ネットワークエミュレータ GINE への Android の組み込み

M2009MM003 馬場 隆章
指導教員：河野 浩之

1 はじめに

近年, Android などを搭載したスマートフォンが普及し, 企業だけでなく一般のユーザもスマートフォン用のアプリケーションを提供できるようになり注目を集めている. なかでもネットワークにどこでも接続できる特性を活かしたアプリケーションも多く開発されている.

そこで本稿では, スマートフォンの容易な通信実験環境を構築するためにネットワークエミュレータ Goto's IP Network Emularor[1] (以下 GINE) に Android エミュレータを組み込むことにより, Android のネットワークアプリケーションを安価で, 容易に構築できるようにする.

また Android エミュレータを GINE に組み込むさい発見した Android エミュレータを外部と直接通信させる方法と, GINE で構築した実験環境での Proxy を介した Android からの Web アクセスの方法についても考察する.

2 Android エミュレータとネットワーキング

各 Android エミュレータはオープンソースの PC エミュレータである QEMU 上で動いており, 仮想ルータと仮想ファイアウォールを起動してホストマシン上で独立したネットワークを構成する. そのため, 他の Android エミュレータを認識できない. 仮想ルータの管理する IP アドレスは 10.0.2/24 で, ホストマシン上にディフォルトで表 1 のように IP アドレスが割り当てられ, Android エミュレータから見たネットワークは図 1 のようになる.

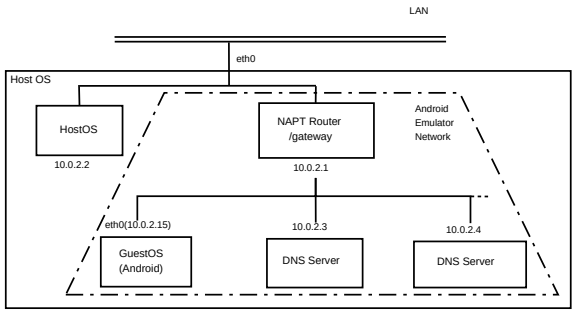

図 1 Android エミュレータから見たネットワーク図

DNS サーバはホスト OS の`/etc/resolv.conf` (`/etc/host` は無視される) を読み込み, エイリアスとして設定される. また, Android エミュレータは開発マシン上のアプリケーションとして動作するため Android エミュレータはホスト OS から見て図 2 のように動いている. そのため, Android エミュレータはホスト OS のネットワーク設定の制約のもとで通信する. また, Android エミュレータは本稿執筆時に IPv6 や IGMP, マルチキャストなどに対応していない.

そこでここでは QEMU の tap 機能を使うことにより, Android エミュレータを外部と直接通信できるように設定

表 1 Android エミュレータのネットワークアドレス構成

IP アドレス	説明
10.0.2.1	ゲートウェイアドレス/仮想ルータ (NAPT)
10.0.2.2	ホストマシンへのループバックインタフェース. ホストマシンにアクセスするために使用
10.0.2.3〜10.0.2.6	DNS サーバ. ホストマシンのシステムで使用している DNS サーバのエイリアス (最大 4 つ)
10.0.2.15	Android エミュレータ自身のネットワークインタフェース. ホストマシンから Android エミュレータにアクセスするために使用
127.0.0.1	Android エミュレータ自身のループバックインタフェース

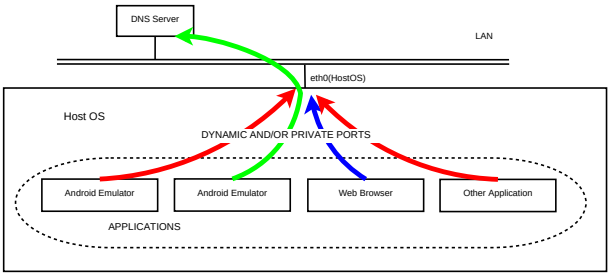

図 2 ホスト OS から見た Android エミュレータ

する. tap 機能は QEMU で作成したゲスト OS とホスト OS をつなぐ tap デバイスを作成する機能のことである.

