The TCP *Minimum* RTO Revisited

Ioannis Psaras and Vassilis Tsaoussidis

`(ipsaras, vtsaousi)@ee.duth.gr`

COMputer NETworks (COMNET) Group

Democritus University of Thrace, Xanthi, Greece

`http://comnet.ee.duth.gr/`

Introductory Notes

According to RFC 2988 (Computing TCP’s Retransmission Timer):

1. $RTO = SRTT + 4 \times RTTVAR,$
2. $RTO \geq 1\ second$ (i.e., *Minimum* RTO)

The *Minimum* RTO protects TCP against spurious timeouts caused by:

1. coarse-grained clocks (500ms for most OSs *at that time, i.e., Nov. 2000*)
2. the Delayed Acknowledgments (usually set to 200 ms), RFC1122

Our Contribution

- We re-examine the two reasons for the conservative 1-second Minimum TCP-RTO:
  1. the OS clock granularity, and
  2. the Delayed ACKs.
- We find that reason 1 is canceled in modern OSs,
- We carefully design a mechanism to deal with reason 2.
- We show (through simulations) that in next generation's high-speed, wireless-access networks, TCP-RTO should not be limited by a *fixed, conservative* lower bound.

Cost Function

We define a *Cost Function* to capture the impact of the Minimum RTO to TCP's performance:

$$C(f) = \frac{RTO_{min}}{RTO_{current}}$$

- If $C(f) \leq 1$, the Minimum RTO adds no extra waiting time.
- Otherwise, the Minimum RTO will negatively impact TCP Throughput.

Clock Granularity (1/5): Motivation

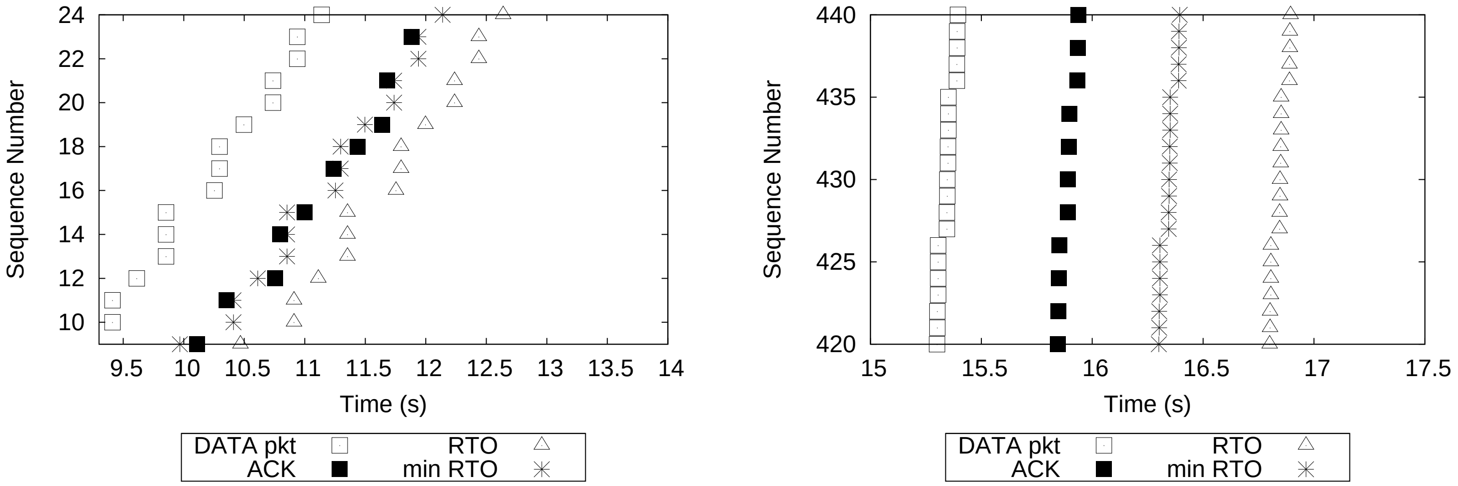
We simulate a coarse-grained flow (i.e., G=500ms) over a 500ms Round-Trip Propagation Delay (RTPD) path, to observe:

1. the rationale behind the conservative 1-second Minimum RTO setting, and
2. the impact of the Minimum RTO value relatively with the actual TCP-RTO value.

Clock Granularity (2/5): Observations

We observe that:

1. $C(f) < 1$ (no negative impact on TCP Throughput),
2. the Minimum RTO is only needed as a safety margin.



