

琴剑飘零

博客园 首页 新随笔 联系 订阅 管理

【Atheros】minstrel速率调整算法源码走读

先说几个辅助的宏，因为内核不支持浮点运算，当然还有实现需要，minstrel对很多浮点值做了缩放：

```
/* scaled fraction values */
#define MINSTREL_SCALE 16
#define MINSTREL_FRAC(val, div) (((val) << MINSTREL_SCALE) / div)
#define MINSTREL_TRUNC(val) ((val) >> MINSTREL_SCALE)
```

MINSTREL_SCALE是一个放大的倍数，minstrel设定的是16，缩放16位也就是2^16倍，我不知道为什么要设置成这么大的数，不过没有关系不影响理解。MINSTREL_FRAC是两个数相除之后放大，MINSTREL_TRUNC则是逆运算，将某个被放大的数缩小。

下面进行源码分析，根据对外暴露的结构体对指针函数的定义：

```
static struct rate_control_ops mac80211_minstrel_ht = {
    .name = "minstrel_ht",
    .tx_status = minstrel_ht_tx_status,
    .get_rate = minstrel_ht_get_rate,
    .rate_init = minstrel_ht_rate_init,
    .rate_update = minstrel_ht_rate_update,
    .alloc_sta = minstrel_ht_alloc_sta,
    .free_sta = minstrel_ht_free_sta,
    .alloc = minstrel_ht_alloc,
    .free = minstrel_ht_free,
#ifdef CONFIG_MAC80211_DEBUGFS
    .add_sta_debugfs = minstrel_ht_add_sta_debugfs,
    .remove_sta_debugfs = minstrel_ht_remove_sta_debugfs,
#endif
};
```

我们需要着重关注核心的三个函数：tx_status负责每次发送完聚合帧之后根据ACK的状况更新各个速率状态，get_rate负责每次要发新的数据包的时候指定发送速率，rate_init在与另一站点建立连接的时候初始化相关参数。下面首先介绍一个抽随机速率用的表的生成，然后按照rate_init、get_rate、tx_status的顺序介绍minstrel的原理。

1. 初始化探测速率表

minstrel对速率的管理是通过速率组来管理的，这关乎几个重要的变量：

```
const struct mcs_group minstrel_mcs_groups[];

static u8 sample_table[SAMPLE_COLUMNS][MCS_GROUP_RATES];

mi->groups[]
```

先说minstrel_mcs_groups，我的实验环境最多支持双流，minstrel_mcs_groups也就是一个长度为8的数组，如果是三流就是长度为12的数组，这8个group的配置按顺序分别是：

组号	空间流数	是否支持SGI	20MHz/40MHz
0	1	否	20
1	2	是	20
2	1	否	20
3	2	是	20

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)

4	1	否	40
5	2	是	40
6	1	否	40
7	2	是	40

这个变量存的是这几个速率组不变的配置信息，mi->groups这个数组和minstrel_mcs_groups相对应，存储在对应配置下8个速率的动态数据统计。

sample_table是随机生成的一个速率表，本节就是讲这个表的生成和作用。

当Minstrel模块加载的时候，首先初始化速率表，也就是sample_table，当需要进行探测的时候，探测顺序就是以这个速率表做参考：

```
static u8 sample_table[SAMPLE_COLUMNS][MCS_GROUP_RATES];
static void
init_sample_table(void)
{
    int col, i, new_idx;
    u8 rnd[MCS_GROUP_RATES];

    memset(sample_table, 0xff, sizeof(sample_table));
    for (col = 0; col < SAMPLE_COLUMNS; col++) {
        for (i = 0; i < MCS_GROUP_RATES; i++) {
            get_random_bytes(rnd, sizeof(rnd));
            new_idx = (i + rnd[i]) % MCS_GROUP_RATES;

            while (sample_table[col][new_idx] != 0xff)
                new_idx = (new_idx + 1) % MCS_GROUP_RATES;

            sample_table[col][new_idx] = i;
        }
    }
}
```