3 外部ネットワークへのアクセス

図 3 に Android エミュレータを外部と直接通信ができるようにした概要を示す [2].

図 3 のとおり, Android エミュレータの AndroidOS の eth0 とホスト OS の tap を接続し, ホスト OS をルータと見立て AndroidOS の通信を外部ネットワークに中継する.

tap 機能を Android エミュレータで使用する場合, エミュレータを起動するとき次のようなオプションを付けて起動する.

- -qemu -net user,vlan=0 -net nic,vlan=1 -net tap,vlan=1,ifname=tap0

-qemu は QEMU のオプションである. -net nic は Android の NIC を表し, -net tap は指定した tap デバイスを Android エミュレータとつなぐためのオプションである. -net user オプションは QEMU をユーザモードで起動するために必要であり, tap 機能を用いる場合は必要ないがこのオプションがなければ Android エミュレータは起動しない. これは QEMU を Android エミュレータではカ

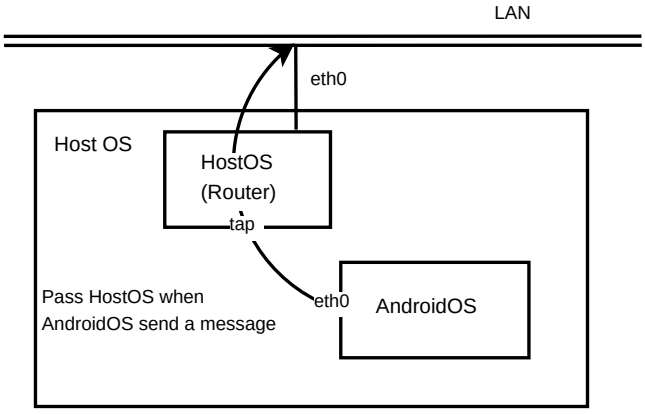

図 3 外部ネットワークへのアクセス

スタマイズしているからである．また-net のオプションすべてについている vlan はどの仮想コネクションに接続するかの指定である．ディフォルトでは vlan は 0 になっている．

これで外部ネットワークに Android エミュレータが直接通信できるようになった．しかし，2 節で述べたように Android エミュレータの IP アドレスが 10.0.2.15 になっているためこのままでは通信が不便である．そこで AndroidOS の IP アドレスやネットワーク設定を変更する．

4 ネットワーク設定の変更方法

Android エミュレータの実体部分を担う system.img を書き換えることにより任意の IP アドレスを設定することができる．

Android エミュレータの IP アドレスとディフォルトゲートウェイの設定は`/system/etc/init.goldfish.sh`に記述されており，これを編集することで任意の IP アドレスを設定することができる．system.img はクロス開発環境のない Linux 上では編集することができないため，Android エミュレータ内で yaffs を用いて編集する．しかし，Android エミュレータを通常起動させた場合ファイルシステムが 100 %になっているため書き換えた system.img を保存できない．そのため以下のオプションをつけて Android エミュレータを起動させる必要がある．

- -partition-size 128

これは Android エミュレータの/system の空き容量を 128MB の容量を増やした状態で Android エミュレータを起動させるためのオプションである．このオプションで起動した Android エミュレータにマウント後，エミュレータの/system を自分の望む状態に書き換えた後，yaffs によりエミュレータの SD カード上に system.img を再構築し，このイメージをローカルディスクに移動する．Android エミュレータを再構築したイメージで起動し，内容が書き換わっていれば成功である．再構築したイメージで Android エミュレータを起動させるために次のオプションを付ける．

- -system “ 再構築したイメージファイル”

5 GINE への接続

3, 4 節にて Android エミュレータが外部と直接通信する方法について述べた．その方法を用いて Android エミュレータと GINE をつなぐための仕組みについて説明する [3]．図 4 に Android エミュレータと GINE の接続図を示す．