(a) G = 500ms, RTPD = 500ms (b) G = 500ms, RTPD = 6ms

Clock Granularity (3/5): OS Details

Table 1: Details on Modern OSs

OS	Clock Granularity	Delayed ACK
Windows	15-16ms	200ms
Solaris	10ms	50-100ms
Linux	$\leq$ 25ms	Dynamically Set

- We repeat the above experiment using, this time, a finer-grained clock of 10ms.

Clock Granularity (4/5): Impact

- $C(f) \approx \frac{RTO_{min}}{T(ACK\ Arr)} \leq \frac{RTO_{min}}{RCG+RTPD+QD} \approx 62.5$

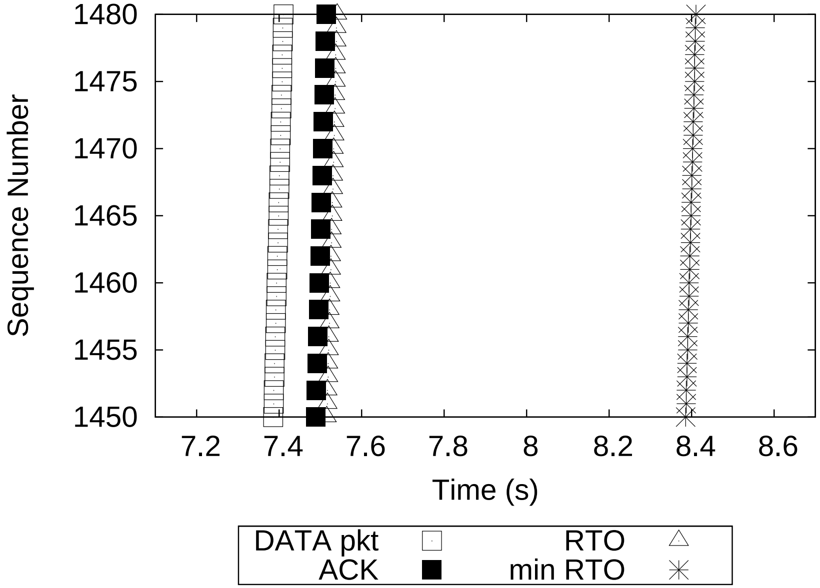


Figure 2: G = 10ms, RTPD = 6ms

Clock Granularity (5/5): Conclusions

We conclude that:

1. the clock granularity should not be a matter of concern for the setting of the Minimum RTO, and
2. the conservative 1-second Minimum RTO will have major impact on TCP's performance, in case of packet losses.

Delayed ACKs (1/6): Notes

- TCP sends `D` back-to-back packets, according to RFC 2581: `D = snd.una + min(cwnd, rwnd) - snd.nxt`.
- TCP does not know the application’s sending pattern.
- Only the ACK of the last packet of the "train" of back-to-back packets *may* be delayed.
- Every 2^{nd} packet will always be ACKed.

Delayed ACKs (2/6): An Example

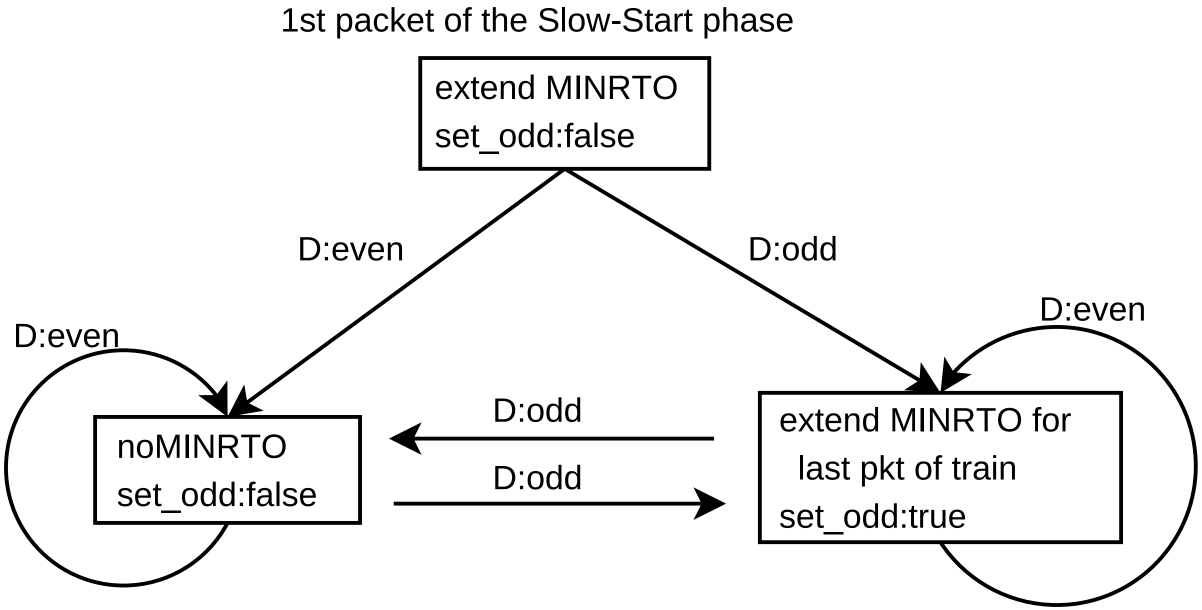
- At time t_0 all previously transmitted packets are already ACKed.
- `D = 4:` (or generally `D`: even)
  - The client will ACK the 2^{nd} and 4^{th} packets.
  - No Delayed ACKs $\Rightarrow$ no need for extended Minimum RTO.
- `D = 3:` (or generally `D`: odd)
  - Client will ACK the 2^{nd} packet and will trigger the DelACK timer for the 3^{rd} packet.
  - The 3^{rd} packet's ACK *may* be Delayed $\Rightarrow$ extend the Minimum RTO, for the 3^{rd} packet *only*.

