MobilityFirst Software and API

MobilityFirst Workshop

Organized as a part of:
Project: Engaging undergraduates in research that speaks their language
Sponsoring Agency: The Thurgood Marshall College Fund
Sponsoring program: Undergraduate Research to Retain and Graduate Students in STEAM

Shweta Jain, PH.D.
Assistant Professor and Doctoral Faculty
sjain@york.cuny.edu

York College of CUNY

Slides based on: F. Bronzino, K. Nagaraja, I. Seskar, D. Raychaudhuri, "Network Service Abstractions for a Mobility-Centric Future Internet Architecture" MobiArch 2013

Overview

MobilityFirst API: Network Service Abstractions

- Name based services
- Direct addressability of all network principals
- Trust and privacy
- Point to multi-point communication
- In-network storage and computation

Each abstraction translates to a function in the network architecture

Network Architecture

Name based service

Requirement: Refer to everything by name and seamlessly handle mobility of moving targets
Architecture provides: Dynamic and fine-grained location resolution

Direct addressability of all network principals

Requirement: Virtually inhaustible name space
Architecture provides: 256 bit space to encode names

Trust and privacy

Requirement: Longer names with flat structure (for location privacy)
Architecture provides: Public key based name and name assignment and resolution services

Network Architecture

Point to multi-point communication

Requirement: Suport to create and manage groups of member IDs dynamically available to the routing fabric
Architecture provides: Routing protocol that supports unicast, multicast, anycast, multihoming and late binding of name to location

In-network storage and computation

Requirement: Storage and computation capability in the routers or attached to routers
Architecture provides: Storage aware routing and content caching capabilities in routers

MobilityFirst API: Basic Operations

Endpoint/Socket creation, customization and teardown

Open a mobilityFirst socket

```
open ( profile, [profile-opts],[src-GUID])
```

- `Profile` is a descriptive element that inform the system about the intent of the application.
- `Profile-opts` are optional additional parameters that to request additional services expressed as flags
- `GUID-set` is optional and represents the set of GUIDs that the application is willing of listening at

Close the session initiated with an open call and clear the resources that were allocated.

```
close();
```

MobilityFirst API: Basic Operations

Name-based messaging

```
send(dst-GUID, data, [svc-opts]);
```

- `dst-GUID` represents the destination GUID.
- `data` is the data message to be sent.
- `svc-opts` are additional service options used to determine how the message is delivered expressed as flags.

```
recv(src-GUID, buffer, [GUID-set]);
```

- `src-GUID` is a variable that will be filled with the GUID of the entity that sent the message
- `buffer` is filled with the message that has been transmitted
- `GUID-set` is optional and can be used to limit the set of GUIDs to receive from

MobilityFirst API: Basic Operations

Management of network presence

Add additional GUIDs at other than those originally identified during the open call

```
attach(GUID-set);
```

- `GUID-set` represents the set of GUIDs that the application wants to add at the set that it is listening at

Remove GUIDs from the one originally identified during the open call or the ones added through the use of `attach`

```
detach(GUID-set);
```

- GUID-set represents the set of GUIDs that the application wants to remove from the set that it is listening at

MobilityFirst API: Basic Operations Example

Filename: mf-usage.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mfapi.h>
int main(int argc, char *argv[]) {
        struct Handle handle;
        int mine, other, sent = 0, received = 0;
  char role;
        int size = 65*1024;
        int *recvFrom;

       // --maybe read a file and save it in  the buffer
        u_char buf[size];
        role = atoi(argv[1]);
  mine = atoi(argv[2]);
  other = atoi(argv[3]);
```

MobilityFirst API: Basic Operations Example

```
int ret = mfopen(&handle, "basic\0", 0, mine);
if(ret) {
        fprintf(stderr, "receiver: mfopen error\n");
        return (EXIT_FAILURE);
}
if (role == 's')
{
    sent = mfsend(&handle, buf, size, other, 0);
    if (sent < 0) {
        fprintf (stderr,"mfsendmsg error\n");
        return EXIT_FAILURE;
    }
}
```

MobilityFirst API: Basic Operations Example

