Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms

David Hayes
Centre for Advanced Internet Architectures, Technical Report 100219A
Swinburne University of Technology
Melbourne, Australia
dahayes@swin.edu.au

Abstract—**The enhanced Round Trip Time module (h_ertt) is used by delay and rate based TCP congestion control algorithms in the Modular Congestion Control Framework for FreeBSD. Three key protocol issues need to be addressed when making robust accurate RTT estimates in the FreeBSD TCP stack: delayed acknowledgements, TCP Segmentation Offload, and Selective Acknowledgements. The h_ertt module algorithm addresses these providing an accurate robust measure of RTT.**

I. Introduction

Delay and rate based congestion control algorithms required robust accurate measurements of packet Round Trip Time (RTT). The h_ertt module provides these measurements for use by algorithms implemented within the Modular Congestion Control Framework[1] for FreeBSD. H_ertt is implemented as a Khelp (Kernel helper) module using the Khelp Module Framework for FreeBSD and is available, along with other NEWTCP tools, at [2].

The report proceeds by outlining three key protocol issues along with increased robustness for time stamp based measurements. Following this the enhanced timing algorithm is described. The report finishes with conclusions and future work.

II. Measuring RTT

Delay (and Rate) based congestion control indicators require a more accurate measurement of the RTT than the TCP stack's `SRTT` provides.

Key issues that need to be addressed for accurate timing are: the effect of delayed acknowledgements, TCP segmentation offload, selective acknowledgements, and time stamp robustness.

A. Delayed Acknowledgements

An accurate RTT measurement is made difficult when the receiver waits for a second packet so that it can use a single packet to acknowledge both the incoming packets. If the second packet does not arrive within a certain time after the first packet, a time-out occurs, following which an acknowledgement is sent for the first packet. This issue was highlighted in [3].

Solutions:

- The measurement algorithm guesses whether delayed acknowledgements have been employed by the receiver. It does this by looking for acknowledgements that acknowledge more than one packet.
- If delayed acknowledgements are detected, then the RTT is measured between the second sent packet and its acknowledgement. If time stamping was used, this is ignored since the receiver reflects the time stamp of the first packet.
- If delayed acknowledgements are detected, then the RTT is not measured when only one packet is acknowledged. This is necessary, since the measured RTT will include the delayed acknowledgement timeout.

B. TCP Segmentation Offload (TSO)

Timing is difficult when TSO is employed since the one template (time stamps included) is used for the entire transmission. To ensure at least one accurate measurement at least once per RTT, or if a TSO send is longer — once in between TSO sends, TSO is temporarily turned off for the sending of a single packet. This provides good timing information for the single packet and the first TSO sent packet, allowing for a receiver using duplicate acknowledgements.

C. Selective Acknowledgements

Selective acknowledgements, when enabled, are useful for ensuring accurate RTT measurements. The largest SACK acknowledgement is matched with its corresponding transmitted packet to make the measurement. Previous packets and SACK holes are skipped.

D. *Robustness*

The record of each transmitted *block* contains a copy of the send and receive time stamp if the time stamp option is enabled, otherwise it contains the current value of the kernel tick. A packet's reflected time stamp is used to help identify packet↔acknowledgement pairs (if sane and true), but not used in the RTT calculation. This prevents possible manipulation by a TCP receiver in an attempt to receive an unfair advantage.

III. Enhanced RTT measurement for FreeBSD — h_ertt

This section describes the data collection and RTT measurements performed by h_ertt.

A record is kept of each *block* of data sent until it is acknowledged. A *block* is equivalent to a packet, except when TSO is being used. Each record (named `txsi` for transmitted segment information) contains the segment's sequence number, send time (`tx_ts`), and the last receiver time stamp[1] (`rx_ts`). The records are placed at the end of a FreeBSD TAILQ (named `txsegi_q`). When acknowledgements are received they are matched with their corresponding transmissions in the TAILQ. If the ACK is valid, and can be timed accurately, the RTT is calculated. Refer to Algorithm 1 for the details about the RTT calculation, and Algorithm 2 for how the TCP sender guesses whether or not the receiver is using delayed acknowledgements.

