WiFi与3G混合通信技术阶段性报告（二）

DOC #: 文档编号
Version: v1.0
Date: 2014-02-12

目录

1. 概述

本阶段主要对开发计划中“虚拟混合接口 TCP/IP 协议栈的启用”、“Wi-Fi 接口转发虚拟接口发送的 IP 分组”和“接收端 IP 分组的接收与回应”三个技术难点进行了实验，测试结果达到了预期目标。

2. 虚拟混合接口TCP/IP协议栈的启用

2.1 实现平台

（1）硬件平台：
一台笔记本电脑；
（2）软件平台：
Linux 操作系统；2.6.32 以上版本内核。

2.2 实现方法

用 C 语言编写了一个测试程序，生成一个虚拟网络接口，获取上层应用程序的数据，并将数据封装

到虚拟接口的 TCP/IP 协议栈中。

2.3 验证结果

（1）本测试程序以 ICMP 协议为例进行了实验。测试程序能够启用虚拟网卡，并且可以直接回应 ICMP 请求，自动回复 ICMP-echo 包。通过修改路由表，使得所有应用程序的数据都通过虚拟网络接口转发。由于本程序可以处理其中的 ICMP-echo 包，并自动进行响应，使得上层的应用程序能够顺利得 ping 通一个不存在的主机。

（2）测试程序启动后，使用 Wireshark 软件检测可以看到它收到了很多 IP 分组。当使用 ping 程序发出 ICMP 包时，会被程序检测到，并直接进行回复。测试可以看到，此时可以“ping”到原本不存在的 IP 地址。

综上所述，可以确认，这种方式能够将上层数据封装到为 IP 分组内，IP 层向上的 TCP/IP 协议栈已经启用。本阶段完成了“虚拟混合接口 TCP/IP 协议栈的启用”的测试，达到了预期目标。

```
→  ~  route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.1     0.0.0.0         UG    0      0        0 wlp3s0
10.0.0.1        0.0.0.0         255.255.255.255 UH    0      0        0 0@
192.168.1.0     0.0.0.0         255.255.255.0   U     9      0        0 wlp3s0
→  ~  ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.239 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.207 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.198 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.198/0.214/0.239/0.024 ms
```

