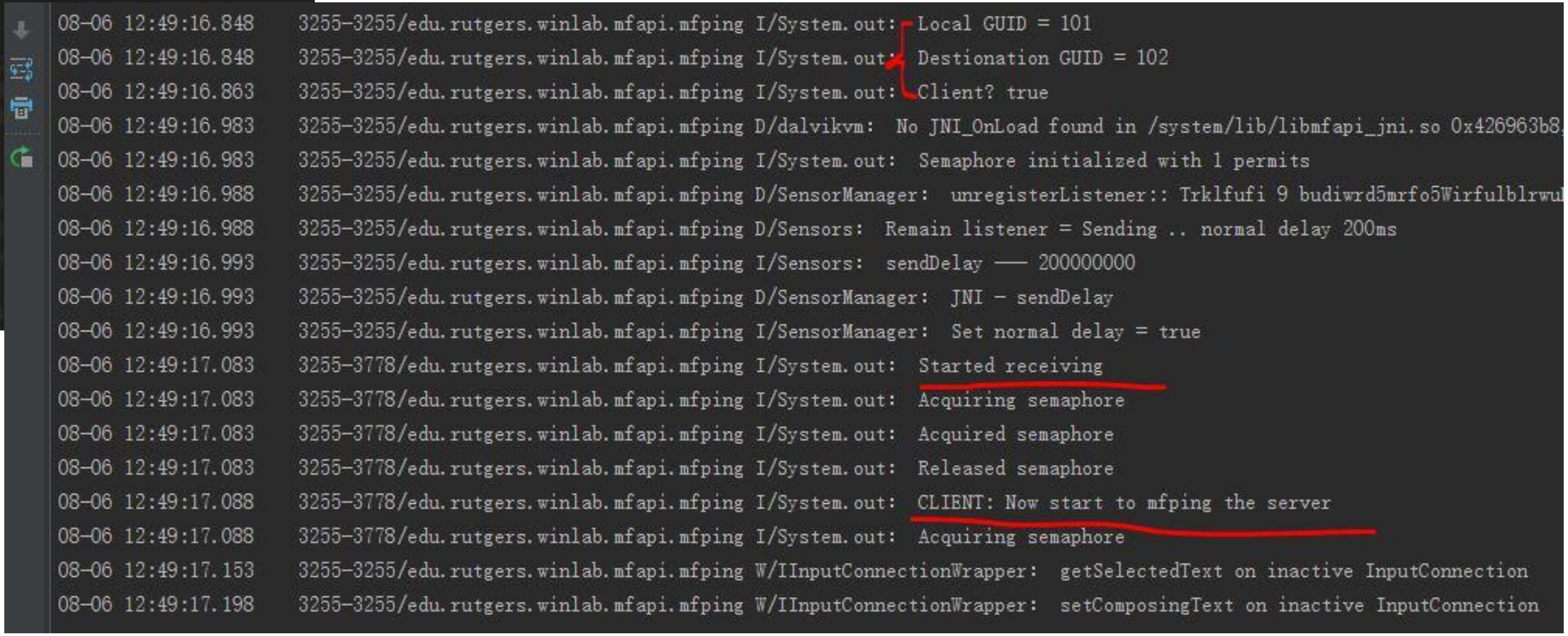
SET UP THE CONNECTION

Node1-1



```
root@node1-1:~# mfping -d -s -m 102 -o 101
It's a server
Start server
Opening application
Application opened
Waiting to receive
Something received
Waiting to receive
Something received
Waiting to receive
Something received
Waiting to receive
Something received
Waiting to receive
Something received
Waiting to receive
Something received
Waiting to receive
```

