Subject: Networkproblem: VE stops sending ACKs
Posted by kaymes on Sat, 02 Dec 2006 00:43:03 GMT
View Forum Message <> Reply to Message

Hi!
I've got a strange networkingproblem around here - the VE suddenly stops to send ACKs.

Setup is a VE with its own ip and proxyarp/iptables running on HN. Sometimes a tcp/ip-connection simply hangs and does not recover at all. I did look into it with ethereal and discovered the following situation:

The VE was downloading a larger file via http on a fast connection. Apparently a packet got lost, so the VE started to send dupACKs. Of course the sender kept sending packets, because the dupACKs hadn't arrived jet. So far so good. However, after 155 (154 in another case) dupACKs, the VE simply stopped sending any ACKs whatsoever. The sender kept retransmitting the missing packet, but the VE simply stayed quiet.

Any idea about what's going on here and how to solve this problem?

Cheers,
Konstantin

Subject: Re: Networkproblem: VE stops sending ACKs
Posted by kaymes on Sat, 02 Dec 2006 13:55:52 GMT
View Forum Message <> Reply to Message

On Saturday 02 December 2006 01:43, Konstantin Seiler wrote:
> I've got a strange networkingproblem around here - the VE suddenly stops to
> send ACKs.

Hi!
I finally found something related to that problem. It seems to be a problem with TCPRCVBUF in the VEs config. For a check I multiplied the values by 10 and the download went fine.
(By the way, what are considered proper values - the defaults caused the trouble)

So I interpret the situation as a Bug of OpenVZ: The VE doesn't take TCPRCVBUF into account when setting the TCP-Window-Size and the propagated window is too big for the available buffer.
If a packet gets lost now, the connection is doomed to die: The sender keeps sending packets and eventually the buffer is full. When the retransmission of the lost packet finally arrives, it is discarded because of the full buffer and the connection is in a deadlock.
The retransmission can't be accepted because of the full buffer and the buffer

can't be emptied, because a packet is missing.

Cheers,
Konstantin

Subject: Re: Networkproblem: VE stops sending ACKs
Posted by dev on Wed, 13 Dec 2006 12:54:35 GMT
View Forum Message <> Reply to Message

Konstantin,

can you please check the patch attached whether it helps?