SAMPLE_COLUMNS的默认取值是10，MCS_GROUP_RATES是每组MCS中有几个速率，也就是8（单流、双流、三流里面各有8个速率），这个速率取样表就是一个10*8的方阵。这段函数生成的速率表，一共10行，每一行都是0-7这七个数字的随机分布，minstrel是随机探测，但是用哪个速率不是在发送的时候才随机获取的，而是提前随机生成这个表，发送的时候依次遍历这个表，所以说这个速率表其实没有什么讲究，就是避免了在运行过程中频繁的抽随机数罢了，这个表已经提前生成了10组MCS随机排列的序列，运行过程中只要顺序读取，就是随机探测了。

2. 初始化探测的相关参数

minstrel_ht_rate_init和minstrel_ht_rate_update分别在连接站点初始化或者需要更新的时候被调用，他们的本质都是调用了minstrel_ht_update_caps，其中和速率调整相关的呢是这么几句：

```
mi->avg_ampdu_len = MINSTREL_FRAC(1, 1);

/* When using MRR, sample more on the first attempt, without delay */
if (mp->has_mrr) {
    mi->sample_count = 16;
    mi->sample_wait = 0;
} else {
    mi->sample_count = 8;
    mi->sample_wait = 8;
}
mi->sample_tries = 4;
.....
for (i = 0; i < ARRAY_SIZE(mi->groups); i++) {
    u16 req = 0;

    mi->groups[i].supported = 0;
```

- 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 d 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)

```

if (minstrel_mcs_groups[i].flags & IEEE80211_TX_RC_SHORT_GI) {
    if (minstrel_mcs_groups[i].flags & IEEE80211_TX_RC_40_MHZ_WIDTH)
        req |= IEEE80211_HT_CAP_SGI_40;
    else
        req |= IEEE80211_HT_CAP_SGI_20;
}

if (minstrel_mcs_groups[i].flags & IEEE80211_TX_RC_40_MHZ_WIDTH)
    req |= IEEE80211_HT_CAP_SUP_WIDTH_20_40;

if ((sta_cap & req) != req)
    continue;

mi->groups[i].supported =
    mcs->rx_mask[minstrel_mcs_groups[i].streams - 1];

if (mi->groups[i].supported)
    n_supported++;
}

if (!n_supported)
    goto use_legacy;

```



这一段的主要作用呢，就是初始化平均AMPDU长度（聚合帧长度）为1，因为我们需要MRR（Multi-rate retry，多速率重传），设置sample_count=16\sample_wait=0\sample_tries=4，这三个参数和探测的频率相关，后面会介绍，最后，确定本文最前面介绍的关于（空间流数、SGI、带宽）设置的数组中的每一数组项是不是被硬件支持。

3. 发送时确定发送速率

minstrel_ht_get_rate是速率调整最重要的两个函数之一，它决定了当前待发送数据包的发送速率。

```

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

```

首先，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。

下面获取本次取样的速率：

```

sample_idx = minstrel_get_sample_rate(mp, mi);

```

深入minstrel_get_sample_rate：

```

mg = &mi->groups[mi->sample_group];
sample_idx = sample_table[mg->column][mg->index];
mr = &mg->rates[sample_idx];
sample_idx += mi->sample_group * MCS_GROUP_RATES;
minstrel_next_sample_idx(mi);

```

前面已经介绍，有一个随机生成的速率表用作获取随机数用，首先在这个速率表中选取下一个sample_idx，之后，调用minstrel_next_sample_idx把mi->sample_group指向下一个可用的group，这个group就是前面介绍的拥有空间流数、SGI、带宽配置的组了。然后sample_idx在自身取值的基础上加上了sample_group*MCS_GROUP_RATES，所以这个sample_idx可以用来得到当前用的第几个取样组，比如sample_idx=27，那么就是8*3+4，它表示的就是第4个速率组（双流、SGI、20MHz）的第四个速率，因为是双流，就表示MCS11。正常情况下，minstrel_get_sample_rate这个函数会把sample_idx返回，如果不探测，则返回-1，下面继续看minstrel_get_sample_rate的几个返回-1的情况：

```

if (!mp->has_mrr && (mr->probability > MINSTREL_FRAC(95, 100)))
    return -1;