3. Wi-Fi接口转发虚拟接口发送的IP分组

3.1 实现平台

（1）硬件平台：

一台笔记本电脑，启动了 Wi-Fi 连接；

（2）软件平台：

Linux 操作系统；2.6.32 以上版本内核。

3.2 实现方法

用 C 语言编写了一个程序，将虚拟混合接口发下来的 IP 分组接收下来，重新封装后使用 Wi-Fi 接口转发出去。

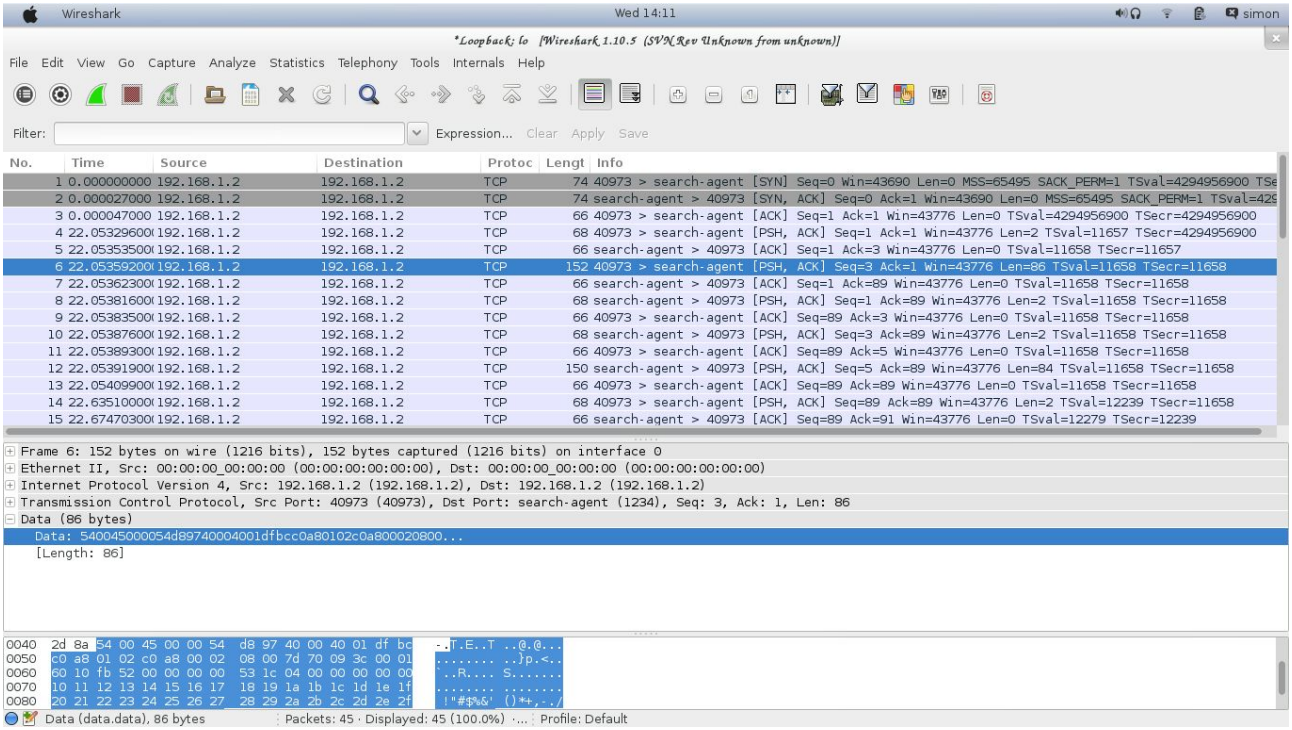
3.3 验证结果

（1）通过测试程序收发真实 Wi-Fi 网络接口上的 MAC 帧。从 Wireshark 上可以看到，该程序可以收到 Wireshark 能够看到的所有 MAC 帧，发出的 MAC 帧也能被 Wireshark 监听到。通过监听可以看出，本测试程序可以实现对指定实体网卡的 MAC 通信控制，可以将虚拟网络接口上收到的 IP 分组通过一个指定的网络接口转发出去。

（2）根据 Wireshark 监听的结果，可以看到测试程序能够通过真实网络接口收发 MAC 帧和 IP 分组，这说明 IP 层上的 TCP/IP 协议栈是可以使用的。然后，将关键点（2）程序接收到的 IP 分组直接发出，就实现了 IP 数据包的透明传输。

综上所述，可以确认，这种方式能够将关键点（2）中虚拟接口接收到 IP 分组进行封装，通过指定的真实 Wi-Fi 接口转发出去。本阶段完成了“Wi-Fi 接口转发虚拟接口发送的 IP 分组”的测试，达到了预期目标。

可以看到原来的 IP 数据包被封装在了 socket 中，前面有了系统添加的 IP 和 MAC 首部



4. 接收端IP分组的接收与回应

4.1 实现平台

（1）硬件平台：

两台计算机，启动了 Wi-Fi 连接；

（2）软件平台：

Linux 操作系统；2.6.32 以上版本内核。

4.2 实现方法

用 C 语言编写了一　个接收端的程序，能够将 Wi-Fi 接口收到的 IP 分组接收下来，通过系统的协议栈获得数据，交给它的虚拟混合接口，并根据上层协议的要求返回分组给发送端。

4.3 验证结果

（1）编写了一个接收端程序，该程序定义为关键技术点（3）对应的服务器程序，它能够与客户端程序进行交互，在服务器端回应客户端发送的 ping 程序，向客户端返回 ICMP-echo 包。根据 Wireshark 监听的结果，可以看到 ICMP 报文的交互过程。

（2）客户端和服务器之间的通信是在 TCP 连接上通过 Socket 方式进行的，在 TCP 分段内封装虚拟接口转发过来的 IP 分组。为了能够在 TCP 分段内封装 IP 分组，我们拟定了一个简单的协议：

- 8bit 首部长
- 8bit 类型
- 16bit 数据包长

使用此协议来标识内部封装的 IP 分组，该分组可以在两端的 Wireshark 上监听到。通过这个程序，可以实现本机上 IP 数据包传送到远程机的任务。

综上所述，可以确认，这种方式能够将关键点（3）中虚拟接口通过物理接口重新封装的 IP 分组转发出去，通过 Wi-fi 转发到对应的服务器端。服务器端对分组外面的首部进行剥离，根据内部首部进行回应。本阶段完成了“接收端 IP 分组的接收与回应”的测试，达到了预期目标。

