

琴剑飘零

博客园 首页 新随笔 联系 订阅 管理

【Atheros】Ath9k速率调整算法源码走读

上一篇文章介绍了驱动中minstrel_ht速率调整算法，atheros中提供了可选的两种速率调整算法，分别是ath9k和minstrel，这两个算法分别位于：

```
drivers\net\wireless\ath\ath9k\rc.c.....Ath9k
net\mac80211\minstrel_ht.c.....Minstrel
```

无论从理论分析还是实验结果上看，minstrel都要胜ath9k一筹，为了一个完整性，这里也把ath9k算法介绍一下，相比较于minstrel的随机探测，ath9k是按照特定顺序来探测的，不过这个顺序的排列却有问题。

1. 速率表

ath9k根据当前的标准是802.11a还是b/g/n，使用不同的速率表，这个速率表是硬编码的，首先来看存储这个速率表的结构体：

```
struct ath_rate_table {
    int rate_cnt;
    int mcs_start;
    struct {
        u16 rate_flags;
        u8 phy;
        u32 ratekbps;
        u32 user_ratekbps;
        u8 ratecode;
        u8 dot11rate;
        u8 ctrl_rate;
        u8 cw40index;
        u8 sgi_index;
        u8 ht_index;
    } info[RATE_TABLE_SIZE];
    u32 probe_interval;
    u8 initial_ratemax;
};
```

因为我的实验环境用的是802.11n，所以使用的速率表是11na的：

```
static const struct ath_rate_table ar5416_11na_ratetable = {
    68,
    8, /* MCS start */
    {
        [0] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 6000,
            5400, 0, 12, 0, 0, 0, 0 }, /* 6 Mb */
        [1] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 9000,
            7800, 1, 18, 0, 1, 1, 1 }, /* 9 Mb */
        [2] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 12000,
            10000, 2, 24, 2, 2, 2, 2 }, /* 12 Mb */
        [3] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 18000,
            13900, 3, 36, 2, 3, 3, 3 }, /* 18 Mb */
        [4] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 24000,
            17300, 4, 48, 4, 4, 4, 4 }, /* 24 Mb */
        [5] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 36000,
            23000, 5, 72, 4, 5, 5, 5 }, /* 36 Mb */
    }
};
```

公告

昵称：琴剑飘零
园龄：6年
粉丝：68
关注：12
+加关注

2017年9月						
日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

随笔分类

- Android(6)
- JavaSE/EE(11)
- Web技术(3)
- Web前端(6)
- 编程语言(2)
- 设备驱动(10)
- 设计模式(2)

随笔档案

- 2015年8月 (1)
- 2015年7月 (1)
- 2015年3月 (2)
- 2015年1月 (3)
- 2014年11月 (18)
- 2014年10月 (1)
- 2014年9月 (1)
- 2014年5月 (1)
- 2014年4月 (1)
- 2014年2月 (1)
- 2014年1月 (2)

```
[6] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 48000,
        27400, 6, 96, 4, 6, 6, 6 }, /* 48 Mb */
[7] = { RC_L_SDT, WLAN_RC_PHY_OFDM, 54000,
        29300, 7, 108, 4, 7, 7, 7 }, /* 54 Mb */
[8] = { RC_HT_SDT_2040, WLAN_RC_PHY_HT_20_SS, 6500,
        6400, 0, 0, 0, 38, 8, 38 }, /* 6.5 Mb */
.....
[15] = { RC_HT_S_20, WLAN_RC_PHY_HT_20_SS, 65000,
        59000, 7, 7, 4, 45, 16, 46 }, /* 65 Mb */
[16] = { RC_HT_S_20, WLAN_RC_PHY_HT_20_SS_HGI, 72200,
        65400, 7, 7, 4, 45, 16, 46 }, /* 75 Mb */
.....
[38] = { RC_HT_SDT_40, WLAN_RC_PHY_HT_40_SS, 13500,
        13200, 0, 0, 0, 38, 38, 38 }, /* 13.5 Mb */
.....
[45] = { RC_HT_S_40, WLAN_RC_PHY_HT_40_SS, 135000,
        112000, 7, 7, 4, 45, 46, 46 }, /* 135 Mb */
[46] = { RC_HT_S_40, WLAN_RC_PHY_HT_40_SS_HGI, 150000,
        122000, 7, 7, 4, 45, 46, 46 }, /* 150 Mb */
.....
[67] = { RC_HT_T_40, WLAN_RC_PHY_HT_40_TS_HGI, 450000,
        346400, 23, 23, 4, 66, 67, 67 }, /* 450 Mb */
},
50, /* probe interval */
WLAN_RC_HT_FLAG, /* Phy rates allowed initially */
};
```

这个变量的定义和前面的结构体——对应，首先是rate_cnt，是68，即有68个速率，之后是mcs_start，也就是从第几个速率开始是MCS的速率，对速率比较熟悉的能够看出来，0-7分别是11b/g的8个传统速率，最大54Mbps，从第8个开始，就是MCS速率，以[8]为例，结合前面的结构体看各个域的含义：

rate_flags	phy	ratekps	user_ratekps	ratecode	dot11rate	ctrl_rate	cw40index	sgi_index	ht_index
RC_HT_SDT_2040	WLAN_RC_PHY_HT_20_SS	6500	6400	0	0	0	38	8	38

