Internet Engineering Task Force
INTERNET DRAFT
File: draft-ietf-ippm-btc-cap-00.txt

Mark Allman
BBN/NASA GRC

February, 2001
Expires: August, 2001

A Bulk Transfer Capacity Methodology for Cooperating Hosts

Abstract

This document specifies a specific Bulk Transfer Capacity (BTC) metric based on the BTC framework outlined in [MA00].

1 Introduction

This document specifies a methodology that performs Bulk Transfer Capacity (BTC) measurements based on the BTC framework outlined in [MA00]. This particular methodology assumes cooperating processes on the sender and receiver. As outlined in [MA00], there are a number of considerations that need to be made when writing a particular BTC metric. This document specifies these items for a BTC methodology that uses cooperating processes on the sender and receiver.

Readers are assumed to be familiar with [MA00] and [RFC2581]. The terminology used to describe the congestion control algorithms in this document is taken from [RFC2581].

We implemented this methodology in two programs, cap and capd. The

discussion in this document is conducted in terms of these two programs. However, alternate programs can be written that conform to this BTC methodology.

2 Congestion Control Algorithm Specifications

As specified in section 2 of [MA00], each BTC document must tightly specify several details of the congestion control algorithms that are not tightly specified in [RFC2581]. The following is the specification of those details for the sender's behavior in the defined methodology:

* Window Increase During Congestion Avoidance: During congestion avoidance, cap counts the number of packets that are acknowledged (ACKed) by the cumulative acknowledgment, denoted SA. When SA becomes greater than or equal to the current value of the congestion window (cwnd), SA is decreased by the current value of cwnd and cwnd is increased by 1 segment unless the increase is not possible due to the configured advertised window size.

* When To Enter Congestion Avoidance. [RFC2581] allows TCP to use either slow start or congestion avoidance when cwnd equals ssthresh. Cap uses congestion avoidance.

* Cap uses a segment size of 1500 bytes by default. The segment size can be changed using a command-line option. Cap does not use Path MTU Discovery [RFC1191].

* By default, cap assumes 40 bytes of header are prepended to each segment (default TCP and IP headers). When using timestamps [RFC1323] the header size is increased by 12 bytes. Additionally, when using selective acknowledgments (SACKs) [RFC2018] the header size on returning ACKs depends on the number of SACK blocks being returned (per [RFC2018]).

* The algorithm for calculating the retransmission timeout (RTO) is similar to the algorithm outlined in [RFC2988]. The algorithm is fully specified in section 3.

[MA00] recommends each BTC take a number of ancillary metrics, in addition to a simple BTC measurement. Cap does not perform any of these ancillary metrics, but can produce a segment trace which may be used to derive these metrics via post-processing.

3 Calculating the Retransmission Timeout

The RTO used in this BTC methodology is generally outlined in [RFC2988]. The following is a sketch of the initial conditions, as well as a discussion of how our estimator differs from the one outlined in [RFC2988]. The reader is assumed familiar with [RFC2988].

Cap takes high-precision round-trip time (RTT) measurements and converts these into a retransmission timeout (RTO) based on a clock with a given granularity. The RTO is initialized as follows:

* The default clock granularity, G, is 500 ms. However, the clock granularity may be changed via a command-line option.

* The initial RTO in clock ticks is: (int)(3 seconds / G).

* When cap is started, the first heartbeat is determined by generating a uniform random number between 0-G and subtracting the obtained value from the current time. The time of the first heartbeat is denoted HB_FIRST.

* We define bounds on the RTO, as follows:

```
MIN_TICKS = ceil (1.0 / G)
MAX_TICKS = ceil (64.0 / G)
```

The RTO is calculated based on RTT measurements. We derive RTT measurements in one of two ways. When the timestamp option is enabled by the user, we use the timestamps in incoming ACKs to take RTT measurements. Otherwise, we time one segment and its corresponding ACK per RTT, as outlined in [RFC2988]. We update the SRTT and RTTVAR upon taking each sample as defined in [RFC2988].

The timer is armed in the situations outlined in [RFC2988]. Each time we are the timer the following algorithm is used to convert the fine-grained SRTT and RTTVAR values to a course-grained RTO estimate.

```
now = get_current_time;
if (!SRTT)
    ticks = 3.0 / G
else
    rto = SRTT + (4 * RTTVAR)
    ticks = ceil (rto / G)
ticks *= BACKOFF
if (ticks < MIN_TICKS)
    ticks = MIN_TICKS
else if (ticks > MAX_TICKS)
    ticks = MAX_TICKS
so_far = now - HB_FIRST;
so_far_ticks = (int)(so_far / G)
gone = so_far - (so_far_ticks * G)
partial = G - gone;
full = (ticks - 1) * G
real_rto = full + partial
arm_timer (real_rto)
```

4 Receiver Specification

The receiving process, capd, sends ``ACKs'', UDP datagrams, to the sender with the following properties.

* Each ACK contains a cumulative sequence number, as done in TCP.

* The default size of an ACK is 40 bytes.

* In the case when capd echoes the timestamp sent by cap the ACK consists of 52 bytes.

* By default ACKs are sent in response to every incoming data segment.

* The user may enable the use of delayed ACKs [RFC1122,RFC2581] via a command-line option.

5 Conclusion

This document specifies a BTC methodology involving two processes based on the framework outlined in [MA00]. This methodology has been shown to accurately gauge the BTC of a given network path over various network conditions [All01].

6 Security Considerations

The BTC methodology outlined in this document does not pose security problems beyond those expressed in the BTC framework document [MA00].

Acknowledgments

Thanks to Vern Paxson for encouraging the development of cap.

References

[All01] Mark Allman Measuring End-to-End Bulk Transfer Capacity, February 2001. Under review.

[MA00] Matt Mathis, Mark Allman. A Framework for Defining Empirical Bulk Transfer Capacity Metrics, February 2001. Internet-Draft draft-ietf-ippm-btc-framework-05.txt (work in progress).

[RFC1191] Jeff Mogul, Steve Deering, "Path MTU Discovery", RFC 1191, November 1990.

[RFC1323] Van Jacobson, Robert Braden, David Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2018] Matt Mathis, Jamshid Mahdavi, Sally Floyd, Allyn Romanow, "TCP Selective Acknowledgment Options", RFC 2018, 1996.

[RFC2581] Mark Allman, Vern Paxson, W. Richard Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2988] Vern Paxson, Mark Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

Author's Address:

Mark Allman
BBN Technologies/NASA Glenn Research Center
Lewis Field
21000 Brookpark Rd. MS 54-5
Cleveland, OH 44135
Phone: 216-433-6586
Fax: 216-433-8705
mallman@bbn.com
http://roland.grc.nasa.gov/~mallman