远程服务器端

```
 75  3 63 6f 6d  0  0  1  0  1
recv head: 4a04
recv 59 bytes:45  0  0 3b bd dc 40  0 40 11 76 79 c0 a8  1  2 3d b1  7  1 e2 5a
 0 35  0 27 f6 9d 7a 25  1  0  0  1  0  0  0  0  0  0  3 70 61 6e  5 62 61 69 64
 75  3 63 6f 6d  0  0  1  0  1
recv head: 4a04
recv 60 bytes:45  0  0 3c 43 55 40  0 40  6 fc c3 c0 a8  1  2 b4 95 84 63 b8 19
 0 50 be 1a 40 24  0  0  0  0 a0  2 72 10 e6 46  0  0  2  4  5 b4  4  2  8  a  0
 2d 3e 30  0  0  0  0  1  3  3  7
recv head: 4a04
recv 40 bytes:45  0  0 28 67 5e 40  0 40  6 6e 3f 64 40 36  c 4a 7d 80 69 af 5e
 1 bb 4b 39 c2 29 39 79  f 53 50 10  1 62 41 f7  0  0
recv head: 4a04
recv 60 bytes:45  0  0 3c  0  2 40  0 40  6 d5 66 64 40 36  c 4a 7d 80 8a b2 41
 1 bb 45 df b6 a3  0  0  0  0 a0  2 71 70 80 97  0  0  2  4  5 ac  4  2  8  a  0
 2d 40  0  0  0  0  0  1  3  3  7
recv head: 4a04
recv 40 bytes:45  0  0 28 ef 32 40  0 40  6 e6  c 64 40 36  c 4a 7d 80 c7 8f 9a
 0 50 b1 6c 35 5d c0 eb 61 1f 50 10  0 fc b0 88  0  0
recv head: 4a04
recv 59 bytes:45  0  0 3b bd dd 40  0 40 11 76 78 c0 a8  1  2 3d b1  7  1 e2 c7
 0 35  0 27 96 7c d9 d9  1  0  0  1  0  0  0  0  0  0  3 70 61 6e  5 62 61 69 64
 75  3 63 6f 6d  0  0  1  0  1
```

本地客户端

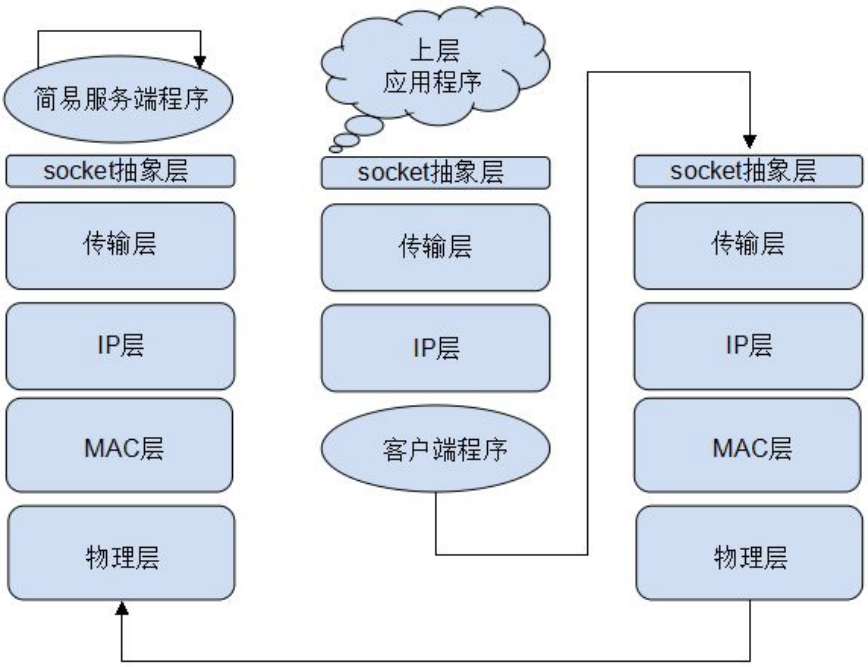
```
c1 ef aa e6 e4 75 f6 20 50 10 0 f4 2e 8b 0 0
send 40 bytes:45 0 0 28 2b 90 40 0 40 6 3b fb 64 40 36 c b4 95 84 63 da 6a 0 50
8e 8e 36 e4 a0 ed 0 70 50 11 0 ec 9a 17 0 0
send 60 bytes:45 0 0 3c 39 63 40 0 40 6 9c 5 64 40 36 c 4a 7d 80 8a b2 42 1 bb a
8 34 74 1 0 0 0 0 a0 2 71 70 5c e3 0 0 2 4 5 ac 4 2 8 a 0 2d 44 0 0 0 0 0 1 3 3
7
send 59 bytes:45 0 0 3b 29 9c 40 0 40 11 72 85 c0 a8 1 2 dd e4 ff 1 8a 59 0 35 0
 27 b6 6a 7a 25 1 0 0 1 0 0 0 0 0 0 3 70 61 6e 5 62 61 69 64 75 3 63 6f 6d 0 0 1
 0 1
send 59 bytes:45 0 0 3b 29 9d 40 0 40 11 72 84 c0 a8 1 2 dd e4 ff 1 a4 10 0 35 0
 27 3c ff d9 d9 1 0 0 1 0 0 0 0 0 0 3 70 61 6e 5 62 61 69 64 75 3 63 6f 6d 0 0 1
 0 1
send 59 bytes:45 0 0 3b bd de 40 0 40 11 76 77 c0 a8 1 2 3d b1 7 1 e2 5a 0 35 0
27 f6 9d 7a 25 1 0 0 1 0 0 0 0 0 0 3 70 61 6e 5 62 61 69 64 75 3 63 6f 6d 0 0 1
0 1
send 52 bytes:45 0 0 34 5 92 40 0 40 6 cf eb 64 40 36 c 4a 7d 80 7d df e9 14 66
6b 9c fd 71 98 de 1b f5 80 10 1 60 8a 37 0 0 1 1 8 a 0 2d 5b 59 a9 e2 6e 44
send 59 bytes:45 0 0 3b bd df 40 0 40 11 76 76 c0 a8 1 2 3d b1 7 1 d0 c5 0 35 0
27 71 c0 10 98 1 0 0 1 0 0 0 0 0 0 3 70 61 6e 5 62 61 69 64 75 3 63 6f 6d 0 0 1
0 1
send 59 bytes:45 0 0 3b bd e0 40 0 40 11 76 75 c0 a8 1 2 3d b1 7 1 cf dd 0 35 0
27 49 be 39 82 1 0 0 1 0 0 0 0 0 0 3 70 61 6e 5 62 61 69 64 75 3 63 6f 6d 0 0 1
0 1
```