这里面字段比较多，不过很多我都没有深究，一知半解，rate_flags的含义结合rc.h中的宏定义更清晰一些，RC_HT_S（单流）D（双流）T（三流）_2040（20/40M带宽），物理意义我不是很清楚，可能是说这个参数可以在哪些配置下用吧，比如说在配置的40MHz带宽下，是可以使用20MHz的MCS0的，但是配置的20MHz带宽就不能用40Mhz的MCS0发送。phy是当前速率所对应的物理层参数，这个就是20MHz的SS（单流），数据率是6.5Mbps，user_ratekps我没有仔细看过，可能是对去掉MAC和PHY头部之后对速率的估算吧，ratecode和dot11rate说的基本是一个事，就是说这是哪个MCS，这里就是MCS0，ctrl_rate是说，在11b/g的速率里面，哪一个可以作为当前速率的保底速率，最后三个就是说，当前的速率，也就是20MHz的MCS0，在40MHz下对应速率表里的第几个速率、SGI的情况下对应于哪一个，40MHz+SGI的情况下又对应于第几个速率。

介绍完速率表，剩下的，就按照和minstrel同样的思路来分析，先来看注册rate_control_ops的结构体：

```
static struct rate_control_ops ath_rate_ops = {
    .module = NULL,
    .name = "ath9k_rate_control",
    .tx_status = ath_tx_status,
    .get_rate = ath_get_rate,
    .rate_init = ath_rate_init,
    .rate_update = ath_rate_update,
    .alloc = ath_rate_alloc,
    .free = ath_rate_free,
    .alloc_sta = ath_rate_alloc_sta,
    .free_sta = ath_rate_free_sta,
#ifdef CONFIG_ATH9K_DEBUGFS
    .add_sta_debugfs = ath_rate_add_sta_debugfs,
#endif
};
```

- 2013年11月 (2)
- 2012年10月 (2)
- 2012年2月 (1)
- 2011年10月 (1)
- 2011年9月 (3)

最新评论

- 1. Re: 【JavaEE】Springmvc+Spring+Hibernate整合及example
严重: Exception sending context initialize event to listener instance of class org.springframework.we.....
--qidianxianxia
- 2. Re: 【JavaEE】SSH+Spring Security+Spring oauth2整合及example
@奉先 搞定了吗？能发一个例子让看看吗？...
--星辰海
- 3. Re: 【python】获取高德地图省市区县列表
你好，请问你这种方法能爬到路名吗？
--vvvvvww
- 4. Re: 【JavaEE】SSH+Spring Security+Spring oauth2整合及example
搞定了,谢谢你的文章
--奉先
- 5. Re: 【JavaEE】SSH+Spring Security+Spring oauth2整合及example
你好,我集成了您的ssh简易版oauth,项目名是demo4ssh-security-oauth的项目,security的配置几乎一致,前几步都可以走的通,但在用code换取access-token的.....
--奉先

阅读排行榜

- 1. 【JavaEE】SSH+Spring Security+Spring oauth2整合及example(9146)
- 2. 【JavaEE】Springmvc+Spring整合及example(4143)
- 3. 【JavaEE】Springmvc+Spring+Hibernate整合及example(3618)
- 4. 【JWPlayer】官方JWPlayer去水印步骤(3607)
- 5. 【Android】开源项目UI控件分类汇总之ProgressBar(3565)

评论排行榜

- 1. 【Atheros】Ath9k速率调整算法源码走读(25)
- 2. 【JavaEE】SSH+Spring Security+Spring oauth2整合及example(21)
- 3. 【Atheros】如何在驱动中禁用ACK(14)
- 4. 【Atheros】minstrel速率调整算法源码走读(12)
- 5. 【设计模式】简单工厂-工厂方法-抽象工厂(11)

推荐排行榜

- 1. 【Android】百度地图自定义弹出窗口(5)
- 2. 【百度地图】显示从某站点出发的所有公交线路(5)
- 3. 【JavaEE】SSH+Spring Security+Spring oauth2整合及example(5)



下面依次来看ath_rate_init、ath_get_rate、ath_tx_status。

2. 算法的初始化



```
for (i = 0; i < sband->n_bitrates; i++) {
    if (sta->supp_rates[sband->band] & BIT(i)) {
        ath_rc_priv->neg_rates.rs_rates[j]
            = (sband->bitrates[i].bitrate * 2) / 10;
        j++;
    }
}
ath_rc_priv->neg_rates.rs_nrates = j;

if (sta->ht_cap.ht_supported) {
    for (i = 0, j = 0; i < 77; i++) {
        if (sta->ht_cap.mcs.rx_mask[i/8] & (1<<(i%8)))
            ath_rc_priv->neg_ht_rates.rs_rates[j++] = i;
        if (j == ATH_RATE_MAX)
            break;
    }
    ath_rc_priv->neg_ht_rates.rs_nrates = j;
}
```



这两个循环的代码不用深究，n_bitrates一般是8，第一个循环就是看看11b/g的那8个速率哪一个被当前的驱动和硬件支持，第二个循环就是看看MCS的速率又有哪些被支持，把能支持的加入到ath_rc_priv->neg_rates和ath_rc_priv->neg_ht_rates中，neg表示Negotiated。