```

如果不支持mrr（多速率重传），并且当前速率的投递率已经达到了95%以上，就不再取样。



```

if (minstrel_get_duration(sample_idx) >
    minstrel_get_duration(mi->max_tp_rate)) {

```

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)

```

if (mr->sample_skipped < 20)
    return -1;

if (mi->sample_slow++ > 2)
    return -1;
}

```



这里把刚刚得到的这个要取样的速率sample_idx和当前吞吐率最高的速率进行比较，那么这个duration是个什么东西？这个duration就是对网卡用当前速率发送一个数据包所用时间的估算，duration越大，基本等效于速率值越低，具体含义如下：

```

#define AVG_PKT_SIZE    1200
#define MCS_NBITS  (AVG_PKT_SIZE << 3)
#define MCS_NSYMS(bps) ((MCS_NBITS + (bps) - 1) / (bps))

#define MCS_SYMBOL_TIME(sgi, syms) \
    (sgi ? \
        ((syms) * 18 + 4) / 5 : /* syms * 3.6 us */ \
        (syms) << 2 /* syms * 4 us */ \
    )

#define MCS_DURATION(streams, sgi, bps) MCS_SYMBOL_TIME(sgi, MCS_NSYMS((streams) * (bps)))

/*
 * Define group sort order: HT40 -> SGI -> #streams
 */
#define GROUP_IDX(_streams, _sgi, _ht40) \
    MINSTREL_MAX_STREAMS * 2 * _ht40 + \
    MINSTREL_MAX_STREAMS * _sgi + \
    _streams - 1

/* MCS rate information for an MCS group */
#define MCS_GROUP(_streams, _sgi, _ht40) \
    [GROUP_IDX(_streams, _sgi, _ht40)] = { \
        .streams = _streams, \
        .flags = \
            (_sgi ? IEEE80211_TX_RC_SHORT_GI : 0) | \
            (_ht40 ? IEEE80211_TX_RC_40_MHZ_WIDTH : 0), \
        .duration = { \
            MCS_DURATION(_streams, _sgi, _ht40 ? 54 : 26), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 108 : 52), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 162 : 78), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 216 : 104), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 324 : 156), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 432 : 208), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 486 : 234), \
            MCS_DURATION(_streams, _sgi, _ht40 ? 540 : 260) \
        } \
    }

```



这一段宏的定义比较复杂，简单来说，假定平均每个数据包长度是1200字节，也就是（1200<<3）即（1200*8）bits，根据每个码元（symbol）包含几个bit算出来这个数据包一共有多少码元（MCS_NSYMS），最后根据数据率计算数据包在每一个MCS参数下的发送延时。

花了这么多时间介绍minstrel_get_sample_rate，因为这个函数包括了最主要的，探测速率从哪儿来的问题，现在回到minstrel_ht_get_rate，刚才从是：

```
sample_idx = minstrel_get_sample_rate(mp, mi);
```

展开的，获取到sample_idx，后面就是重头戏了：



```

if (sample_idx >= 0) {
    sample = true;
    minstrel_ht_set_rate(mp, mi, &ar[0], sample_idx, true, false);
    info->flags |= IEEE80211_TX_CTL_RATE_CTRL_PROBE;
} else {
    minstrel_ht_set_rate(mp, mi, &ar[0], mi->max_tp_rate, false, false);
}