Delayed ACKs (3/6): Algorithm States

The proposed mechanism operates in one of the following States:

- State 1: "noMINRTO". Do *not* apply extended Minimum RTO to any outgoing packet (i.e., the receiver will always ACK the last packet of the back-to-back train of packets); set `set_odd` to false.
- State 2: "extended MINRTO". Apply extended Minimum RTO to the last packet of the next train of back-to-back packets; set `set_odd` to true.

Delayed ACKs (4/6): State Diagram



1st packet of the Slow-Start phase

extend MINRTO
set_odd:false

D:even

D:odd

D:even

D:even

noMINRTO
set_odd:false

D:odd

D:odd

extend MINRTO for
last pkt of train
set_odd:true

Figure 3: State Diagram

Delayed ACKs (5/6): Example

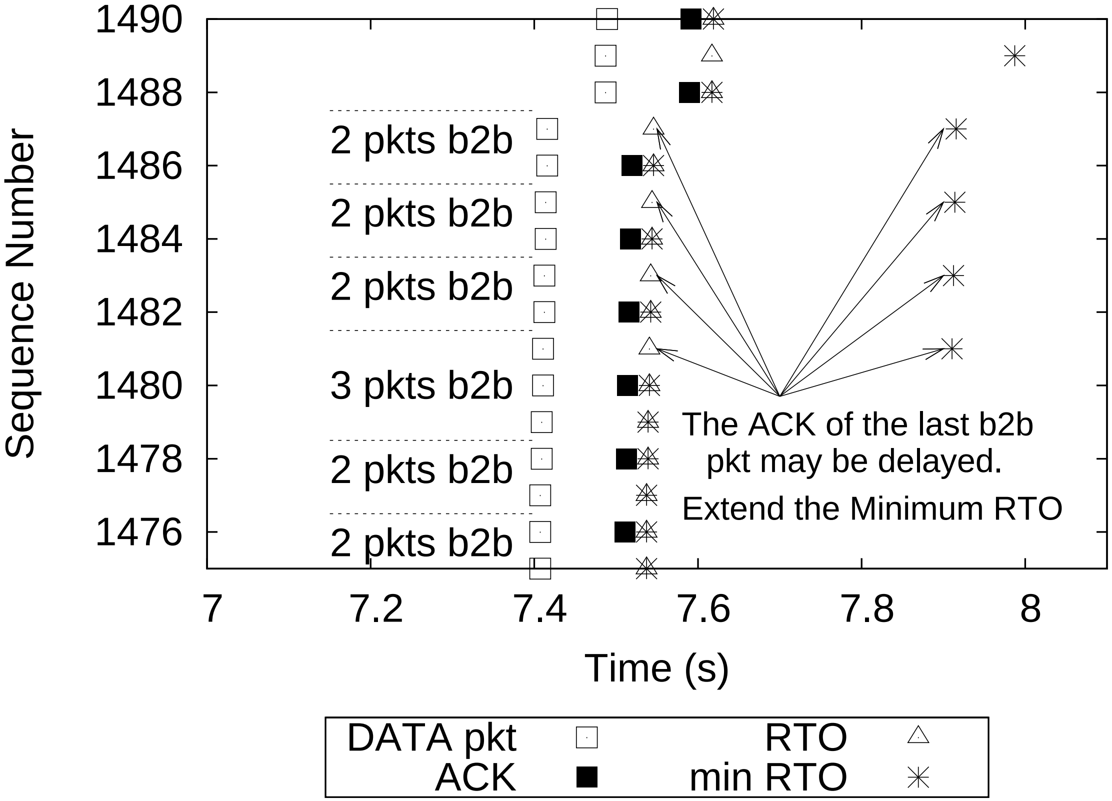


Figure 4: Modeling ACKs Arrival

Delayed ACKs (6/6): Impact

$$RTO_{min} = \begin{cases} R\ ms, & \text{for the last pkt } \textit{if } \texttt{set\_odd} = 1, \\ RTO_{cur}, & \text{otherwise,} \end{cases}$$

