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_probe.c

```
/*
 * tcpprobe - Observe the TCP flow with kprobes.
 *
 * The idea for this came from Werner Almesberger's umlsim
 * Copyright (C) 2004, Stephen Hemminger <shemminger@osdl.org>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt

#include <linux/kernel.h>
#include <linux/kprobes.h>
#include <linux/socket.h>
#include <linux/tcp.h>
#include <linux/slab.h>
#include <linux/proc_fs.h>
#include <linux/module.h>
#include <linux/ktime.h>
#include <linux/time.h>
#include <net/net_namespace.h>

#include <net/tcp.h>

MODULE_AUTHOR("Stephen Hemminger <shemminger@linux-foundation.org>");
MODULE_DESCRIPTION("TCP cwnd snooper");
MODULE_LICENSE("GPL");
MODULE_VERSION("1.1");

static int port __read_mostly;
MODULE_PARM_DESC(port, "Port to match (0=all)");
module_param(port, int, 0);
```

```

static unsigned int bufsize __read_mostly = 4096;
MODULE_PARM_DESC(bufsize, "Log buffer size in packets (4096)");
module_param(bufsize, uint, 0);

static unsigned int fwmark __read_mostly;
MODULE_PARM_DESC(fwmark, "skb mark to match (0=no mark)");
module_param(fwmark, uint, 0);

static int full __read_mostly;
MODULE_PARM_DESC(full, "Full log (1=every ack packet received,  0=only cwnd changes)");
module_param(full, int, 0);

static const char procname[] = "tcpprobe";

struct tcp_log {
        ktime_t tstamp;
        union {
                struct sockaddr         raw;
                struct sockaddr_in      v4;
                struct sockaddr_in6     v6;
        }       src, dst;
        u16     length;
        u32     snd_nxt;
        u32     snd_una;
        u32     snd_wnd;
        u32     rcv_wnd;
        u32     snd_cwnd;
        u32     ssthresh;
        u32     srtt;
};

static struct {
        spinlock_t      lock;
        wait_queue_head_t wait;
        ktime_t         start;
        u32             lastcwnd;

        unsigned long   head, tail;
        struct tcp_log  *log;
} tcp_probe;


static inline int tcp_probe_used(void)
{
        return (tcp_probe.head - tcp_probe.tail) & (bufsize - 1);
}

static inline int tcp_probe_avail(void)
{
        return bufsize - tcp_probe_used() - 1;
}

#define tcp_probe_copy_fl_to_si4(inet, si4, mem)                \
        do {                                                    \
                si4.sin_family = AF_INET;                       \
                si4.sin_port = inet->inet_##mem##port;          \
                si4.sin_addr.s_addr = inet->inet_##mem##addr;   \
        } while (0)                                             \


/*
 * Hook inserted to be called before each receive packet.
 * Note: arguments must match tcp_rcv_established()!
 */
```

```
static void jtcp_rcv_established(struct sock *sk, struct sk_buff *skb,
                                 const struct tcphdr *th, unsigned int len)
{
        const struct tcp_sock *tp = tcp_sk(sk);
        const struct inet_sock *inet = inet_sk(sk);

        /* Only update if port or skb mark matches */
        if (((port == 0 && fwmark == 0) ||
             ntohs(inet->inet_dport) == port ||
             ntohs(inet->inet_sport) == port ||
             (fwmark > 0 && skb->mark == fwmark)) &&
            (full || tp->snd_cwnd != tcp_probe.lastcwnd)) {

                spin_lock(&tcp_probe.lock);
                /* If log fills, just silently drop */
                if (tcp_probe_avail() > 1) {
                        struct tcp_log *p = tcp_probe.log + tcp_probe.head;

                        p->tstamp = ktime_get();
                        switch (sk->sk_family) {
                        case AF_INET:
                                tcp_probe_copy_fl_to_si4(inet, p->src.v4, s);
                                tcp_probe_copy_fl_to_si4(inet, p->dst.v4, d);
                                break;
                        case AF_INET6:
                                memset(&p->src.v6, 0, sizeof(p->src.v6));
                                memset(&p->dst.v6, 0, sizeof(p->dst.v6));
#if IS_ENABLED(CONFIG_IPV6)
                                p->src.v6.sin6_family = AF_INET6;
                                p->src.v6.sin6_port = inet->inet_sport;
                                p->src.v6.sin6_addr = inet6_sk(sk)->saddr;

                                p->dst.v6.sin6_family = AF_INET6;
                                p->dst.v6.sin6_port = inet->inet_dport;
                                p->dst.v6.sin6_addr = sk->sk_v6_daddr;
#endif
                                break;
                        default:
                                BUG();
                        }

                        p->length = skb->len;
                        p->snd_nxt = tp->snd_nxt;
                        p->snd_una = tp->snd_una;