```



如果sample_idx为-1，那么就不探测，仍然用之前确定的最高吞吐率的速率mi->max_tp_rate来发送，除了前面说的几个情况下会返回-1之外，还有一个重要的控制，就是关于探测周期的控制，本文最后一部分会做介绍，下面继续看需要探测的情况下怎么选速率，minstrel_ht_set_rate这个函数的其他参数可以不关注，重点是我标红的这三个，第一个是一个（struct ieee80211_tx_rate）类型的参数，最后驱动底层就是从这个结构体里面拿到发送的MCS、重传次数、发送带宽等参数，minstrel用一个ar数组来存，ar[0]是第一个速率，重传多次仍然失败的话就用ar[1]，以此类推，第二个参数就是速率索引sample_idx/max_tp_rate，第三个参数代表当前帧是不是探测采样用的帧。对这个函数要重点说的，是重传策略：

```

if (sample)
    rate->count = 1;
else if (mr->probability < MINSTREL_FRAC(20, 100))
    rate->count = 2;
else if (rtscts)
    rate->count = mr->retry_count_rtscts;
else
    rate->count = mr->retry_count;

```



如果当前速率用作探测，则只发送一次，如果当前速率的投递率小于20%，则发两次，其他情况下重传次数依赖于mr->retry_count，这个变量是在每次更新各个速率状态之后更新的，如果该速率的投递率小于10%，初始为1，否则初始为2，然后根据帧平均发送时间还要适当增加。

最后就是MRR，也就是多速率重传的其它几个速率怎么设置了：

```

if (mp->hw->max_rates >= 3) {
    if (sample_idx >= 0)
        minstrel_ht_set_rate(mp, mi, &ar[1], mi->max_tp_rate, false, false);
    else
        minstrel_ht_set_rate(mp, mi, &ar[1], mi->max_tp_rate2, false, true);

    minstrel_ht_set_rate(mp, mi, &ar[2], mi->max_prob_rate, false, !sample);

    ar[3].count = 0;
    ar[3].idx = -1;
} else if (mp->hw->max_rates == 2) {
    minstrel_ht_set_rate(mp, mi, &ar[1], mi->max_prob_rate, false, !sample);

    ar[2].count = 0;
    ar[2].idx = -1;
} else {
    ar[1].count = 0;
    ar[1].idx = -1;
}

```



如果驱动支持的速率个数多于3，那么按照max_tp_rate>max_prob_rate的顺序来设置剩下两个速率，如果第一个速率不作为探测速率，也就是说第一个速率是用max_tp_rate，那么第二三个速率就用max_tp_rate2>max_prob_rate，如果底层支持的多速率就是能支持两个，则第二速率就用max_prob_rate发送，这个速率是投递率最高的速率，确保在有限次发送后正确传输。

4. 更新各个速率状态

minstrel_ht_tx_status函数是当每个帧发送完成时的回调函数，但是每个聚合帧中只有一个帧携带了该聚合帧都使用了哪些速率发送、哪一次发送才成功等我们需要的信息，因此对于未携带这些信息的帧，直接返回不予处理：

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

之后根据ar[0]、ar[1]和ar[2]里面携带的实际发送次数，得到每个速率在本次发送中共发送了多少个帧（指单帧）、成功了多少个（单帧）。然后进入核心处理逻辑：

```
rate = minstrel_get_ratestats(mi, mi->max_tp_rate);
if (rate->attempts > 30 &&
    MINSTREL_FRAC(rate->success, rate->attempts) <
    MINSTREL_FRAC(20, 100))
    minstrel_downgrade_rate(mi, &mi->max_tp_rate, true);

rate2 = minstrel_get_ratestats(mi, mi->max_tp_rate2);
if (rate2->attempts > 30 &&
    MINSTREL_FRAC(rate2->success, rate2->attempts) <
    MINSTREL_FRAC(20, 100))
    minstrel_downgrade_rate(mi, &mi->max_tp_rate2, false);

if (time_after(jiffies, mi->stats_update + (mp->update_interval / 2 * HZ) / 1000)) {
    minstrel_ht_update_stats(mp, mi);
    if (!(info->flags & IEEE80211_TX_CTL_AMPDU))
        minstrel_aggr_check(sta, skb);
}
```

minstrel_downgrade_rate是把系统当前的max_tp_rate或者max_tp_rate2切换到比原来速率所在组的前一个较低或相同流数的组的对应值上，比如，现在的max_tp_rate是第3个group的max_tp_rate，此时就会判断，第2组包含的速率都是几个空间流的，如果组3是双流流的，则组2是单流或双流的话，就把组2的max_tp_rate作为当前系统的max_tp_rate，如果组3是单流，但组2是双流，就会再看组1是不是单流，以此类推。