下面主要做了三件事：1.判断当前的配置是20MHz还是40MHz，2.当前配置是不是SGI，3.根据配置选择速率表，是前面介绍的ar5416_11na_ratetable，还是ar5416_11ng_ratetable、ar5416_11a_ratetable、ar5416_11g_ratetable。

最后调用了ath_rc_init函数，这个函数对速率调整各个速率的基本参数进行初始化：

```
for (i = 0; i < ath_rc_priv->rate_table_size; i++) {
    ath_rc_priv->per[i] = 0;
}
```

有这么一个叫per的数组，per是packet error rate的缩写，数组的每一项对应于刚才的那个速率表，这里先对所有的速率的per初始化为0。之后初始化两个变量：

```
for (i = 0; i < WLAN_RC_PHY_MAX; i++) {
    for (j = 0; j < MAX_TX_RATE_PHY; j++)
        ath_rc_priv->valid_phy_rateidx[i][j] = 0;
    ath_rc_priv->valid_phy_ratecnt[i] = 0;
}
```

打眼一看这两个变量不是很好理解，但是只要一看WLAN_RC_PHY_MAX是什么，就比较明朗了：



```
enum {
    WLAN_RC_PHY_OFDM,
    WLAN_RC_PHY_CCK,
    WLAN_RC_PHY_HT_20_SS,
    WLAN_RC_PHY_HT_20_DS,
    WLAN_RC_PHY_HT_20_TS,
    WLAN_RC_PHY_HT_40_SS,
    WLAN_RC_PHY_HT_40_DS,
    WLAN_RC_PHY_HT_40_TS,
    WLAN_RC_PHY_HT_20_SS_HGI,
    WLAN_RC_PHY_HT_20_DS_HGI,
    WLAN_RC_PHY_HT_20_TS_HGI,
    WLAN_RC_PHY_HT_40_SS_HGI,
    WLAN_RC_PHY_HT_40_DS_HGI,
}
```

4. 【JWPlayer】官方JWPlayer去水印步骤(3)

5. 【python】获取高德地图省市区县列表(2)

```

WLAN_RC_PHY_HT_40_TS_HGI,
WLAN_RC_PHY_MAX
};

```



前面的那个速率表如果看熟了，这些东西就会变得非常眼熟，正是每个速率的第二个域（也就是前面说明速率表结构体的那个表格的第二列），phy。换句话说，ath9k在这个时候也是给这些速率分了个组，分组的依据就是每个速率的phy域，valid_phy_ratecnt是存储这个组里有几个速率，valid_phy_rateidx就是存储各个组里都有哪些速率了，后面的一段代码就不粘帖了，就是把速率表里的速率遍历一下，把被支持的速率挑出来，给这两个数组赋上正确的值。这时候valid_phy_rateidx就是一个存储被支持的速率的二维数组了，这里要是不能理解也没有关系，因为这个函数之后基本就用不到这两个变量了，下一步，把这个二维数组压成一个一维数组，存储在valid_rate_index里面，只要明白valid_rate_index是什么意思就够了，结合一下代码看看把二维数组压成一维数组是什么意思（看代码应该比较清晰）：

```

for (i = 0, k = 0; i < WLAN_RC_PHY_MAX; i++) {
    for (j = 0; j < ath_rc_priv->valid_phy_ratecnt[i]; j++) {
        ath_rc_priv->valid_rate_index[k++] =
            ath_rc_priv->valid_phy_rateidx[i][j];
    }
    .....
}

```



然后，按照速率值的大小，也就是那个多少Mbps，对这些速率排序（也就是对ath_rc_priv->valid_rate_index这个数组排序）：

```

ath_rc_sort_valirates(rate_table, ath_rc_priv);

```

这个数组，就决定了以后的探测顺序，Ath9k的探测就是沿着这个方向，碰到投递率低的速率就停止探测，但是因为有带宽和空间流的影响，120Mbps的速率投递率是10%，150Mbps的投递率就一定很低吗？这是不一定的。

2. 获取发送速率

一开始，和minstrel一样，当目标站点不存在，或者本次发送不需要等ACK的时候，为了确保数据包尽可能被对方正确接收，那么会直接用传统速率来发送，不给它分配MCS速率：

```

if (rate_control_send_low(sta, priv_sta, txrc))
    return;

```

接下来调用ath_rc_get_highest_rlx函数，计算当前哪个速率是最好的速率：

```

maxindex = ath_rc_priv->max_valid_rate-1;
minindex = 0;
best_rate = minindex;

for (index = maxindex; index >= minindex ; index--) {
    u8 per_thres;

    rate = ath_rc_priv->valid_rate_index[index];
    .....
    per_thres = ath_rc_priv->per[rate];
    if (per_thres < 12)
        per_thres = 12;

    this_thruput = rate_table->info[rate].user_ratekbps * (100 - per_thres);

    if (best_thruput <= this_thruput) {
        best_thruput = this_thruput;
        best_rate = rate;
    }
}

```



从maxindex开始往minindex遍历，是在信道状况好的情况下，可以减少内存计算开销，当速率的per小于12%的时候，就按12%来对待，这个在源码里有解释：*For TCP the average collision rate is around 11%, so we ignore PERs less than this. This is to prevent the rate we are currently using (whose PER might be in the 10-15 range because of TCP collisions) looking worse than the next lower rate whose PER has decayed close to 0. If we used to next lower rate, its PER would grow to 10-15 and we would be worse off then staying at the current rate.*

