MPTCP 发送

Wednesday, August 5, 2015 10:28 PM

http://www.cnblogs.com/lxgeek/p/4187164.html

下面的情景是服务端收到上图1中ACK/MP_JOIN(HMAC-A)包，这时状态将由SYN_RECV变为ESTABLISHED。
函数的调用

关系如下：

```
tcp_v4_rcv
          =》tcp_v4_do_rcv
               =》mptcp_v4_do_rcv
                    =》tcp_v4_hnd_req
                         =》tcp_check_req
                              =》mptcp_check_req_child
                                   =》mptcp_add_sock
```

mptcp_sched.c mptcp_next_segment()>>get_available_subflow

tcp_write_wakeup>>mptcp_write_wakeup>>mptcp_skb_entail

mptcp_output.c

tcp_sendmsg>>push>>mptcp_write_xmit(这里就分segment，并且分出了不同subsk)
>>mptcp_skb_entail
>>mptcp_save_dss_data_seq
>>mptcp_write_dss_mapping对 Data Sequeue Number 和 Subflow Sequence Number进行了赋值

http://www.cnblogs.com/lxgeek/p/4330119.html

>>push(subsk)>>tcp_write_xmit>>tcp_transmit_skb

tcp.h tcp_sock

mptcp.h mpcb mptcp_cb

```
struct mptcp_cb {
    struct sock *meta_sk;
```

```
/* list of sockets in this multipath connection */
struct tcp_sock *connection_list;
/* list of sockets that need a call to release_cb */
struct list_head callback_list;


spinlock_t     tw_lock;
struct list_head tw_list;
unsigned char     mptw_state;


atomic_t    mpcb_refcnt;


/* High-order bits of 64-bit sequence numbers */
u32 snd_high_order[2];
u32 rcv_high_order[2];


u16    send_infinite_mapping:1,
    in_time_wait:1,
    list_rcvd:1, /* XXX TO REMOVE */
    dss_csum:1,
    server_side:1,
    infinite_mapping_rcv:1,
    infinite_mapping_snd:1,
    dfin_combined:1,   /* Was the DFIN combined with subflow-fin? */
    passive_close:1,
    snd_hiseq_index:1, /* Index in snd_high_order of snd_nxt */
    rcv_hiseq_index:1; /* Index in rcv_high_order of rcv_nxt */


/* socket count in this connection */
u8 cnt_subflows;
u8 cnt_established;


struct sk_buff_head reinject_queue;


u8 dfin_path_index;
```

```
#define MPTCP_PM_SIZE 608
    u8 mptcp_pm[MPTCP_PM_SIZE] __aligned(8);
    struct mptcp_pm_ops *pm_ops;


    struct mptcp_sched_ops *sched_ops;


    /* Mutex needed, because otherwise mptcp_close will complain that the
     * socket is owned by the user.
     * E.g., mptcp_sub_close_wq is taking the meta-lock.
     */
    struct mutex mpcb_mutex;


    /* Master socket, also part of the connection_list, this
     * socket is the one that the application sees.
     */
    struct sock *master_sk;


    u64    csum_cutoff_seq;


    __u64    mptcp_loc_key;
    __u32    mptcp_loc_token;
    __u64    mptcp_rem_key;
    __u32    mptcp_rem_token;


    /* Create a new subflow - necessary because the meta-sk may be IPv4, but
     * the new subflow can be IPv6
     */
    struct sock *(*syn_recv_sock)(struct sock *sk, struct sk_buff *skb,
                      struct request_sock *req,
                      struct dst_entry *dst);


    u32 path_index_bits;
    /* Next pi to pick up in case a new path becomes available */
    u8 next_path_index;
```

```
    /* Original snd/rcvbuf of the initial subflow.
     * Used for the new subflows on the server-side to allow correct
     * autotuning
     */
    int orig_sk_rcvbuf;
    int orig_sk_sndbuf;
    u32 orig_window_clamp;


    /* Timer for retransmitting SYN/ACK+MP_JOIN */
    struct timer_list synack_timer;
};


tcp_established_opt->mptcp_established_opt
static unsigned int tcp_established_options(struct sock *sk, struct sk_buff *skb,
                    struct tcp_out_options *opts,
                    struct tcp_md5sig_key **md5)
{
    struct tcp_skb_cb *tcb = skb ? TCP_SKB_CB(skb) : NULL;
    struct tcp_sock *tp = tcp_sk(sk);
    unsigned int size = 0;
    unsigned int eff_sacks;


    opts->options = 0;


#ifdef CONFIG_TCP_MD5SIG
    *md5 = tp->af_specific->md5_lookup(sk, sk);
    if (unlikely(*md5)) {
        opts->options |= OPTION_MD5;
        size += TCPOLEN_MD5SIG_ALIGNED;
    }
#else
    *md5 = NULL;
#endif


    if (likely(tp->rx_opt.tstamp_ok)) {
        opts->options |= OPTION_TS;
```

```
        opts->tsval = tcb ? tcb->when + tp->tsoffset : 0;
        opts->tsecr = tp->rx_opt.ts_recent;
        size += TCPOLEN_TSTAMP_ALIGNED;
    }
    if (mptcp(tp))
        mptcp_established_options(sk, skb, opts, &size);


    eff_sacks = tp->rx_opt.num_sacks + tp->rx_opt.dsack;
    if (unlikely(eff_sacks)) {
        const unsigned remaining = MAX_TCP_OPTION_SPACE - size;
        if (remaining < TCPOLEN_SACK_BASE_ALIGNED)
            opts->num_sack_blocks = 0;
        else
            opts->num_sack_blocks =
                min_t(unsigned int, eff_sacks,
                  (remaining - TCPOLEN_SACK_BASE_ALIGNED) /
                  TCPOLEN_SACK_PERBLOCK);
        if (opts->num_sack_blocks)
            size += TCPOLEN_SACK_BASE_ALIGNED +
                opts->num_sack_blocks * TCPOLEN_SACK_PERBLOCK;
    }


    return size;
}
```