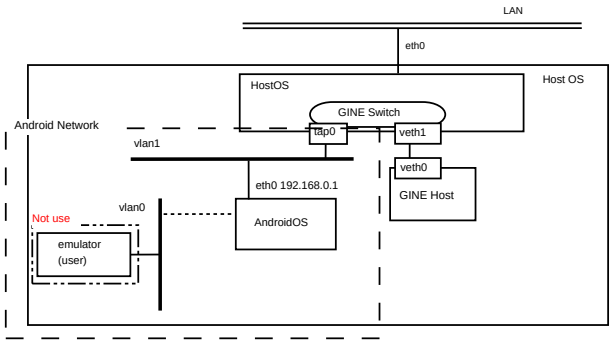

図 4 Android エミュレータと GINE の接続図

ここで注意する点は，AndroidOS の eth0 に対応する tap に IP アドレスを付与しないことである．tap に IP アドレスを付与しないことでホスト OS から IP 的に見えないようにすることで，エミュレーション環境を独立させる．GINE には Switch 機能がありこの Switch に Android エミュレータとつながっている tap をつなぎ，GINE のノードをつなぐ．GINE のノードの実現には NetworkNameSpace(以下 NS) を，NIC の実現には仮想ネットワークデバイスである Virtual Ether Pair(以下 veth) が用いられており，このデバイスはペアデバイスとなっている．そこで GINE のノードの NIC として対応されていない veth を tap が接続されている Switch に接続する．これにより，Android エミュレータと GINE を接続することができる．

また Android エミュレータは最大 16 台動かすことができ，Android エミュレータの起動には開発マシンのメモリをかなり使うことがわかった．多数の Android エミュレータを起動させるためには，次のオプションで Android エミュレータの GUI のサイズを小さくして起動する必要がある．

- -scale 倍率 (0.1〜3)

6 3G/Wi-Fi のエミュレーションの実現

3G/Wi-Fi 環境でのエミュレーションを実現するために，NTT DoCoMo から発売されている HT-03a の実機による通信を実験した．Android の 3G の上りの通信経路を求めるた root 権限が必要な traceroute コマンドを使用したかった．しかし実機の Android は契約や法律上の問題から購入者が root 権限を使用することが認められておらず，traceroute を使用することができない．

そこで root 権限が必要とされていない ping コマンドを用いて通信経路を調べた．ping コマンドはオプションで送信パケットの TTL を決めることができ，途中までの経路を調べることができる．

その結果，上りと下りの通信経路が違っていることがわ

かった. 上りは 211.14.81.30 の mopera.net を必ず経由して目的地まで通信をしているので, ここまでの通信帯域を調べる 3G の上り値とした. また下りの経路は NTT コミュニケーションズと NTT DoCoMo のエリアを必ず通ることがわかった. Wi-Fi についてはアクセスポイント以降の通信経路は PC と同じであるため, アクセスポイントまでの速度を測定した.

3G/Wi-Fi 共に上りの通信速度の測定は ping コマンドの flood を用いて, 下りの通信速度は 1MiB のファイルを http データを GET で取得し, その間の TCP のダウンロードの通信時間をはかるというアプリケーションを作成し測定した.

測定した結果を表 2 にまとめる. また 3G では上りの測定時にパケットロスが 50%発生することがわかった.

表 2 3G/Wi-Fi の通信速度

	上りの速度	下りの速度
3G	1.15Mbps	1.20Mbps
Wi-Fi	4.74Mbps	7.67Mbps

この結果を GINE の FrameQueue クラスを用いて遅延やパケットロス, 帯域制御をすることで 3G/Wi-Fi のエミュレートを実現する.

7 Android 間の通信実験

GINE により接続した Android 間の通信実験の結果を示す. 実験用に構築したネットワーク図を図 5 に示す.

