Low Extra Delay Background Transport

draft-ietf-ledbat-congestion-01a.txt

Stanislav Shalunov ⟨shalunov@bittorrent.com⟩

IETF 77, Anaheim
LEDBAT WG
March 22, 2010

Document status

- draft-ietf-ledbat-00: in the repository
- draft-ietf-ledbat-01: text exists, missed the submission deadline
- since submission deadline was missed, I'll describe the doc changes in detail here

Known issues last time

- framing (out of scope)
- late comers
- fairness
- parameter values
- go over pseudocode

- addressed all

Framing

- Section 4.16
- out of scope
  - implications for wire format
    - carry timestamp
    - carry delay measurement
  - applicable to existing transport protocols
  - application is out of scope

Late comers

- Section 4.17.1
- Once early flow yields, late flow measures the true base delay
- Often faster because of gaps between packet bursts

Fairness

- Random reshuffling
- Normally (more than) enough noise
- Added explicit random input
  - Section 4.17 with motivation
  - pseudocode reflects
  - parameter SHOULD be 0

Parameter values

- Picked specific values for all parameters
- TARGET and GAIN, use MUST
- For the rest, use SHOULD
  - Give ranges as MUSTs

Go over pseudocode

- Added initialization where appropriate
- Is it clear?
- (Multiple implementors were able to follow)

Covers all issues (?)

- Covered what was raised on the mailing list
- Covered what was raised at last meeting
- Is the coverage adequate?
- Is this all?

QUESTIONS?

(supplemental slides follow)

4.16. LEDBAT framing and wire format

The actual framing and wire format of the protocol(s) using the LEDBAT congestion control mechanism is outside of scope of this document, which only describes the congestion control part.

There is an implication of the need to use one-way delay from the sender to the receiver in the sender. An obvious way to support this is to use a framing that timestamps packets at the sender and conveys the measured one-way delay back to the sender in ack packets. This is the method we'll keep in mind for the purposes of exposition. Other methods are possible and valid.

The protocols to which this congestion control mechanism is applicable, with possible appropriate extensions, are TCP, SCTP, DCCP, etc. It is not a goal of this document to cover such applications. The mechanism can also be used with proprietary transport protocols, e.g., those built over UDP for P2P applications.

4.17. Fairness between LEDBAT flows

The design goals of LEDBAT center around the aggregate behavior of LEDBAT flows when they compete with standard TCP. It is also interesting how LEDBAT flows share bottleneck bandwidth when they only compete between themselves.

LEDBAT as described so far lacks a mechanism specifically designed to equalize utilization between these flows. The observed behavior of existing implementations indicates that a rough equalization, in fact, does occur.

The delay measurements used as control inputs by LEDBAT contain some amount of noise and errors. The linear controller converts this input noise into the same amount of output noise. The effect that this has is that the uncorrelated component of the noise between flows serves to randomly shuffle some amount of bandwidth between flows. The amount shuffled during each RTT is proportional to the noise divided by the target delay. The random-walk trajectory of bandwidth utilized by each of the flows over time tends to the fair share. The timescales on which the rates become comparable are proportional to the target delay multiplied by the RTT and divided by the noise.

In complex real-life systems, the main concern is usually the reduction of the amount of noise, which is copious if not eliminated. In some circumstances, however, the measurements might be "too good" -- since the equalization timescale is inversely proportional to noise, perfect measurements would result in lack of convergence.

Under these circumstances, it may be beneficial to introduce some artificial randomness into the inputs (or, equivalently, outputs) of the controller. Note that most systems should not require this and should be primarily concerned with reducing, not adding, noise.

4.17.1. Late comers

With delay-based congestion control systems, there's a concern about the ability of late comers to measure the base delay correctly. Suppose a LEDBAT flow saturates a bottleneck; another LEDBAT flow starts and proceeds to measure the base delay and the current delay and to estimate the queuing delay. If the bottleneck always contains target delay worth of packets, the second flow would see the bottleneck as empty start building a second target delay worth of queue on top of the existing queue. The concern ("late comers' advantage") is that the initial flow would now back off because it sees the real delay and the late comer would use the whole capacity.

However, once the initial flow yields, the late comer immediately measures the true base delay and the two flows operate from the same (correct) inputs.

Additionally, in practice this concern is further alleviated by the burstiness of network traffic: all that's needed to measure the base delay is one small gap. These gaps can occur for a variety of reasons: the OS may delay the scheduling of the sending process until a time slice ends, the sending computer might be unusually busy for some number of milliseconds or tens of milliseconds, etc. If such a gap occurs while the late comer is starting, base delay is immediately correctly measured. With small number of flows, this appears to be the main mechanism of regulating the late comers' advantage.

```
on initialization:
  set all NOISE_FILTER delays used by current_delay() to +infinity
  set all BASE_HISTORY delays used by base_delay() to +infinity
  last_rollover = -infinity # More than a minute in the past.

on acknowledgement:
  delay = acknowledgement.delay
  update_base_delay(delay)
  update_current_delay(delay)
queuing_delay = current_delay() - base_delay()
off_target = TARGET - queuing_delay + random_input()
cwnd += GAIN * off_target / cwnd
# flight_size() is the amount of currently not acked data.
max_allowed_cwnd = ALLOWED_INCREASE + TETHER*flight_size()
cwnd = min(cwnd, max_allowed_cwnd)
```

```
random_input()
  # random() is a PRNG between 0.0 and 1.0
  # NB: RANDOMNESS_AMOUNT is normally 0
  RANDOMNESS_AMOUNT * TARGET * ((random() - 0.5)*2)

update_current_delay(delay)
  # Maintain a list of NOISE_FILTER last delays observed.
  forget the earliest of NOISE_FILTER current_delays
  add delay to the end of current_delays

current_delay()
  min(the NOISE_FILTER delays stored by update_current_delay)
```

```
update_base_delay(delay)
  # Maintain BASE_HISTORY min delays. Each represents a minute.
  if round_to_minute(now) != round_to_minute(last_rollover)
    last_rollover = now
    forget the earliest of base delays
    add delay to the end of base_delays
  else
    last of base_delays = min(last of base_delays, delay)

base_delay()
  min(the BASE_HISTORY min delays stored by update_base_delay)
```

TARGET parameter MUST be set to 25 milliseconds and GAIN MUST be set so that max rampup rate is the same as for TCP. BASE_HISTORY SHOULD be 2; it MUST be no less than 2 and SHOULD NOT be more than 10. NOISE_FILTER SHOULD be 1; it MAY be tuned so that it is at least 1 and no more than cwnd/2. ALLOWED_INCREASE SHOULD BE 1 packet; it MUST be at least 1 packet and SHOULD NOT be more than 3 packets. TETHER SHOULD be 1.5; it MUST be greater than 1. RANDONMESS_AMOUNT SHOULD be 0; it MUST be between 0 and 0.1 inclusively.