根据驱动的注释，是为了防止空间流突然不能用，比如本来用的双流，有一个流突然不起作用了，所以这里判断一下，如果max_tp_rate或者max_tp_rate2已经尝试30次以上，投递率还不超过20%，则降空间流，最后要解决的一个问题就是，每个组的max_tp_rate和max_tp_rate2是怎么算出来的，看代码的最后几行，每过(mp->update_interval / 2 * HZ) / 1000 这些时间（单位是jiffies），就会更新整个速率表中各个mcs_group，针对每个mcs_group，会遍历8个MCS：

```
for (i = 0; i < MCS_GROUP_RATES; i++) {
    if (!(mg->supported & BIT(i)))
        continue;

    mr = &mg->rates[i];
    mr->retry_updated = false;
    index = MCS_GROUP_RATES * group + i;
    minstrel_calc_rate_ewma(mr);
    minstrel_ht_calc_tp(mi, group, i);
}
```

针对每一个速率，也就是mr，都会调用minstrel_calc_rate_ewma去计算吞吐率和投递率的加权平均：

```
mr->sample_skipped = 0;
mr->cur_prob = MINSTREL_FRAC(mr->success, mr->attempts);
if (!mr->att_hist)
    mr->probability = mr->cur_prob;
else
    mr->probability = minstrel_ewma(mr->probability, mr->cur_prob, EWMA_LEVEL);
```

```
mr->att_hist += mr->attempts;
mr->succ_hist += mr->success;
```

计算这个速率在这个计算周期内的投递率cur_prob，EWMA_LEVEL默认为75，如果之前没有发送成功过，也就是历史尝试数等于0，就直接更新该速率的投递率，否则调用minstrel_ewma以（原投递率*75%+本次投递率*25%）的计算方式更新投递率。算完了prob的ewma之后就调用minstrel_ht_calc_tp计算各个速率的吞吐率，这个吞吐率结合了实际数据包的长度、各个编码方案的数据率，应该还是比较准确的算法。

在每个循环中，在算完当前速率的投递率和吞吐率之后，就会和当前组的最佳值进行比较，如果优于最佳值，就进行交换，再把以前的最佳值和以前的max_tp_rate2比较：

```
if ((mr->cur_tp > cur_prob_tp && mr->probability >
    MINSTREL_FRAC(3, 4)) || mr->probability > cur_prob) {
    mg->max_prob_rate = index;
    cur_prob = mr->probability;
    cur_prob_tp = mr->cur_tp;
}

if (mr->cur_tp > cur_tp) {
    swap(index, mg->max_tp_rate);
    cur_tp = mr->cur_tp;
    mr = minstrel_get_ratestats(mi, index);
}

if (index >= mg->max_tp_rate)
    continue;

if (mr->cur_tp > cur_tp2) {
    mg->max_tp_rate2 = index;
    cur_tp2 = mr->cur_tp;
}
```

最后再用同样的方法遍历各个组，更新整个系统的max_tp_rate、max_tp_rate2等域，不再重复帖代码。

5. 探测频率

那么minstrel多久或者说在什么条件下会进行探测呢，这和前文提到的三个重要的变量有关，就是sample_wait、sample_tries和sample_count。我的理解，sample_wait是说再过多少个数据包之后进行探测，sample_tries是说拿几个帧来探测，但不是说经过一个sample_wait+sample_tries之后就开始计算那个速率最好，什么时候重新计算各个速率的吞吐率是用时间来控制的，在下次计算之前，最多进行sample_count组（sample_wait+sample_tries）的循环。

