TCP-NCL: A serialized-timer approach for enhancing TCP over heterogeneous wired/wireless networks

Ka-Cheong Leung *, Chengdi Lai, Huiyi Ding

Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, China

A R T I C L E   I N F O

Keywords:
Congestion control
Flow control
Wireless networks

A B S T R A C T

In heterogeneous wired/wireless networks, TCP performs unsatisfactorily since packet reordering and non-congestive losses may be falsely interpreted as congestive losses. This causes TCP to trigger fast retransmission and fast recovery spuriously, leading to under-utilization of available network resources. In this paper, we have developed a smart TCP sender (STS) model to differentiate congestive issues from the non-congestive ones for constructing more reliable signals of packet loss and network congestion over general error-prone channels. Two serialized timers are employed so that their expirations offer two separate signals of a packet loss and network congestion. The first timer is started when a packet is first injected into the network, and it will be cancelled if the acknowledgement is received. Otherwise, the packet will be retransmitted and the second timer is started upon the expiration of the first timer. The STS model is constructed based on the concept of minimizing expected cost and optimal setting of expiration periods for timer is determined. We have devised a novel TCP variant, known as TCP for non-congestive loss (TCP-NCL), as a practical approximation of the STS model. TCP-NCL can thus serve as a unified solution for effective congestion control, sequencing control, and loss recovery over wireless networks. The deployment of TCP-NCL requires modifications to sender-side TCP only, thereby facilitating possible future wide deployment. Our simulation studies show that TCP-NCL is robust against packet reordering as well as non-congestive packet loss while maintaining good responsiveness against congestive loss.

1. Introduction

Transmission Control Protocol (TCP) [1–3] is the most popular transport layer protocol in the Internet, featuring reliable unicast transmission and end-to-end congestion control. In fulfilling the former feature, signals of packet loss have been used for triggering retransmissions of lost data packets or segments to ensure the eventual delivery of every data byte. In fulfilling the latter, signals of network overload have been used for triggering congestion response, which reduces the size of congestion window (cwnd) and thus avoids further overload of a congested network. However, identifying such signals and utilizing them to perform effective congestion control and loss recovery are very challenging.

Many popular TCP variants, such as TCP Reno [1,4], use the same set of signals for indicating packet loss and network overload. Two types of signals are used, namely, retransmission timeout (RTO) and triple duplicate acknowledgments (ACKs). A retransmission timer is started when a data packet is first injected into the network, and will timeout if an ACK for the packet is still missing when the timer expires. Upon the occurrence of an RTO, all the outstanding packets will be retransmitted. At the same time, the network is deemed severely congested and cwnd will be reset to one packet.

A TCP receiver expects all the data packets received to be consecutively ordered. It will send back a duplicate ACK to its corresponding TCP sender for each received packet failing the expectation. At the sender side, when the number of duplicate ACKs reaches a certain threshold value, known as dupthresh, fast retransmit and fast recovery will be activated, retransmitting the packet expected by the receiver and halving cwnd. Therefore, the arrival of triple duplicate ACKs, a direct signal of out-of-order packet events, is further used as an indication of congestive packet loss.

1.1. Heterogeneous wired/wireless networks

By using the arrival of triple duplicate ACKs for activating packet retransmission and congestion response, the conventional TCP designs rely on the assumptions of a nearly in-order channel of negligible or recoverable transmission error. While the assumptions might hold over traditional wired networks, they are generally violated over modern heterogeneous wired/wireless networks due to the significant level of occurrences of non-congestive packet losses and packet reordering [5].
Packet reordering refers to the disruption of the packet order of a TCP flow. Despite conventional beliefs that packet reordering is a transient or pathological network behaviour, it is in fact persistently observed over modern networks due to the increasing level of parallelism in various network devices [5,6]. For example, a high-speed link between two routers is sometimes realized via multiple parallel physical links. Packets belonging to the same flow may thus traverse different physical paths and arrive at the destination out of order. Recent measurements on TCP traffic shows that, on average, 1% of data packets experience reordering in various networks [7]. In a large portion of the reordering events, packets are reordered by more than dupthresh, leading to spurious fast retransmit and fast recovery by TCP [8].