```
       else
    { //Wait to receive new message
          received = mfrecv_blk(&handle, recvFrom,
                                      buf, size, 0, 0);
           if (received < 0)
           {
               fprintf (stderr,"mfrecv_blk error\n");
               return EXIT_FAILURE;
           }
       }
       printf("Intended to send %d bytes,
                 sent %d bytes, received %d bytes\n",
                 size, sent, received);
      mfclose(&handle);
      return EXIT_SUCCESS;
}
```

Setting up a MobilityFirst network

Download, prepare and install MobilityFirst stack (For Linux and android)

Use this script to streamline your installation on ubuntu based machines:
`"mobilityfirst_install_script.pdf"` (click to open file)
Remember to adjust the commands for your credentials for bitbucket.org

MobilityFirst API: Deployment Steps

1. Define network graph and assign GUIDs to each routing element and end-host. Current implementations accept a 32-bit integer for GUID.
2. Translate network graph to a corresponding GUID-based topology file
3. Establish IP connectivity between all routing elements
4. Bring up GNRS instances (for simple networks, single-server GNRS configuration may suffice). Test the GNRS service using sample command line clients in the release.
5. Bring up a MF routers
6. Bring up MF protocol stacks on each end-host. Host stacks can determine their access router either automatically (by latching onto a period broadcast beacon), or can be forced to associate with a particular one by specifying router MAC and IP.
7. Run your application

Steps 1 and 2: Define network graph and generate topology file

Simple experiment topology:

```
                --------
Sender ------| Router |------ Receiver
(GUID 1)     |(GUID 2)|        (GUID 3)
                --------
```

Create a topology for the experiment above and save it in topo.tp:
Syntax of the topology file is:

```
<GUID> <number of neighbors> <List of neighbor GUIDs>
```

The above example topology is given below:

```
1 1 2
2 2 1 3
3 1 2
```

Step 3: Establish IP connectivity between all routing elements (Wi-Fi Instructions)

Things to consider before proceeding in this step:

- For first time users, a wired set up is much easier to work for example a lab with networked PCs (typical undergraduate teaching lab)
- The wifi hostapd setup works only with wi-fi cards that are able to work in *master* mode (typically Atheros chipsets). For all others, use an access point to establish IP connectivity

Setup up hostapd and dnsmasq on the middle routing element (GUID 2) using instructions in this file: `"hostapd_dnsmasq.pdf"`

Connect all routing elements and end hosts to the access point that was created using the above instructions

Steps 4 and 5

Step 4: Bring up GNRS instances

```
cd $MF_HOME/gnrs/jserver
java \
-Dlog4j.configuration=file:sample-configs/single-server/log4j.xml \
-jar target/gnrs-server-1.0.0-SNAPSHOT-jar-with-dependencies.jar \
 sample-configs/single-server/server.xml
```

Step 5: Start the MobilityFirst Basic Router

```
cd $MF_HOME/router/click/conf
export MF_CLICK_LOG_LEVEL=1
sudo -E click -j 4 MF_BasicRouter.click topo_file=topo.tp \
                  my_GUID=2 core_dev=wlan0
```

Step 6: Bringing up the MobiliyFirst stack on the sender and receiver

Settings file at the sender (GUID 1)

```
INTERFACE = wifi,wlan0,manual, 192.168.1.1, <mac address
                                   of the router (GUID 2)>
POLICY = wifionly
BUFFER_SIZE = 10
DEFAULT_GUID = 1
IF_SCAN_PERIOD = 5
```

Settings file at the receiver (GUID 3)

Same as above except: DEFAULT_GUID = 3

Bring up the MobilityFirst stack on the two end hosts

```
sudo cp $MF_HOME/hoststack/bin/mfstack /data/mfdemo
sudo /data/mfdemo/mfstack -d /data/mfdemo/settings.conf
```

Compiling and running the example application on Linux

Sender application

```
gcc mf-usage.c -I$MF_HOME/common \
    -I$MF_HOME/netapi/c -lmfapi -lpthread  -o mf_usage
sudo ./mf_usage s 1 3
```

Receiver application

```
gcc mf-usage.c -I$MF_HOME/common \
    -I$MF_HOME/netapi/c -lmfapi -lpthread  -o mf_usage
sudo ./mf_usage r 3 1
```

Comments or Questions?

Homework

Final reports are due next week
Blog about what you like, dislike or think about the paper and this presentation