Ping 程序

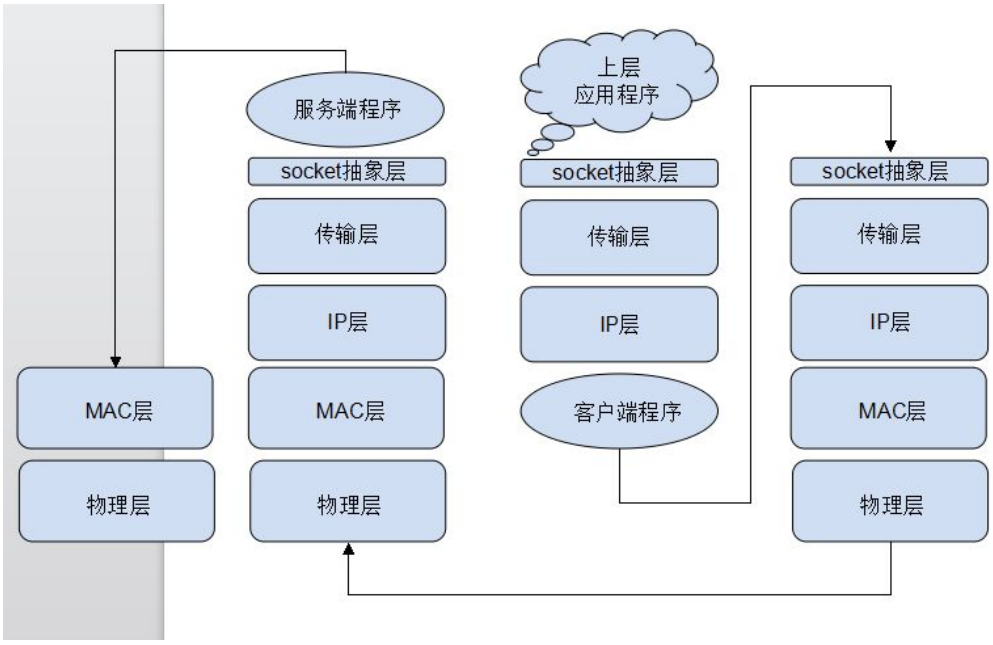
```
→  gemini git:(master) ✗ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         0.0.0.0         0.0.0.0         U     0      0        0 ppp0
100.64.0.1      0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
192.168.1.0     0.0.0.0         255.255.255.0   U     9      0        0 wlp3s0
→  gemini git:(master) ✗ sudo route del -net 0.0.0.0 netmask 0.0.0.0 dev ppp0
[sudo] password for simon:
→  gemini git:(master) ✗ sudo route add -net 0.0.0.0 netmask 0.0.0.0 dev tun0
→  gemini git:(master) ✗ ping 192.168.1.2 -c 1
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.093 ms

--- 192.168.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.093/0.093/0.093/0.000 ms
→  gemini git:(master) ✗
```

当前的测试系统原理图



计划的系统原理图



Acronyms

Reference

Revision History

Author	Date	Version	Comments
李领治	2013/01/06	1.0	Init Version