调用ath_rc_get_highest_rix找到最好的速率之后存到变量rix中，rix是前面讲过的11na速率表中的速率索引。还有一个比较关键的函数是ath_rc_get_lower_rix，这个函数是要获取比rix稍差的速率，获取思路很简单，还记得前面的valid_rate_index吗，这个数组里存储着按比特率排序的各个速率，次佳速率就是这个数组中某速率的前一个（这个取法当然是不怎么科学的）。算法维护了一个当前最大速率的序号，如果新获得的这个highest_rix比这个数大，并且离上次探测时间超过一定阈值，就会启动探测，这个最后会讲。



```
if (is_probe) {
    ath_rc_rate_set_series(rate_table, &rates[i++], txrc, 1, rix, 0);
    ath_rc_get_lower_rix(rate_table, ath_rc_priv, rix, &rix);
    ath_rc_rate_set_series(rate_table, &rates[i++], txrc, try_per_rate, rix, 0);

    tx_info->flags |= IEEE80211_TX_CTL_RATE_CTRL_PROBE;
} else {
    ath_rc_rate_set_series(rate_table, &rates[i++], txrc, try_per_rate, rix, 0);
}

for (; i < 3; i++) {
    ath_rc_get_lower_rix(rate_table, ath_rc_priv, rix, &rix);
    ath_rc_rate_set_series(rate_table, &rates[i], txrc, try_per_rate, rix, 1);
}

try_per_rate = 8;

if ((rates[2].flags & IEEE80211_TX_RC_MCS) &&
    (!(tx_info->flags & IEEE80211_TX_CTL_AMPDU) ||
     (ath_rc_priv->per[high_rix] > 45)))
    rix = ath_rc_get_highest_rix(sc, ath_rc_priv, rate_table, &is_probe, true);
else
    ath_rc_get_lower_rix(rate_table, ath_rc_priv, rix, &rix);

ath_rc_rate_set_series(rate_table, &rates[i], txrc, try_per_rate, rix, 1);
```



和minstrel类似，关键函数长得也很像，标红的三个参数，分别是底层发包需要的速率结构体、最大发送次数和速率编号，ath9k是用4个速率来发送，对应的参数要赋到rates[0]、rates[1]、rates[2]和rates[3]这四个变量上，try_per_rate初始为4，这段代码的含义就是：如果需要探测，则rates[0]用探测速率发1次，如果失败，则换rates[1]用探测速率的次佳速率发4次，再失败就换rates[2]用rates[1]的次佳速率再发4次。如果不是探测，就按当前最佳速率、次佳速率、次次佳速率的顺序各尝试4次。最后还剩一个rates[3]，如果rates[0]的per超过45%，重新获取最高速率作为rates[3]的发送速率，如果rates[0]的per不到45%，就用rates[2]的次佳速率发送，最大尝试8次。在2.4GHz环境下或者数据包分片的情况下还会再微调，此处不再分析。

3. 发送完成后更新速率状态

和minstrel一样，聚合帧发送完之后，每一个子帧都会调用速率调整算法的tx_status函数，但是每个聚合帧里只有一个帧是携带了有用信息的，其他帧直接返回不予处理：

```
if ((tx_info->flags & IEEE80211_TX_CTL_AMPDU) &&
    !(tx_info->flags & IEEE80211_TX_STAT_AMPDU))
    return;
```

对于携带了发送状况信息的帧，驱动用了一个叫做xretries的变量区分这4个速率的状态，现在考虑两个例子：

例1：rates[0]发送4次都失败，rates[1]发送4次也都失败，rates[2]发送第3次成功，那么使用了的速率就是3个，rates[3]没有用到；

例2：rates[0]-rates[3]这总共20次发送全部失败。

xretries	含义
0	数据帧最终被成功发送，并且是使用的本速率，对上面的例子来说，例1的rates[2]的xretries就是0
1	数据帧最终没有成功发送，丢了，例2的4个速率的xretries都是1
2	数据帧最终被成功发送，但不是使用的本速率，例1的rates[0]和rates[1]的xretries是2

针对每个速率，首先调用ath_rc_update_per来更新per：

```
if (xretries == 1) {
    ath_rc_priv->per[tx_rate] += 30;
    if (ath_rc_priv->per[tx_rate] > 100)
        ath_rc_priv->per[tx_rate] = 100;
}
```

如果是xretries=1这种情况，per直接加30%。

```
else {
    /* xretries == 2 */ /* new_PER = 7/8*old_PER + 1/8*(currentPER) */
    ath_rc_priv->per[tx_rate] =
        (u8)(last_per - (last_per >> 3) + (100 >> 3));
}
```