在聚合帧发送完成的回调函数minstrel_ht_tx_status中，对这几个值进行判断：

```
if (!mi->sample_wait && !mi->sample_tries && mi->sample_count > 0) {
    mi->sample_wait = 16 + 2 * MINSTREL_TRUNC(mi->avg_ampdu_len);
    mi->sample_tries = 2;
    mi->sample_count--;
}
```

mi->avg_ampdu_len是每个聚合帧聚合长度的加权平均：

```
mi->avg_ampdu_len = minstrel_ewma(mi->avg_ampdu_len, MINSTREL_FRAC(mi->ampdu_len, mi->
    ampdu_packets), EWMA_LEVEL);
```

而在一个数据包将要发送请求采样速率sample_idx的minstrel_get_sample_rate函数中，一开始会做如下处理：

```
if (mi->sample_wait > 0) {
    mi->sample_wait--;
    return -1;
}
```



```
if (!mi->sample_tries)
    return -1;

mi->sample_tries--;
```

因为要等sample_wait个包之后探测，因此先判断这个值是不是大于0，如果是，则把这个计数器减1，之后返回-1不进行探测。每次探测时会把控制探测数量的计数器sample_tries减1，当这个数已经减到0的时候返回-1不探测。

以上就是Minstrel速率调整算法的基本流程。

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

0

推荐

0

反对

« 上一篇：【Atheros】网卡驱动速率调整算法概述

» 下一篇：【Atheros】Ath9k速率调整算法源码走读

posted @ 2014-11-12 21:01 琴剑飘零 阅读(1605) 评论(12) 编辑 收藏

评论列表

- # 1楼 2016-05-11 15:26 tygy
- “

请教下博主，速率表里面的group数量为什么设置为10，在选择速率时是在哪里确定这个group index的，另外博主优化ath9k的主要思路能说一下吗？先谢过博主了~

支持(0) 反对(0)
- # 2楼[楼主] 2016-05-11 16:07 琴剑飘零
- “

@ tygy
印象中速率表里的那个10没有什么讲究，这十行速率表，每一行都包括随机排布的8个速率取值，所以每个MCS在速率表上的分布是一样的，然后顺序遍历速率表（也就是说group index就是顺序走的），起到抽随机数的效果，所以这个组数最好稍大一点（反向极限情况，如果只有一组，包含8个随机排列的速率，第一次遍历这个表是随机返回rate，第二轮遍历开始，rate的返回顺序就和之前一样了）

支持(0) 反对(0)
- # 3楼[楼主] 2016-05-11 16:08 琴剑飘零
- “

@ tygy
我没有优化ath9k，是自己又设计了一个基于rssi的算法，把ath9k的c代码文件重写了

支持(0) 反对(0)
- # 4楼 2016-05-11 16:23 tygy
- “

@
谢谢博主，我现在想尝试优化sdk的速率选择算法，他们现在采用的就是在ath9k的基础上结合rssi后的版本，你有采用minstrel的随机选择探测速率的思想吗？结合rssi效果会优于单纯采用报文统计的方法吗？另外结合rssi的算法是不是报文达到一定数量时就不采用rssi判断，只有低于一定数量才采用rssi判断吗？请博主多多指教~

支持(0) 反对(0)
- # 5楼[楼主] 2016-05-11 19:14 琴剑飘零
- “

@ tygy
你说的这个版本的具体实现我不知道，所以你说的这两种判断方法怎么相互触发，我没法给你回答，至于效果是不是会比纯用报文统计要好，得看具体场景，如果是多径效应明显、环境多变（比如经常人来人往）的室内场景，rssi不会带来什么帮助，最后再说明一下，我说的“把ath9k的c代码文件重写了”不是在ath9k上扩展加入rssi，而是把ath9的代码全删了，只保留关键的函数声明，然后填进去我的新算法，跟之前的ath9k没有关系