                        p->snd_cwnd = tp->snd_cwnd;
                        p->snd_wnd = tp->snd_wnd;
                        p->rcv_wnd = tp->rcv_wnd;
                        p->ssthresh = tcp_current_ssthresh(sk);
                        p->srtt = tp->srtt_us >> 3;

                        tcp_probe.head = (tcp_probe.head + 1) & (bufsize - 1);
                }
                tcp_probe.lastcwnd = tp->snd_cwnd;
                spin_unlock(&tcp_probe.lock);

                wake_up(&tcp_probe.wait);
        }

        jprobe_return();
}

static struct jprobe tcp_jprobe = {
        .kp = {
                .symbol_name    = "tcp_rcv_established",
        },
        .entry  = jtcp_rcv_established,
};

static int tcpprobe_open(struct inode *inode, struct file *file)
{
        /* Reset (empty) log */
        spin_lock_bh(&tcp_probe.lock);
        tcp_probe.head = tcp_probe.tail = 0;
        tcp_probe.start = ktime_get();
        spin_unlock_bh(&tcp_probe.lock);

        return 0;
}

static int tcpprobe_sprint(char *tbuf, int n)
{
        const struct tcp_log *p
                = tcp_probe.log + tcp_probe.tail;
        struct timespec tv
                = ktime_to_timespec(ktime_sub(p->tstamp, tcp_probe.start));

        return scnprintf(tbuf, n,
                        "%lu.%09lu %pISpc %pISpc %d %#x %#x %u %u %u %u\n",
                        (unsigned long) tv.tv_sec,
                        (unsigned long) tv.tv_nsec,
                        &p->src, &p->dst, p->length, p->snd_nxt, p->snd_una,
                        p->snd_cwnd, p->ssthresh, p->snd_wnd, p->srtt, p->rcv_wnd);
}

static ssize_t tcpprobe_read(struct file *file, char __user *buf,
                             size_t len, loff_t *ppos)
{
        int error = 0;
        size_t cnt = 0;

        if (!buf)
                return -EINVAL;

        while (cnt < len) {
                char tbuf[256];
                int width;

                /* Wait for data in buffer */
                error = wait_event_interruptible(tcp_probe.wait,
                                                 tcp_probe_used() > 0);
                if (error)
                        break;

                spin_lock_bh(&tcp_probe.lock);
                if (tcp_probe.head == tcp_probe.tail) {
                        /* multiple readers race? */
                        spin_unlock_bh(&tcp_probe.lock);
                        continue;
                }

                width = tcpprobe_sprint(tbuf, sizeof(tbuf));

                if (cnt + width < len)
                        tcp_probe.tail = (tcp_probe.tail + 1) & (bufsize - 1);

                spin_unlock_bh(&tcp_probe.lock);

                /* if record greater than space available
                   return partial buffer (so far) */
                if (cnt + width >= len)
```

```
                        break;

                if (copy_to_user(buf + cnt, tbuf, width))
                        return -EFAULT;
                cnt += width;
        }

        return cnt == 0 ? error : cnt;
}

static const struct file_operations tcpprobe_fops = {
        .owner   = THIS_MODULE,
        .open    = tcpprobe_open,
        .read    = tcpprobe_read,
        .llseek  = noop_llseek,
};

static __init int tcpprobe_init(void)
{
        int ret = -ENOMEM;

        /* Warning: if the function signature of tcp_rcv_established,
         * has been changed, you also have to change the signature of
         * jtcp_rcv_established, otherwise you end up right here!
         */
        BUILD_BUG_ON(__same_type(tcp_rcv_established,
                                 jtcp_rcv_established) == 0);

        init_waitqueue_head(&tcp_probe.wait);
        spin_lock_init(&tcp_probe.lock);

        if (bufsize == 0)
                return -EINVAL;

        bufsize = roundup_pow_of_two(bufsize);
        tcp_probe.log = kcalloc(bufsize, sizeof(struct tcp_log), GFP_KERNEL);
        if (!tcp_probe.log)
                goto err0;

        if (!proc_create(procname, S_IRUSR, init_net.proc_net, &tcpprobe_fops))
                goto err0;

        ret = register_jprobe(&tcp_jprobe);
        if (ret)
                goto err1;

        pr_info("probe registered (port=%d/fwmark=%u) bufsize=%u\n",
                port, fwmark, bufsize);
        return 0;
 err1:
        remove_proc_entry(procname, init_net.proc_net);
 err0:
        kfree(tcp_probe.log);
        return ret;
}
module_init(tcpprobe_init);

static __exit void tcpprobe_exit(void)
{
        remove_proc_entry(procname, init_net.proc_net);
        unregister_jprobe(&tcp_jprobe);
        kfree(tcp_probe.log);
}
module_exit(tcpprobe_exit);

```

- Home
- Development
- Services
- Training
- Docs
- Community
- Company
- Blog