where R is a fixed, extended value for the Minimum RTO.

$$C(f) = \begin{cases} \frac{R\ ms}{RTO_{cur}}, & \text{for the last pkt } \textit{if } \texttt{set\_odd} = 1, \\ 1, & \text{otherwise.} \end{cases}$$

Performance Evaluation (1/2)

- TCP version: Reno
- SACK: enabled
- Timestamps: enabled
- Spurious response: enabled
- Delayed ACK Timer: 200ms
- Granularity: 10ms
- Buffers use RED, Buffer size = *BDP*
- We measure the System Goodput:

  $Goodput = \frac{Original\_Data}{Connection\_time}$

Performance Evaluation (2/2)

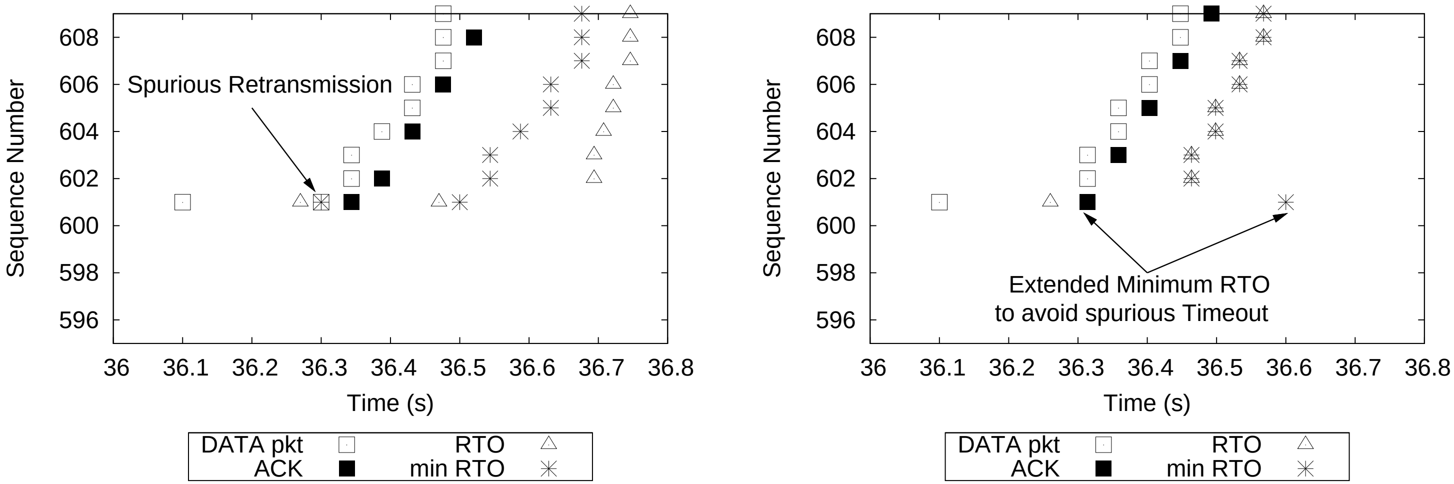
We compare the proposed algorithm with three different TCP implementations:

1. Linux TCP: Minimum RTO = 200ms
2. Solaris TCP: Minimum RTO = 400ms
3. IETF Proposal (RFC 2988): Minimum RTO = 1s (probably Windows TCP)

Results (1/4): The Need for a Standard Mechanism (1/2)

If
$Srv's\ Min\ RTO < RTPD + QD + Clnt's\ DelACK\ Timer$
and
$Minimum\ RTO > RTO_{cur}$,
then the TCP server will spuriously timeout *every time* an ACK is delayed *and* `D` = 1.

