Linux Cross Reference

Free Electrons

Embedded Linux Experts

• *source navigation* • diff markup • identifier search • freetext search •

Version: 2.0.40 2.2.26 2.4.37 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16 ***3.17***

Linux/net/ipv4/tcp_highspeed.c

```
 1 /*
 2  * Sally Floyd's High Speed TCP (RFC 3649) congestion control
 3  *
 4  * See http://www.icir.org/floyd/hstcp.html
 5  *
 6  * John Heffner <jheffner@psc.edu>
 7  */
 8
 9 #include <linux/module.h>
10 #include <net/tcp.h>
11
12
13 /* From AIMD tables from RFC 3649 appendix B,
14  * with fixed-point MD scaled <<8.
15  */
16 static const struct hstcp_aimd_val {
17         unsigned int cwnd;
18         unsigned int md;
19 } hstcp_aimd_vals[] = {
20  {     38,  128, /*  0.50 */ },
21  {    118,  112, /*  0.44 */ },
22  {    221,  104, /*  0.41 */ },
23  {    347,   98, /*  0.38 */ },
24  {    495,   93, /*  0.37 */ },
25  {    663,   89, /*  0.35 */ },
26  {    851,   86, /*  0.34 */ },
27  {   1058,   83, /*  0.33 */ },
28  {   1284,   81, /*  0.32 */ },
29  {   1529,   78, /*  0.31 */ },
30  {   1793,   76, /*  0.30 */ },
31  {   2076,   74, /*  0.29 */ },
32  {   2378,   72, /*  0.28 */ },
33  {   2699,   71, /*  0.28 */ },
34  {   3039,   69, /*  0.27 */ },
35  {   3399,   68, /*  0.27 */ },
36  {   3778,   66, /*  0.26 */ },
37  {   4177,   65, /*  0.26 */ },
38  {   4596,   64, /*  0.25 */ },
39  {   5036,   62, /*  0.25 */ },
40  {   5497,   61, /*  0.24 */ },
41  {   5979,   60, /*  0.24 */ },
42  {   6483,   59, /*  0.23 */ },
43  {   7009,   58, /*  0.23 */ },
44  {   7558,   57, /*  0.22 */ },
45  {   8130,   56, /*  0.22 */ },
```

```
 46  {   8726,   55, /*  0.22 */ },
 47  {   9346,   54, /*  0.21 */ },
 48  {   9991,   53, /*  0.21 */ },
 49  {  10661,   52, /*  0.21 */ },
 50  {  11358,   52, /*  0.20 */ },
 51  {  12082,   51, /*  0.20 */ },
 52  {  12834,   50, /*  0.20 */ },
 53  {  13614,   49, /*  0.19 */ },
 54  {  14424,   48, /*  0.19 */ },
 55  {  15265,   48, /*  0.19 */ },
 56  {  16137,   47, /*  0.19 */ },
 57  {  17042,   46, /*  0.18 */ },
 58  {  17981,   45, /*  0.18 */ },
 59  {  18955,   45, /*  0.18 */ },
 60  {  19965,   44, /*  0.17 */ },
 61  {  21013,   43, /*  0.17 */ },
 62  {  22101,   43, /*  0.17 */ },
 63  {  23230,   42, /*  0.17 */ },
 64  {  24402,   41, /*  0.16 */ },
 65  {  25618,   41, /*  0.16 */ },
 66  {  26881,   40, /*  0.16 */ },
 67  {  28193,   39, /*  0.16 */ },
 68  {  29557,   39, /*  0.15 */ },
 69  {  30975,   38, /*  0.15 */ },
 70  {  32450,   38, /*  0.15 */ },
 71  {  33986,   37, /*  0.15 */ },
 72  {  35586,   36, /*  0.14 */ },
 73  {  37253,   36, /*  0.14 */ },
 74  {  38992,   35, /*  0.14 */ },
 75  {  40808,   35, /*  0.14 */ },
 76  {  42707,   34, /*  0.13 */ },
 77  {  44694,   33, /*  0.13 */ },
 78  {  46776,   33, /*  0.13 */ },
 79  {  48961,   32, /*  0.13 */ },
 80  {  51258,   32, /*  0.13 */ },
 81  {  53677,   31, /*  0.12 */ },
 82  {  56230,   30, /*  0.12 */ },
 83  {  58932,   30, /*  0.12 */ },
 84  {  61799,   29, /*  0.12 */ },
 85  {  64851,   28, /*  0.11 */ },
 86  {  68113,   28, /*  0.11 */ },
 87  {  71617,   27, /*  0.11 */ },
 88  {  75401,   26, /*  0.10 */ },
 89  {  79517,   26, /*  0.10 */ },
 90  {  84035,   25, /*  0.10 */ },
 91  {  89053,   24, /*  0.10 */ },
 92 };
 93
 94 #define HSTCP_AIMD_MAX  ARRAY_SIZE(hstcp_aimd_vals)