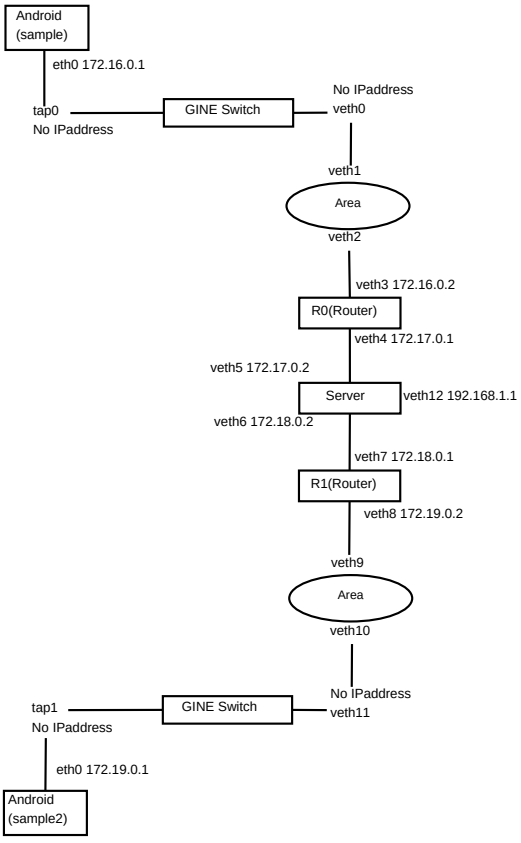

図 5 実験用に構築したネットワーク図 (チャット通信)

実験では自作のソケット通信によるチャットプログラムを 2 台の Android エミュレータで起動し GINE を介してチャットによる通信が正常にやり取りされていることを示す.

構築したネットワーク上で Android エミュレータを 2 台起動する. チャットサーバはメッセージが届いたらチャットサーバに接続している全てのノード (メッセージの送信者含む) にメッセージを送る. Area の部分は 6 節で述べた 3G/Wi-Fi の帯域, パケットロスの設定を施す.

また Android エミュレータを 1 台のホストで複数起動する場合, 起動するエミュレータインスタンスの数だけイメージが必要なので, この実験では 2 台分のエミュレータのイメージを用意した. 起動した Android エミュレータインスタンスをそれぞれ sample, sample2 とする.

図 6 では sample2 でチャットプログラムを起動し, sample からのメッセージを受信した場面である. 下は sample で sample2 から“sample2 ”というメッセージを受信した画像である. sample は先に“sample ”というメッセージを受信しているので, “sample ”の下に“sample2 ”というメッセージを受信している.

図 6 Android 間でのメッセージのやり取り

また 3G/Wi-Fi 環境で通信をした場合の速度も測定した. 結果を表 3 にまとめた. TCP 応答時間は, TCP 接続 (3-way handshake) を含めて, 自分が送信したメッセージが戻ってくるまでの時間である. 後者は相対的に小さく, ほとんどが TCP 接続に要する時間である.

表 3 3G/Wi-Fi の通信時間 (チャット通信)

	TCP 応答時間	メッセージの受信
3G	18.4167s	0.2055s
Wi-Fi	10.0431s	0.0061s

これらにより, GINE を介した Android 間の通信が確認でき, Android エミュレータと GINE の通信実験環境が整えることを示した. また 3G と Wi-Fi の通信時間に差ができており, 3G/Wi-Fi 混合ネットワークで動作するアプリケーションを開発する場合は遅延時間に注意をはらうべきであることがわかった.

この実験を応用することにより, SIP 通信やネットワーク対戦ゲームのエミュレーションも可能である.

8 Android のプロキシ通信

Android は現在, 実機のブラウザにはプロキシ機能がついておらず, Android エミュレータについてはエミュレータ起動時に以下のオプションをつけることによりエミュレータからの通信はプロキシ通信ができる.

- -http-proxy プロキシサーバ名 (もしくは IP アドレス)：ポート番号

しかしこれは, ゲスト OS である Android 自体にプロキシ処理を施しているわけではい.

そのため Android のブラウザから実際にプロキシ通信をさせるためにはどうしてもプロキシ通信用のブラウザを作成する必要がある.