Android Phone

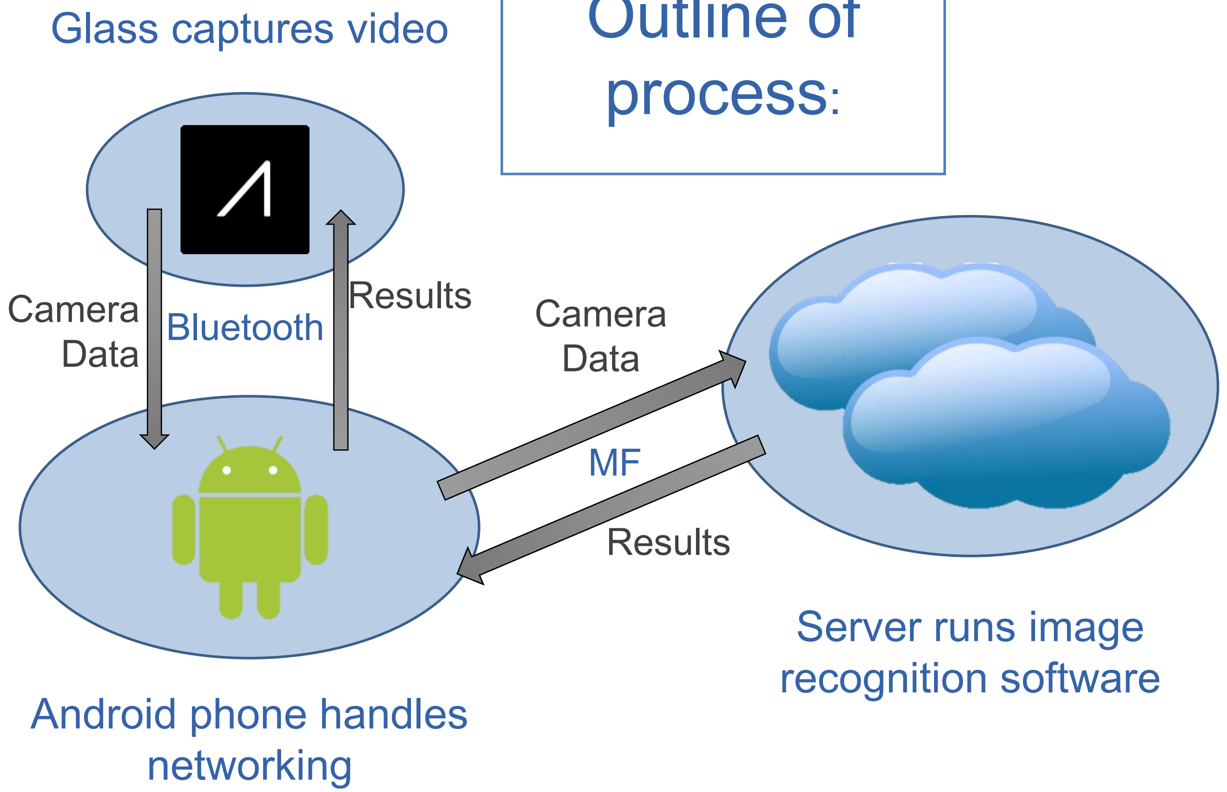


```
08-06 12:49:16.848    3255-3255/edu.rutgers.winlab.mfapi.mfping I/System.out: Local GUID = 101
08-06 12:49:16.848    3255-3255/edu.rutgers.winlab.mfapi.mfping I/System.out: Destionation GUID = 102
08-06 12:49:16.863    3255-3255/edu.rutgers.winlab.mfapi.mfping I/System.out: Client? true
08-06 12:49:16.983    3255-3255/edu.rutgers.winlab.mfapi.mfping D/dalvikvm:  No JNI_OnLoad found in /system/lib/libmfapi_jni.so 0x426963b8
08-06 12:49:16.983    3255-3255/edu.rutgers.winlab.mfapi.mfping I/System.out: Semaphore initialized with 1 permits
08-06 12:49:16.988    3255-3255/edu.rutgers.winlab.mfapi.mfping D/SensorManager:  unregisterListener:: Trklfufi 9 budiwrd5mrfo5Wirfulblrwul
08-06 12:49:16.988    3255-3255/edu.rutgers.winlab.mfapi.mfping D/Sensors: Remain listener = Sending .. normal delay 200ms
08-06 12:49:16.993    3255-3255/edu.rutgers.winlab.mfapi.mfping I/Sensors:  sendDelay --- 200000000
08-06 12:49:16.993    3255-3255/edu.rutgers.winlab.mfapi.mfping D/SensorManager:  JNI - sendDelay
08-06 12:49:16.993    3255-3255/edu.rutgers.winlab.mfapi.mfping I/SensorManager:  Set normal delay = true
08-06 12:49:17.083    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: Started receiving
08-06 12:49:17.083    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: Acquiring semaphore
08-06 12:49:17.083    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: Acquired semaphore
08-06 12:49:17.083    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: Released semaphore
08-06 12:49:17.088    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: CLIENT: Now start to mfping the server
08-06 12:49:17.088    3255-3778/edu.rutgers.winlab.mfapi.mfping I/System.out: Acquiring semaphore
08-06 12:49:17.153    3255-3255/edu.rutgers.winlab.mfapi.mfping W/IInputConnectionWrapper:  getSelectedText on inactive InputConnection
08-06 12:49:17.198    3255-3255/edu.rutgers.winlab.mfapi.mfping W/IInputConnectionWrapper:  setComposingText on inactive InputConnection
```