新的per是旧per的7/8加上本次per的1/8，因为xretries=2对应的是发送全都失败的速率，所以本次per就是100%。

最后也是最复杂的就是最后成功的那个速率，这里需要先介绍几个基本参数，从头来看一下这个函数的参数和一个写死的lookup数组：

```
static void ath_rc_update_per(struct ath_softc *sc, const struct ath_rate_table
*rate_table,
                             struct ath_rate_priv *ath_rc_priv, struct ieee80211_tx_info *tx_info,
                             int tx_rate, int xretries, int retries, u32 now_msec)
{
    int count, n_bad_frames;
    u8 last_per;
    static const u32 nretry_to_per_lookup[10] = {
        100 * 0 / 1,
        100 * 1 / 4,
        100 * 1 / 2,
        100 * 3 / 4,
        100 * 4 / 5,
        100 * 5 / 6,
        100 * 6 / 7,
        100 * 7 / 8,
        100 * 8 / 9,
        100 * 9 / 10
    };
}
```

tx_rate不用说就能猜到，表示当前速率，xretries前面已经介绍，现在介绍的这种情况xretries=0。retries是说这个速率重传了多少次，比如rates[0]发了4次都失败了，就是重传了3次，rates[2]第3次成功，就是retries=2。

下面看逻辑：

```
if (n_bad_frames) {
    if (tx_info->status.ampdu_len > 0) {
        int n_frames, n_bad_tries;
        u8 cur_per, new_per;

        n_bad_tries = retries * tx_info->status.ampdu_len + n_bad_frames;
        n_frames = tx_info->status.ampdu_len * (retries + 1);
        cur_per = (100 * n_bad_tries / n_frames) >> 3;
    }
}
```

```

    new_per = (u8)(last_per - (last_per >> 3) + cur_per);
    ath_rc_priv->per[tx_rate] = new_per;
}
} else {
    ath_rc_priv->per[tx_rate] =
        (u8)(last_per - (last_per >> 3) + (nretry_to_per_lookup[retries] >> 3));
}

```

这里又分两种情况，前面说的成功发送其实是指聚合帧被成功发送，成功与否的标志是收到了块确认，也就是说有可能有子帧因为CRC等错误还是没有被正确接受，这样出错的帧的数量就是n_bad_frames，更新PER的算法都是一样的： $new_per = old_per * 7/8 + cur_per / 8$ ，但是cur_per的算法不固定，如果n_bad_frames是0，那不按照正统方法来算，还是假设第3次发送成功，也就是重传了两次，那么this_per不是常规认为的66%，而是查表得来的50%。如果n_bad_frames不是0，那么就是正规方法了，前面retries次一共丢了多少加上最后一次丢了几个，除以总的发送子帧数就是cur_per。

最后，如果成功发送的速率是探测速率，如果它的PER小于50%且大于30%的话，置成20%，并且，因为这次探测的结果还不错，所以下次探测的时间就会在半个探测间隔（ $rate_table->probe_interval / 2$ ）之后就

```

if (ath_rc_priv->probe_rate && ath_rc_priv->probe_rate == tx_rate) {
    if (retries > 0 || 2 * n_bad_frames > tx_info->status.ampdu_len) {
        .....
    } else {
        .....
        if (ath_rc_priv->per[probe_rate] > 30)
            ath_rc_priv->per[probe_rate] = 20;
        .....
        ath_rc_priv->probe_time = now_msec - rate_table->probe_interval / 2;
    }
}

```

后面还有一些小判断，如果当前速率的PER已经超过55%，而且这是一个比当前最高速率更小的速率，按照ath9k算法的理念，它按比特率排序的结果就是越靠前的速率越稳定，如果当前速率都已经不行了，那么比它更靠后的最大速率肯定更不行了，这时候就降低rate_max_phy的值：

```

if (ath_rc_priv->per[tx_rate] >= 55 && tx_rate > 0 &&
    rate_table->info[tx_rate].ratekbps <=
    rate_table->info[ath_rc_priv->rate_max_phy].ratekbps) {
    ath_rc_get_lower_rix(rate_table, ath_rc_priv, (u8)tx_rate, &ath_rc_priv->rate_max_phy);

    /* Don't probe for a little while. */
    ath_rc_priv->probe_time = now_msec;
}

```

ath9k是以排序越靠前的速率越稳定为前提的，那么如果这个单调性不存在了怎么办？ath9k的做法是强制让它单调，如果前面的速率的PER比后面的大，就赋值成和后面一样的：

```

if (ath_rc_priv->per[tx_rate] < last_per) {
    for (rate = tx_rate - 1; rate >= 0; rate--) {
        if (ath_rc_priv->per[rate] > ath_rc_priv->per[rate+1]) {
            ath_rc_priv->per[rate] =
                ath_rc_priv->per[rate+1];
        }
    }
}

```


同样，从当前速率越往后就需要越不稳定，也需要强制规范一下，这个代码就不贴了。

最后，为了不让PER升的太高，每隔一段时间就会降为原来的7/8：

```
if (now_msec - ath_rc_priv->per_down_time >= rate_table->probe_interval) {
    for (rate = 0; rate < size; rate++) {
        ath_rc_priv->per[rate] = 7 * ath_rc_priv->per[rate] / 8;
    }
    ath_rc_priv->per_down_time = now_msec;
}
```