In wireless networks, link-layer retransmissions (LLRTX) can constitute an additional cause of packet reordering. Ideally, wireless networks like the IEEE 802.11 and the third generation (3G) networks preserve packet ordering when performing LLRTX. However, this has to be attained by locally buffering subsequent packets (at either the receiving end or the sending end of a wireless link) when a packet is transmitted at the link layer, incurring significant extra delay. The delay performance can be best improved by allowing subsequent packets to be forwarded to the next hop when a packet is pending for its local ACK. In this case, a retransmitted packet will become interspersed with its later packets, leading to the occurrence of packet reordering.

Hence, in heterogeneous wired/wireless networks, triple duplicate ACKs amount to fairly poor signals of packet loss. The conventional TCP designs use the same signals for inferring packet loss and network overload. Thus, they tend to falsely trigger packet retransmission and reduce cwnd from time to time, injecting duplicated bytes into the network and keeping cwnd unnecessarily small. Consequently, the available network resources are wasted and under-utilized.

1.2. Our contributions

The focus of this work, first described in [9], and [10], is to develop a unified solution for performing effective congestion control, sequencing control, and loss recovery over communication networks with general, error-prone transmission channels, most notably heterogeneous wired/wireless networks. Our proposed novel TCP variant, known as TCP for non-congestive loss (TCP-NCL), has been devised on the following basic principles:

Risk minimization: There are always high risks associated with both activating congestion response and delaying congestion response. We develop an idealized TCP sender model, known as smart TCP sender (STS). STS employs two serialized timers so that the expirations offer two separate signals of packet loss and network congestion. The aforementioned risks are quantified and the timer expiration periods are optimized for minimizing the quantified risks. We further devise TCP-NCL as a practical approximation of STS. Therefore, TCP-NCL is built on the basis of rational risk adversity rather than ad-hoc protocol tuning.

Minimal assumption on network functionality: The end-to-end design principle of the Internet [11,12] suggests that the task of reliable, in-order packet delivery can be best fulfilled by end systems rather than the network. Moreover, keeping network functionality minimal ensures that the Internet is open to various kinds of innovative applications. To this effect, both the STS model and TCP-NCL make the minimal assumptions on the packet delivery guarantee by the underlying network. They have been devised to work well in the presence of significant packet reordering and/or non-congestive loss. Not only is this crucial to ensuring the robust performance of TCP over heterogeneous wired/wireless networks, but it also helps simplify the functionality guarantee by the network.

Backward compatibility: Specifically, (1) TCP-NCL can work well given the limited support from present network environment, and (2) TCP-NCL can share bandwidth fairly with other widely deployed TCP variants. To this effect, we deliberately confine our design space to the scope of loss-based TCP, which uses congestive loss and delay as signals of network congestion concurrently.

Ease of deployment: The deployment of TCP-NCL requires modifications to sender-side TCP only. This mainly involves the maintenance of a few unused bits in the existing data structure for TCP and a few extra variables, which is performed upon the ACK arrivals. TCP-NCL incurs no additional requirement on the receiver-side TCP and packet header.

We have performed extensive simulation experiments to examine the performance of TCP-NCL and compare it with several well-known TCP variants in the literature using Network Simulator (ns) Version 2.29 [13]. Our results show that TCP-NCL attains significant performance improvement over general, error-prone communication networks, thereby demonstrating robustness against packet reordering and non-congestive packet loss. At the same time, multiple competing TCP connections, with some employing TCP-NCL and some employing SACK TCP, the popular standardized TCP variant, do share bandwidth fairly over conventional wired networks. TCP-NCL is therefore responsive against congestive loss, and TCP-friendly.

1.3. Organization of the paper

This paper is organized as follows. Section 2 discusses existing solutions for TCP packet reordering and non-congestive loss. Section 3 develops the smart TCP sender (STS) model, explaining the motivation behind our serialized-timer structure and modelling the optimal timer expiration periods. Section 4 presents TCP-NCL. Section 5 examines the performance of TCP-NCL and compares it with some popular TCP variants. Section 6 concludes and discusses some possible extensions of our work.

2. Related work

In this section, we summarize the existing work for adapting TCP to perform congestion control and loss recovery over general, error-prone transmission channels in wired/wireless networks. TCP variants are classified as loss-based TCP and delay-based TCP.