 95
 96 struct hstcp {
 97         u32     ai;
 98 };
 99
100 static void hstcp_init(struct sock *sk)
101 {
102         struct tcp_sock *tp = tcp_sk(sk);
103         struct hstcp *ca = inet_csk_ca(sk);
104
105         ca->ai = 0;
106
107         /* Ensure the MD arithmetic works.  This is somewhat pedantic,
108          * since I don't think we will see a cwnd this large. :) */
109         tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp, 0xffffffff/128);
110 }
111
```

```
112 static void hstcp_cong_avoid(struct sock *sk, u32 ack, u32 acked)
113 {
114         struct tcp_sock *tp = tcp_sk(sk);
115         struct hstcp *ca = inet_csk_ca(sk);
116
117         if (!tcp_is_cwnd_limited(sk))
118                 return;
119
120         if (tp->snd_cwnd <= tp->snd_ssthresh)
121                 tcp_slow_start(tp, acked);
122         else {
123                 /* Update AIMD parameters.
124                  *
125                  * We want to guarantee that:
126                  *     hstcp_aimd_vals[ca->ai-1].cwnd <
127                  *     snd_cwnd <=
128                  *     hstcp_aimd_vals[ca->ai].cwnd
129                  */
130                 if (tp->snd_cwnd > hstcp_aimd_vals[ca->ai].cwnd) {
131                         while (tp->snd_cwnd > hstcp_aimd_vals[ca->ai].cwnd &&
132                                ca->ai < HSTCP_AIMD_MAX - 1)
133                                 ca->ai++;
134                 } else if (ca->ai && tp->snd_cwnd <= hstcp_aimd_vals[ca->ai-1].cwnd) {
135                         while (ca->ai && tp->snd_cwnd <= hstcp_aimd_vals[ca->ai-1].cwnd)
136                                 ca->ai--;
137                 }
138
139                 /* Do additive increase */
140                 if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
141                         /* cwnd = cwnd + a(w) / cwnd */
142                         tp->snd_cwnd_cnt += ca->ai + 1;
143                         if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
144                                 tp->snd_cwnd_cnt -= tp->snd_cwnd;
145                                 tp->snd_cwnd++;
146                         }
147                 }
148         }
149 }
150
151 static u32 hstcp_ssthresh(struct sock *sk)
152 {
153         const struct tcp_sock *tp = tcp_sk(sk);
154         const struct hstcp *ca = inet_csk_ca(sk);
155
156         /* Do multiplicative decrease */
157         return max(tp->snd_cwnd - ((tp->snd_cwnd * hstcp_aimd_vals[ca->ai].md) >> 8), 2U);
158 }
159
160
161 static struct tcp_congestion_ops tcp_highspeed __read_mostly = {
162         .init           = hstcp_init,
163         .ssthresh       = hstcp_ssthresh,
164         .cong_avoid     = hstcp_cong_avoid,
165
166         .owner          = THIS_MODULE,
167         .name           = "highspeed"
168 };
169
170 static int __init hstcp_register(void)
171 {
172         BUILD_BUG_ON(sizeof(struct hstcp) > ICSK_CA_PRIV_SIZE);
173         return tcp_register_congestion_control(&tcp_highspeed);
174 }
175
176 static void __exit hstcp_unregister(void)
```

```
177 {
178         tcp_unregister_congestion_control(&tcp_highspeed);
179 }
180
181 module_init(hstcp_register);
182 module_exit(hstcp_unregister);
183
184 MODULE_AUTHOR("John Heffner");
185 MODULE_LICENSE("GPL");
186 MODULE_DESCRIPTION("High Speed TCP");
187
```