4. 探测频率

最后一个问题，ath9k什么时候开始探测，本文开头介绍的rate_table里有一个域是probe_interval，11na的速率表设定的值是50，再加上ath_rc_priv->probe_time存储上次探测的时间，根据这两个变量，在时间上控制探测的间隔。除此之外还有一个非常重要的变量，沿着这个变量的赋值搜索下去，就能弄清楚探测的原理，这个变量就是ath_rc_priv->rate_max_phy，前面反复提到，ath9k对所有的速率排了个序，它认为，这些速率的表现是单调变化的，如果排序之后的速率n已经不行了，那么n+1、n+2肯定都已经不行了，这个rate_max_phy就是当前性能不错的最大速率的序号，下面就沿着这个这个变量分析：

在ath_rc_init函数中完成对所有速率的排序之后，将排序后的倒数第三个速率设置为rate_max_phy：

```
ath_rc_priv->rate_max_phy = ath_rc_priv->valid_rate_index[k-4];
```

此时k=valid_rate_index.length，为什么是取倒数第三个作为rate_max_phy，不了解。

前面介绍，在发送数据包时，会寻找当前吞吐率最高的速率，找到之后会进行下面的判断：

```
if (rate >= ath_rc_priv->rate_max_phy) {
    rate = ath_rc_priv->rate_max_phy;

    /* Probe the next allowed phy state */
    if (ath_rc_get_nextvalid_txrate(rate_table, ath_rc_priv, rate, &next_rate) &&
        (now_msec - ath_rc_priv->probe_time > rate_table->probe_interval) &&
        (ath_rc_priv->hw_maxretry_pktcnt >= 1)) {
        rate = next_rate;
        ath_rc_priv->probe_rate = rate;
        ath_rc_priv->probe_time = now_msec;
        ath_rc_priv->hw_maxretry_pktcnt = 0;
        *is_probing = 1;
    }
}
```

rate就是找到的最大速率，仔细看代码会发现，这个rate其实一定会是一个小于等于rate_max_phy的数，因为算法自动忽略了大于rate_max_phy的速率，不过没有关系，对这段代码的解释可以是这样：当找到的最佳速率不比之前设定的最大速率小的时候，说明这个时候可以往高速率上探测一下了，于是呢

get_nextvalid_txrate获取rate之后更高的速率，并且判断一下是不是已经到了该探测的时间了，如果是，则使用这个更高速率进行探测。最后，这个rate_max_phy是怎么改变的呢，当探测帧的投递率大于50%的时候，会把刚刚探测的速率作为新的rate_max_phy：

```
if (ath_rc_priv->probe_rate && ath_rc_priv->probe_rate == tx_rate) {
    if (retries > 0 || 2 * n_bad_frames > tx_info->status.ampdu_len) {
        ath_rc_priv->probe_rate = 0;
    } else {
        u8 probe_rate = 0;
        ath_rc_priv->rate_max_phy = ath_rc_priv->probe_rate;
        probe_rate = ath_rc_priv->probe_rate;
        .....
        ath_rc_priv->probe_rate = 0;
    }
}
```



```
/*
 * Since this probe succeeded, we allow the next probe twice as soon.
 * This allows the maxRate to move up faster if the probes are successful.
 */
ath_rc_priv->probe_time = now_msec - rate_table->probe_interval / 2;
}
}
```

至此，ath9k速率调整算法的基本流程就差不多了。

分类: [设备驱动](#)

好文要顶

关注我

收藏该文

琴剑飘零
关注 - 12
粉丝 - 68
[+加关注](#)

1

推荐

0

反对

« 上一篇: [【Atheros】minstrel速率调整算法源码走读](#)

» 下一篇: [【Atheros】pktgen的ipv6地址处理函数参考及ipv6基本知识](#)

posted @ 2014-11-12 21:02 琴剑飘零 阅读(1837) 评论(25) 编辑 收藏

评论列表

- # 1楼 2015-09-16 11:44 zyzferrari

“ 您好，我想关闭低于12Mb的报文，请问怎么实现，谢谢。

支持(0) 反对(0)
- # 2楼[楼主] 2015-09-17 08:51 琴剑飘零

“ @ zyzferrari
关闭低于12M的报文是什么意思，是说发送方不用低于12Mbps的速率发吗，如果是这样的话，首先得看自己用的速率表，是802.11na、ng的表还是传统速率表，最好的方法是在速率调整算法里面改，要想知道你用的什么速率调整算法，还有一种简单的方法不用管速率调整算法，直接在drivers/net/wireless/ath/ath9k/xmit.c的ath_buf_set_rate函数里改，这是速率调整算法已经作用之后了，如果速率调整算法调出来的最好速率比12M低，统统换成一个大于12M的速率，希望对您有帮助~

支持(0) 反对(0)
- # 3楼 2015-09-17 11:27 zyzferrari

“ 真是太感谢您的回复了，不过我是新手，按照你说的方案，在ath_buf_set_rate函数修改，但是我看了上午，看不出所以然来，还望大神多多指点，具体怎么改，改哪个变量。因为问题比较急，所以。。。。

支持(0) 反对(0)
- # 4楼[楼主] 2015-09-17 23:28 琴剑飘零

“ @ zyzferrari
驱动默认是用4个速率来尝试发送，这四个速率的设置是在set_rate函数：
for (i = 0; i < 4; i++) {
bool is_40, is_sgi, is_sp;
int phy;

if (!rates[i].count || (rates[i].idx < 0))
continue;

rix = rates[i].idx;
info->rates[i].Tries = rates[i].count;
这个循环里，rates[i]这个结构是速率调整算法的结果，现在要把这个结果赋值到tx_info里面去，所以就比较简单了，如果rates[i].idx代表的速率小于12M，就把rix设成一个大于12M速率的index，至于哪个速率是大于12M的，就要看当前使用的速率表了。