A. *Per RTT Measurement Calculations*

The burstiness of the window flow control mechanism can significantly increase the variance of per packet RTT measurements. This can be especially problematic when a high speed sender is connected directly to a low speed bottleneck link, such as home computer connected to an ADSL modem. Measuring once per RTT eliminates this self induced noise, but at the expense of not seeing the full range of RTT measurements. The original Vegas[4] idea used the measurement of a marked packet once per RTT. The h_ertt module provides this measurement in addition to the per packet RTT measurements.

[1]if time stamps are being used

B. *Data available from h_ertt module*

The h_ertt module makes available the information shown in Data element 1. The ERTT_NEW_MEASUREMENT flag is used to indicate when a new marked packet (once per RTT) measurement is available for use by a congestion control algorithm. To initiate a new marked packet measurement, a congestion control algorithm clears the ERTT_NEW_MEASUREMENT flag.

IV. Conclusions and Future Work

The h_ertt module provides a robust estimate of the per packet and marked packet round trip times, suitable for use by modular delay and rate based TCP congestion control algorithms in FreeBSD. It is envisaged that some delay and rate based algorithms may require specialised processing of some of the measurements provided by the h_ertt module. Hooks for Khelp modules may be added to support this.

Acknowledgements

The development of rate based congestion control for FreeBSD is part of the NEWTCP project and was made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

References

[1] J. H. L. Stewart, "Light-weight modular TCP congestion control for FreeBSD 7," CAIA, Tech. Rep. 071218A, Dec. 2007. [Online]. Available: http://caia.swin.edu.au/reports/070717B/CAIA-TR-070717B.pdf

[2] "NewTCP project tools," Feb. 2010, accessed 19 February 2010. [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools.html

[3] G. D. McCullagh, "Exploring delay-based TCP congestion control," Master's thesis, The National University of Ireland, Feb. 2008.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

Algorithm 1 Basic algorithm for calculating RTT

```
if ACK acknowledges something new

  txsi = first entry in txsegi_q

  while txsegi_q is not empty

    if ACK acknowledges more than txsi
      remove txsi
      get next txsi from txsegi_q
      continue
    endif

    if reflected timestamp is more recent than txsi
      remove txsi
      get next txsi from txsegi_q
      continue
    endif

    if ACK acknowledges less than txsi
      break
    endif

    if receiver is not using delayed acknowledgements
     OR if ACK acknowledges more than one segment
     OR is SACK
      calculate new rtt -- ticks - txsi->tx_ts + 1
    endif

    remove txsi

  endwhile

endif
```

Algorithm 2 Delayed ACK guessing

```
if acknowledging unacknowledged segments, and not sack
  if acking more than maxseg
     dlyack_rx += 1
  else if acking more than txsi recorded length
     dlyack_rx += 1
  else if acking exactly txsi recorded length OR exactly maxseg
     dlyack_rx -= (dlyack_rx > 0) ? 1 : 0
  endif
endif
```

When `dlyack_rx` > 0, we guess that the receiver is using delayed acknowledgements.
Note `dlyack_rx` $\in [0, \texttt{DLYACK\_SMOOTH}]$

Data element 1 Available in h_ertt.h

```
/* Structure contains the information about the
   sent segment, for comparison with the corresponding ack */
struct txseginfo;

/* Structure used as the ertt data block. */
struct ertt {
    /* information about transmitted segments to aid in
       RTT calculation for delay/rate based CC */
    TAILQ_HEAD(txseginfo_head, txseginfo) txsegi_q;
    int rtt;   /* per packet measured round trip time */
    int maxrtt;             /* maximum seen rtt */
    int minrtt;             /* minimum seen rtt */
    int dlyack_rx;          /* guess if the receiver is using
                               delayed acknowledgements.*/
    int timestamp_errors;   /* for keeping track of inconsistencies
                               in packet timestamps */
    int markedpkt_rtt;      /* rtt for a marked packet */
    long bytes_tx_in_rtt;   /* bytes tx so far in marked rtt */
    long bytes_tx_in_marked_rtt;/* final version of above */
    u_long marked_snd_cwnd; /* cwnd for marked rtt */
    int flags;              /* flags*/
};

/* flags */
#define ERTT_NEW_MEASUREMENT          0x01 /* new measurement */
#define ERTT_MEASUREMENT_IN_PROGRESS  0x02 /* measuring marked RTT */
#define ERTT_TSO_DISABLED             0x04 /* indicates TSO has been
                                              temporarily disabled */
```