Results (2/4): The Need for a Standard Mechanism (2/2)



(a) Linux Server - 200ms Delayed ACK Client (e.g., Windows client)

(b) Modified Linux Server - 200ms Delayed ACK Client (e.g., Windows client)

Figure 5: The Need for a Standard Mechanism

Results (3/4): Long FTP Flows (1/2)

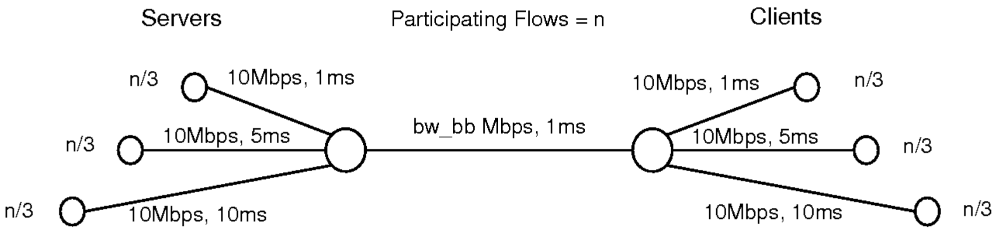
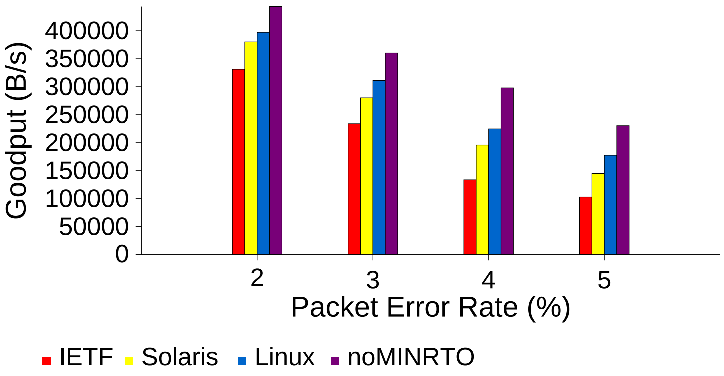


Figure 6: Simulation Topology

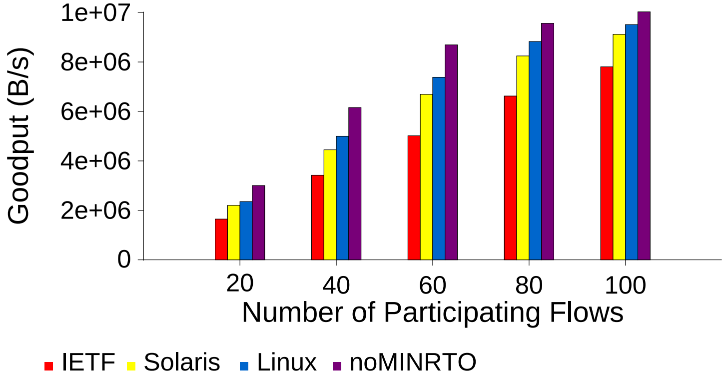
Table 2: Experiment Details

	PER	TCP Flows	bw_bb
Fig. 7(a)	see Fig.	3	6 Mbps
Fig. 7(b)	3%	see Fig.	100 Mbps
Fig. 7(c)	3%	500	see Fig.

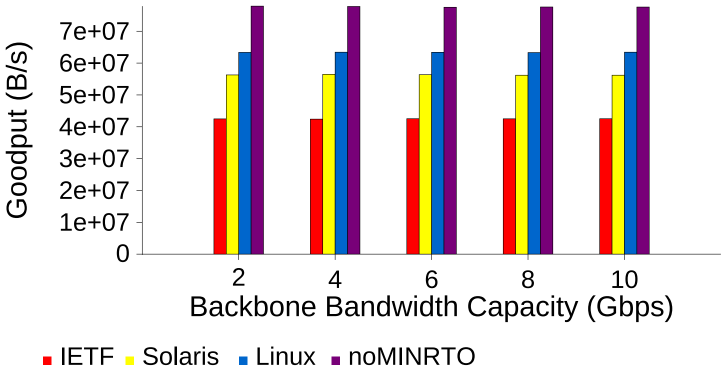
Results (4/4): Long FTP Flows (2/2)



(a) Increasing PER



(b) Increasing TCP Contention



(c) Increasing Bandwidth Capacity

Conclusions

1. The conservative 1-second Minimum RTO setting causes severe TCP performance degradation.
2. The Minimum RTO setting is not needed, since:
   - modern OSs use fine-grained clocks, and
   - the proposed algorithm deals with the Delayed ACK response.
3. The proposed algorithm:
   - may improve TCP performance up to 50%,
   - effectively avoids spurious timeouts, and
   - overcomes communication inconsistencies, caused by the absense of official instructions regarding the Minimum RTO setting.