Thanks,
Kirill

> On Saturday 02 December 2006 01:43, Konstantin Seiler wrote:
>
>>I've got a strange networkingproblem around here - the VE suddenly stops to
>>send ACKs.
>
>
> Hi!
> I finally found something related to that problem. It seems to be a problem
> with TCPRCVBUF in the VEs config. For a check I multiplied the values by 10
> and the download went fine.
> (By the way, what are considered proper values - the defaults caused the
> trouble)
>
> So I interpret the situation as a Bug of OpenVZ: The VE doesn't take TCPRCVBUF
> into account when setting the TCP-Window-Size and the propagated window is
> too big for the available buffer.
> If a packet gets lost now, the connection is doomed to die: The sender keeps
> sending packets and eventually the buffer is full. When the retransmission of
> the lost packet finally arrives, it is discarded because of the full buffer
> and the connection is in a deadlock.
> The retransmission can't be accepted because of the full buffer and the buffer
> can't be emptied, because a packet is missing.
>
> Cheers,
> Konstantin
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 0ff49a5..7e8f200 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -1815,8 +1815,9 @@ static inline int tcp_win_from_space(int
 /* Note: caller must be prepared to deal with negative returns */

```
 static inline int tcp_space(const struct sock *sk)
 {
- return tcp_win_from_space(sk->sk_rcvbuf -
-      atomic_read(&sk->sk_rmem_alloc));
+ int ub_tcp_rcvbuf = (int) sock_bc(sk)->ub_tcp_rcvbuf;
+ return tcp_win_from_space(min(sk->sk_rcvbuf, ub_tcp_rcvbuf)
+       - atomic_read(&sk->sk_rmem_alloc));
 }

 static inline int tcp_full_space(const struct sock *sk)
diff --git a/include/ub/beancounter.h b/include/ub/beancounter.h
index 3d87afa..fc236e8 100644
--- a/include/ub/beancounter.h
+++ b/include/ub/beancounter.h
@@ -144,6 +144,8 @@ struct sock_private {
  unsigned long  ubp_rmem_thres;
  unsigned long  ubp_wmem_pressure;
  unsigned long  ubp_maxadvmss;
+ /* Total size of all advertised receive windows for all tcp sockets */
+ unsigned long          ubp_rcv_wnd;
  unsigned long  ubp_rmem_pressure;
 #define UB_RMEM_EXPAND         0
 #define UB_RMEM_KEEP           1
@@ -177,6 +179,7 @@ #define ub_held_pages  ppriv.ubp_held_pa
  struct sock_private spriv;
 #define ub_rmem_thres  spriv.ubp_rmem_thres
 #define ub_maxadvmss  spriv.ubp_maxadvmss
+#define ub_rcv_wnd  spriv.ubp_rcv_wnd
 #define ub_rmem_pressure spriv.ubp_rmem_pressure
 #define ub_wmem_pressure spriv.ubp_wmem_pressure
 #define ub_tcp_sk_list  spriv.ubp_tcp_socks
diff --git a/include/ub/ub_sk.h b/include/ub/ub_sk.h
index e65c9ed..02d0137 100644
--- a/include/ub/ub_sk.h
+++ b/include/ub/ub_sk.h
@@ -34,6 +34,8 @@ struct sock_beancounter {
   */
  unsigned long          poll_reserv;
  unsigned long  forw_space;
+ unsigned long          ub_tcp_rcvbuf;
+ unsigned long          ub_rcv_wnd_old;
  /* fields below are protected by bc spinlock */
  unsigned long          ub_waitspc;     /* space waiting for */
  unsigned long          ub_wcharged;
diff --git a/kernel/ub/ub_net.c b/kernel/ub/ub_net.c
index 74d651a..afee710 100644
--- a/kernel/ub/ub_net.c
+++ b/kernel/ub/ub_net.c
```

```
@@ -420,6 +420,7 @@ static int __sock_charge(struct sock *sk

  added_reserv = 0;
  added_forw = 0;
+ skbc->ub_rcv_wnd_old = 0;
  if (res == UB_NUMTCPSOCK) {
   added_reserv = skb_charge_size(MAX_TCP_HEADER +
     1500 - sizeof(struct iphdr) -
@@ -439,6 +440,7 @@ static int __sock_charge(struct sock *sk
    added_forw = 0;
   }
   skbc->forw_space = added_forw;
+  skbc->ub_tcp_rcvbuf = added_forw + SK_STREAM_MEM_QUANTUM;
  }
  spin_unlock_irqrestore(&ub->ub_lock, flags);

@@ -528,6 +530,7 @@ void ub_sock_uncharge(struct sock *sk)
          skbc->ub_wcharged, skbc->ub, skbc->ub->ub_uid);
  skbc->poll_reserv = 0;
  skbc->forw_space = 0;
+ ub->ub_rcv_wnd -= is_tcp_sock ? tcp_sk(sk)->rcv_wnd : 0;
  spin_unlock_irqrestore(&ub->ub_lock, flags);

  uncharge_beancounter_notop(skbc->ub,
@@ -768,6 +771,44 @@ static void ub_sockrcvbuf_uncharge(struc
  * UB_TCPRCVBUF
  */

+/*
+ * UBC TCP window management mechanism.
+ * Every socket may consume no more than sock_quantum.
+ * sock_quantum depends on space available and ub_parms[UB_NUMTCPSOCK].held.
+ */
+static void ub_sock_tcp_update_rcvbuf(struct user_beancounter *ub,
+          struct sock *sk)
+{
+ unsigned long allowed;
+ unsigned long reserved;
+ unsigned long available;
+ unsigned long sock_quantum;
+ struct tcp_opt *tp = tcp_sk(sk);
+ struct sock_beancounter *skbc;
+ skbc = sock_bc(sk);
+
+ if( ub->ub_parms[UB_NUMTCPSOCK].limit * ub->ub_maxadvmss
+     > ub->ub_parms[UB_TCPRCVBUF].limit) {
+  /* this is defenitly shouldn't happend */
+  return;
```

```
+ }
+ allowed = ub->ub_parms[UB_TCPRCVBUF].barrier;
+ ub->ub_rcv_wnd += (tp->rcv_wnd - skbc->ub_rcv_wnd_old);
+ skbc->ub_rcv_wnd_old = tp->rcv_wnd;
+ reserved = ub->ub_parms[UB_TCPRCVBUF].held + ub->ub_rcv_wnd;
+ available = (allowed < reserved)?
+  0:allowed - reserved;
+ sock_quantum = max(allowed / ub->ub_parms[UB_NUMTCPSOCK].held,
+     ub->ub_maxadvmss);
+ if ( skbc->ub_tcp_rcvbuf > sock_quantum) {
+  skbc->ub_tcp_rcvbuf = sock_quantum;
+ } else {
+  skbc->ub_tcp_rcvbuf += min(sock_quantum - skbc->ub_tcp_rcvbuf,
+        available);
+ }
+
+}
+
 int ub_sock_tcp_chargerecv(struct sock *sk, struct sk_buff *skb,
        enum ub_severity strict)
 {
@@ -804,6 +845,7 @@ int ub_sock_tcp_chargerecv(struct sock *
  retval = 0;
  for (ub = sock_bc(sk)->ub; ub->parent != NULL; ub = ub->parent);
  spin_lock_irqsave(&ub->ub_lock, flags);
+ ub_sock_tcp_update_rcvbuf(ub, sk);
  ub->ub_parms[UB_TCPRCVBUF].held += chargesize;
  if (ub->ub_parms[UB_TCPRCVBUF].held >
    ub->ub_parms[UB_TCPRCVBUF].barrier &&
```