支持(0) 反对(0)
- # 6楼 2016-05-11 19:22 tygy
- “

@ 琴剑飘零
谢谢博主的耐心解答，学习了~

支持(0) 反对(0)
- # 7楼 2016-09-28 19:17 61个夜

- “

楼主您好，传统速率和MCS速率表在哪里呢？找了半天没有找到哇，都不知道去哪里找，新人求教

支持(0) 反对(0)
- #8楼[楼主👤] 2016-09-29 09:24 琴剑飘零📧

“

@ 61个夜
在rc.c这个文件的开头有针对802.11协议簇各个版本的速率表~

支持(0) 反对(0)
- #9楼 2016-12-06 20:16 杰尔米📧

“

@琴剑飘零 请教博主，ath9k最终引用的速率发送表在哪里？我只找到了ath9k/common-init.c里面的ath9k_legacy_rates这个数组，但是为什么驱动里的速率列表只有10, 20, 55, 110, 60, 90, 120, 180, 240, 360, 480, 540 Kbps这几种，11n的速率不是可以达到150Mbps和300Mbps甚至更高的吗？它们之间是怎么对应的？

支持(0) 反对(0)
- #10楼[楼主👤] 2016-12-06 23:02 琴剑飘零📧

“

@ 杰尔米
11n的速率和11b/g的速率不能直接对应，ath9k用的就在ath9k算法的文件里：drivers\net\wireless\ath\ath9k\rc.c

支持(0) 反对(0)
- #11楼 2016-12-06 23:04 杰尔米📧

“

@ 琴剑飘零
我的驱动里面怎么没有rc.c这个文件啊？

ahb.c ar9003_buffalo_initvals.h ar9330_1p2_initvals.h calib.h dfs_debug.c htc_drv_init.c mci.c
ani.c ar9003_calib.c ar9340_initvals.h channel.c dfs_debug.h htc_drv_main.c mci.h
ani.h ar9003_eeprom.c ar9462_2p0_initvals.h common-beacon.c dfs.h htc_drv_txrx.c pci.c
antenna.c ar9003_eeprom.h ar9462_2p1_initvals.h common-beacon.h dynack.c htc.h phy.h
ar5008_initvals.h ar9003_hw.c ar9485_initvals.h common.c dynack.h htc_hst.c rcv.c
ar5008_phy.c ar9003_mac.c ar953x_initvals.h common-debug.c eeprom_4k.c htc_hst.h reg_aic.h
ar9001_initvals.h ar9003_mac.h ar955x_1p0_initvals.h common-debug.h eeprom_9287.c hw.c reg.h
ar9002_calib.c ar9003_mci.c ar9565_1p0_initvals.h common.h eeprom.c hw.h reg_mci.h
ar9002_hw.c ar9003_mci.h ar9565_1p1_initvals.h common-init.c eeprom_def.c hw-ops.h reg_wow.h
ar9002_initvals.h ar9003_paprd.c ar956x_initvals.h common-init.h eeprom.h init.c tx99.c
ar9002_mac.c ar9003_phy.c ar9580_1p0_initvals.h common-spectral.c gpio.c Kconfig wmi.c
ar9002_phy.c ar9003_phy.h ath9k.h common-spectral.h hif_usb.c link.c wmi.h
ar9002_phy.h ar9003_rtt.c beacon.c debug.c hif_usb.h mac.c wow.c
ar9003_2p2_initvals.h ar9003_rtt.h btcoex.c debug.h htc_drv_beacon.c mac.h xmit.c
ar9003_aic.c ar9003_wow.c btcoex.h debug_sta.c htc_drv_debug.c main.c
ar9003_aic.h ar9330_1p1_initvals.h calib.c dfs.c htc_drv_gpio.c Makefile


支持(0) 反对(0)
- #12楼 2016-12-07 00:01 杰尔米📧

“

@ 琴剑飘零
那minstrel_ht最终引用的速率表在哪个地方呢？

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
Meituan Cloud

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...



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...