支持(0) 反对(0)
- # 5楼 2015-09-18 17:27 zyzferrari

“ 您好，我用了三种方法，都无效。
(1) 在ath_buf_set_rate函数中添加了如下代码，不生效；
if (!rates[i].count || (rates[i].idx < 0))
continue;

```
switch(sc->cur_rate_mode){
case ATH9K_MODE_11A:
if(rates[i] < 2)
rates[i].idx = 2;
break;
case ATH9K_MODE_11G:
if(rates[i] < 6)
rates[i].idx = 6;
break;
case ATH9K_MODE_11NA_HT20:
case ATH9K_MODE_11NA_HT40PLUS:
case ATH9K_MODE_11NA_HT40MINUS:
if(rates[i] < 2)
rates[i].idx = 2;
if(rates[i] == 8)
rates[i].idx = 9;
break;

case ATH9K_MODE_11NG_HT20:
case ATH9K_MODE_11NG_HT40PLUS:
case ATH9K_MODE_11NG_HT40MINUS:
if(rates[i] < 6)
rates[i].idx = 6;
if(rates[i] == 12)
rates[i].idx = 13;
break;
default:

}

rix = rates[i].idx;
```

- (2) 用速率表中12Mb的参数替换低于12Mb定义的参数，无效
- (3) 直接删除速率表中低于12Mb的定义，无效

支持(0) 反对(0)

#6楼[楼主] 2015-09-19 23:12 琴剑飘零

@ zyzferrari
第二种第三种方法是肯定无效的，因为驱动里的速率表只是一个在上层估算吞吐率的参考数据，底层到底用什么速率，只是通过上层给它的一个速率idx来选择的，所以改驱动里的速率表没有用。第一种方法应该可以啊，你是怎么测的速率啊？

支持(0) 反对(0)

#7楼 2015-09-21 09:04 zyzferrari

@ 琴剑飘零
我用OmniPeek和无线抓包网卡，手机连接wifi，ping路由器，慢慢远离路由器，直到信号比较弱的地方，然后看看抓包的情况，有一项就是DataRate，当信号强度为40%~70%的时候，速率都是低于6Mb的。

支持(0) 反对(0)

#8楼[楼主] 2015-09-22 17:52 琴剑飘零

@ zyzferrari
你在循环开始的时候加一句这个试试吧：rates[i].flags |= IEEE80211_TX_RC_MCS;当投速率下降之后速率调整算法有可能会切换到传统速率（非MCS）上去，用这个强制使用MCS。你说的6MB是用iw命令看到的吗？如果是个人统计的话，速率低有可能是因为竞争信道或者丢包造成吞吐率下降，物理层使用的发送速率有可能是高于12M的。

支持(0) 反对(0)

#9楼 2016-01-06 20:55 知否，知否

你好，我用PCAP的INJECT注入到数据链路层，利用监听模式发无线包，自己构造好了无线包，速率也在radiotap头部指定了。但是发送出去之后底层是按照1Mb发的。看了你的速率控制之后，有点了解。但不是很清楚，就是发送包的速率到底是由什么控制的？难道上层应用 不能控制速率吗？如果我要让底层按照我想要的速率发送出去，应该怎么做呢？可以提示一下吗？谢谢啦。

支持(0) 反对(0)

#10楼[楼主] 2016-01-07 00:02 琴剑飘零

@ 知否，知否
您好，我不清楚您说的无线包是在哪里构造，您说的上层是哪一层，怎么能在上层构造radiotap头呢？关于您的问题，我的理解是不能在上层指定底层的发送速率，网络协议栈本身在各层之间耦合比较低，多速率的指定在802.11的各个协议上各不相同，所以很难在高层提供这种指定速率的接口，所以我个人觉得只能在链路层控制，或者驱动提供的iw命令可能能够指定速率（这个我记不很清了，iw命令比较多）。

支持(0) 反对(0)

#11楼 2016-01-07 10:07 知否，知否

- “

@ 琴剑飘零

在应用层，利用PCAP中的 INJECT 这个函数提供了一个接口，直接通到mac80211这层。好像是必须要使用monitor模式，，所以构造包的时候必须把80211头部构造完之后 也要构造radiotap头部给网卡这种monitor模式看。但是 奇怪的是，包可以发送出去，但是速率 不是我想要的那个速率（这个速率是radiotap头部的那个速率我自己封装的多少就是多少）。所以我才想到，要改速率。如果我要在链路层控制速率的话。我该怎么做呢？

支持(0) 反对(0)
- # 12楼 2016-01-07 10:13 知否, 知否 ￼
- “

drivers\net\wireless\ath\ath9k\rc.c.....Ath9k
net\mac80211\minstrel_ht.c.....Minstrel