Google Glass

Why Google Glass? Glass provides us with access to both a camera that is recording exactly what the user is seeing, and a way to give information back to the user unobtrusively. What the user sees and how they sees it is very important.

Outline of process:


Glass captures video
Camera Data
Bluetooth
Results
Camera Data
MF
Results
Server runs image recognition software
Android phone handles networking

Why the phone as a go between? Glass is relatively low on battery and computing. Transferring networking to a phone will improve both battery life and networking speeds.

Bluetooth transmission

Bluetooth specs: Bluetooth can only send 20 byte packets at a time so my solution was to:

1. Split the byte array of the file (the image from the camera) into 18 byte packets
2. Give each packet a heading of its place among the other packets:
    1. each byte has a range of 256. So with 3 counter- header bytes, times the 17 important bytes in the packet, there is a maximum transmission size of 285,212,672 bytes, or 272 MB. (With only 2 header- bytes, though there are 18 body- bytes, the maximum transmission size will only be 1.168 MB)
    2. Update: pictures taken on the Google Glass have been compressed and now only require 52.734 KB, so the switch was made to 2 headers and 18 body bytes
3. Receive each packet on the server side and place it into a 2D array, ordered by their headers.
4. Iterate though the array, pulling out only the body- bytes and place them into a single byte array.

byte array layout: [counter, counter, byte 1, byte 2, byte 3, byte 4, byte 5, byte 6, byte 7, byte 8, byte 9, byte 10, byte 11, byte 12, byte 13, byte 14, byte 15, byte 16, byte 17, byte 18]

Bluetooth transmission (cont.)

Packet corruption: Since the packets can get corrupted in their transmission, due to noise interference, I have implemented the following checker.

1. Transmit each packet 3 times
2. After all the packets are received, check that at least 2 of the 3 match each other. If none of them match, the phone asks for the packet again, and checks it the same way.
    1. Since the packets are sent at totally different times, the chance that the packets would be corrupted in the exact same way is practically zero

Findings: For every transmission, about 1 packet from the first send, the one of critical importance in the previous version without any checker, is corrupted. This is a very small amount compared to the thousands of packets being sent, but even one lost packet, or up to 20 lost bytes, can drastically change the color or the placement of large sections of the picture.

Google Glass- Progress

This week's progress: Last week I got the picture transmission working with transmission time at about 5 seconds and routinely losing at least 5 packets. It is now at 1 second and losing no packets. The android Bluetooth APIs have been implemented on the Glass and the phone and the data transmission of pictures works with 100% accuracy with about 1 second of transmission time. Separately, the MobilityFirst client code has been implemented in Java.

Next week's Goals: The continuous camera data collection will be implemented from a project written by another student in the past for a Google Glass facial recognition application. Also, the MobiltyFirst client code will be integrated with the existing Bluetooth transmission system to push the data further along to the server. Lastly, a system to tell the user the results of the image recognition algorithm needs to be designed.

Next Week Plan

Continue develop the client program based on google glass and Android phone.

Do some experiments to figure out the maximum QPS(query per second) of our current system.

Base on the load experiments, decide the strategy and implement the server program with STORM Framework.

Questions?