Android では Webkit ベースのブラウザ機能を WebView というビュークラスからアクセスできる. この WebView を http クライアントとして loadUrl を呼び出す. この loadUrl は WebView クラスで用意されているメソッドで, アクセスしたい URL を入力し, そのページの HTML データを取得後レンダリングして出力する.

また, Android ではあらかじめプログラムが必要とする権限を AndroidManifest.xml に記述しておく必要がある.

だがこの方法ではプロキシの設定をすることができない. そこで loadDataWithBaseURL メソッドを用いる. これは HTML データをレンダリングするメソッドである. 標準の java.net の HttpURLConnection クラスを用いてプロキシ接続した後, HTML データを loadDataWithBaseURL メソッドによりレンダリングする. この方法により, ブラウザにプロキシ機能を持たせることができる [4].

作成したプロキシ接続可能なブラウザを用いて実際にプロキシ接続が可能か実験をする. 図 7 に構築した実験用のネットワーク図を示す.

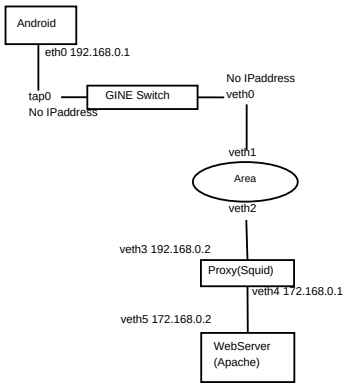

図 7 プロキシ接続によるネットワーク図

なお, プロキシサーバには Squid を, Web サーバには Apache を使用した. Area の部分には 6 節で述べた 3G/Wi-Fi の設定を施す. Android からプロキシサーバにポート 8080 でアクセスし, プロキシサーバから Web サーバへアクセスを確認できた. tcpdump による通信確認を次に示す.

Android からプロキシサーバへのアクセス

```
14:55:31.438824 IP 192.168.0.1.50348 >
192.168.0.2.http-alt: S 2003880798:
略
14:55:31.438875 IP 192.168.0.2.http-alt >
192.168.0.1.50348: S 2032705563:
```

プロキシサーバから Web サーバへのアクセス

```
14:55:31.503453 IP 172.168.0.1.59529 >
172.168.0.2.www: S 2038304630:2038304630(0)
略
14:55:31.503489 IP 172.168.0.2.www >
172.168.0.1.59529: S 2039892791:
```

また 3G/Wi-Fi 環境での通信速度も測定した. まとめた結果を表 4 に示す. firest access, second access は, それぞれ Proxy サーバにキャッシュされていないページ, キャッシュ済みページにアクセスした時の応答時間である. アクセスしたページはテキストのみの短い HTML ファイルである. 3G/Wi-Fi 共に Proxy 通信により通信時間が短縮された.

表 4 3G/Wi-Fi の通信時間 (プロキシ通信)

	first access	second access
3G	10.8274s	10.2667s
Wi-Fi	10.3264s	10.0559s

9 おわりに

Android エミュレータを GINE に組み込むことにより安価で, 容易に Android のネットワークアプリケーションの研究, 開発環境を構築することができた.

今後は別のスマートフォンを組み込み, 大規模なスマートフォン間通信を構築する必要がある.

参考文献

[1] Sugiyama, Y., Goto, K. : Design and Implementation of a Network Emulator using Virtual Network Stack, Proc. of the Seventh International Symposium on Operations Research and Its Applications(ISORA2008, China), Lecture Note in Operations Research, Vol.8, pp.351-358(2008).

[2] 馬場隆章, 後藤邦夫 : Android エミュレータのネットワーキング. 平成 22 年度電気関係学会東海支部連合大会, 愛知, 議演論文集, D3-1(DVD-ROM), (2010).

[3] 馬場隆章, 後藤邦夫 : ネットワークエミュレータへの Android の組み込み. FIT2010 第 9 回情報科学技術フォーラム, 福岡, 公演論文集 第 4 分冊 pp.189-190(2010).

[4] 株式会社ウサギィ, android/proxy の設定 : http://wiki.usagee.co.jp/index.php?android%2Fproxy の設定 (accessed February, 2011)