还有个问题，就是我这边那个compat驱动里面的没有rc.c这个文件。。这个文件在内核里面。看你写的这个意思 rc.c好像是在 驱动文件里面。

支持(0) 反对(0)
- # 13楼 2016-01-11 16:52 海贼王507 ￼
- “

@琴剑飘零

在无线路由器的嵌入式linux中，无线驱动包含两个：Compat-wireless(新版的改名为backport-wireless) 和linux-3.10.49/driver/net...(你博客上说的应该是这个)。我的问题是在嵌入式系统中，具体使用上面的哪一个？（另外在Compat-wireless的ath9k中没有rc.c，但是有net\mac80211\minstrel_ht.c），谢谢！

支持(0) 反对(0)
- # 14楼[楼主 ￼] 2016-01-12 14:32 琴剑飘零 ￼
- “

@ 知否, 知否

不好意思刚注意到，rc.c就是在compat-wireless/drivers里面的，最后编译完会在ath9k.ko里面

支持(0) 反对(0)
- # 15楼[楼主 ￼] 2016-01-12 14:34 琴剑飘零 ￼
- “

@ 海贼王507

是用的compat-wireless，我用的还是11年或者12年的一个版本，就后来好像也没有更新吧

支持(0) 反对(0)
- # 16楼 2016-01-12 15:40 知否, 知否 ￼
- “

@ 琴剑飘零

现在的compat-wireless里面没有rc.c这个文件。。而且我们编译给路由器的时候。内核里面的rc.c也没有rc.o文件。这是不是就意味着 我们这个路由器没有使用 ath9k这个 算法。。使用的是minstrel算法？minstrel的.o文件是有的。

支持(0) 反对(0)
- # 17楼[楼主 ￼] 2016-01-17 14:32 琴剑飘零 ￼
- “

@ 知否, 知否

那我不甚清楚了，你的驱动自带的补丁里面有没有一个117-remove_ath9k_rc.patch？它里面的内容是注释掉了config.mk里面的CONFIG_ATH9K_RATE_CONTROL=y，这是禁用Ath9k算法的方式。如果你那里没有这个补丁，可能就设这个算法吧

支持(0) 反对(0)
- # 18楼 2016-01-18 21:25 知否, 知否 ￼
- “

@ 琴剑飘零

谢谢。确实是这样，现在的网卡都不用ATH9K这种速率控制算法了。都用的是Minstrel这种算法。

支持(0) 反对(0)
- # 19楼 2016-04-21 09:50 piaoxuebing_feng ￼
- “

@ 琴剑飘零

楼主用的开发平台是openwrt的么，我想了解一下这两种算法是怎么切换的，是需要用哪些宏来控制么？

支持(0) 反对(0)
- # 20楼[楼主 ￼] 2016-04-21 12:47 琴剑飘零 ￼
- “

@ piaoxuebing_feng

是在openwrt上，在config.mk文件中有一行：CONFIG_ATH9K_RATE_CONTROL=y的配置，有这一行的话用ath9k，注释掉的话用minstrel，如果要加别的，我就不知道咋弄了，我自己做算法是在ath9k上改的

支持(0) 反对(0)
- # 21楼 2016-06-07 19:50 gechangwei ￼
- “

@ 琴剑飘零


请问这个config.mk文件在哪个目录？

支持(0) 反对(0)
- # 22楼 2016-08-03 09:38 61个夜 ￼
- “

你好楼主，如果我是想改变信道，为了通信质量，改到一个现在尚不支持的信道，应该是一个什么思路哇？

- #23楼[楼主] 2016-08-03 10:21 琴剑飘零 @ 61个夜
不支持的信道也没法工作吧？我没试过在代码里改信道，印象中iw命令可以设置信道，对应代码里应该也有，可以在hw等文件里搜一搜channel试试，切换的时机，可以参考重发（冲突）状况或者信噪比，谢谢
支持(0) 反对(0)
- #24楼 2016-08-03 16:53 61个夜 @ 琴剑飘零
是做一个局域网通信，不去管规定的信道，希望可以移动到一个新的频点工作，看了半天也没有看出个所以然来，可以参考信噪比是什么意思？
支持(0) 反对(0)
- #25楼[楼主] 2016-08-03 20:56 琴剑飘零 @ 61个夜
我本来以为是自适应切换到干扰最小的信道，标准自然就是丢包情况，还有信号噪声比了~
支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

[【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库](#)
[【推荐】腾讯云上实验室 1小时搭建人工智能应用](#)
[【推荐】可嵌入您系统的“在线Excel”！SpreadJS 纯前端表格控件](#)
[【推荐】阿里云“全民云计算”优惠升级](#)




美团云
GPU云主机 5折起
M60 原价1660/月 现价830/月 P40 原价3400/月 现价2100/月
查看详情

最新IT新闻：

- 谷歌新DOODLE纪念墨西哥国家歌舞团创始人阿玛莉亚
- 乐视游戏或被冰穹互娱收购 2016年净利润仅2万元
- Facebook AI研发主管：一味模仿人脑将阻碍AI的发展
- 50位演唱嘉宾助阵！魅蓝6演唱会又回来了
- 京东联合欧莱雅发布“包容美力计划” 助力残障群体就业

» 更多新闻...



JIGUANG | 极光
极光统计 多维洞察用户
增长运营指标

最新知识库文章：

- Google 及其云智慧
- 做到这一点，你也可以成为优秀的程序员
- 写给立志做码农的大学生
- 架构腐化之谜
- 学会思考，而不只是编程

» 更多知识库文章...