2.1. Loss-based TCP

AVG, DEL, EWMA, and INC in [14], RR-TCP [15], TCP-DCR [16], and TCP-PR [17] abandon the use of the fixed triple duplicate ACKs as a signal of congestive packet loss. Instead, they proactively postpone a packet retransmission until a corresponding timer expires or when the number of duplicate ACKs received reaches an adaptively evolved threshold value. This is considered a more reliable signal of packet loss over reordered channels.

DSACK TCP [18], TCP-DOOR [19], and TCP-Eifel [20] are designed with the premise that triple duplicate ACKs may be unreliable indications of congestive loss. Their approaches differ, however, in that they try to detect spurious retransmissions after activating fast retransmission and fast recovery upon the arrival of triple duplicate ACKs. Upon successful detection, cwnd will be restored to its size before fast recovery.

However, the above TCP variants are incapable of differentiating between congestive and non-congestive losses, which may be generally unable to recover cwnd unnecessarily reduced due to non-congestive losses. Several congestion distinguishable variants are designed.

JTCP [21], TCP Veno [22], and TCP-Westwood (TCPW) [23] focus on differentiating congestive and non-congestive losses by using the estimated network load. Upon the arrival of triple duplicate ACKs, they retransmit the packet being inferred lost and, instead of halving cwnd, reduce cwnd based on estimated network load. Yet, the inherent assumption of nearly in-order channel obviously limits the applicability of these variants over networks where packet reordering is common.

TCP-Probing [24] applies a different approach for differentiating congestive and non-congestive losses upon inferring packet loss from
the arrival of triple duplicate ACKs. After cwnd is halved in response to the inferred packet loss, probing data packets are injected into the network. The inferred packet loss is categorized as non-congestive if RTT of the first two acknowledged probing packets are smaller than the best RTT, or the minimum of the measured RTTs during the TCP session. However, TCP-Probing will trigger frequent false fast retransmissions due to persistent packet reordering. Furthermore, the stored best RTT may fail to capture network changes. Specifically, when a path change or handoff occurs within a TCP session and increases the minimum attainable RTT, the stored best RTT will not be updated. Consequently, it may become impossible for the probe packets to be acknowledged within the best RTT even under light network load.

The existing unified solution for performing effective congestion control, sequencing control, and loss recovery requires information and/or modifications of the network protocol stack beyond the transport layer. ATCP [25] introduces an ATCP layer between TCP and IP. The new layer switches TCP among various pre-defined states in accordance with the network condition, trying to avoid spurious packet retransmission and congestion response.

2.2. Delay-based TCP

Vegas [26], mVegas [27], and weighted Vegas (wVegas) [28] choose queuing delay as a network congestion signal. Vegas measures packet queuing delay for estimation of link congestion and backlog packets in link queues. mVegas and wVegas extend Vegas to the multipath cases. The dynamics of queuing delay has the right scaling with respect to network capacity, according to the commonly used ordinary differential equation model of TCP, which helps maintain stability as a network scales up in capacity. However, it may be affected by an unfairness problem due to an inaccurate measure of propagation delay.

FAST TCP [29] also chooses queuing delay as the congestion indication to adapt to high bandwidth-delay product networks. It uses the minimum RTT to detect the network congestion to grab the network bandwidth rapidly. Generalized FAST TCP [30] is proposed as an enhanced FAST TCP to achieve the proportional fairness based on the virtual link price. FAST TCP aims at high throughput and stable transmission in large bandwidth networks. However, there are some open issues on it, such as propagation delay measurement and parameter tuning.

TCP BBR [31] has gained wide-spread attention recently. Unlike other algorithms, TCP BBR is designed to operate without creating packet loss or filling buffers. It can perform well in Google’s B4 network, which is a high-speed wide-area network (WAN). However, Our simulation results (in Section 5) have shown that the performance of TCP BBR may deteriorate significantly in the presence of non-congestive losses in wireless networks. One possible reason is that the sending way for TCP BBR is often limited by severe acknowledgement (ACK) aggregation. TCP BBR calculates the congestion window size by measuring the maximum bottleneck bandwidth and the minimum round-trip propagation time and sends packets at a paced rate. However, the maximum bottleneck bandwidth and the minimum round-trip propagation time cannot be measured at the same time, and they should be measured by turns. If packets are ACKed in bursts after long delays, TCP BBR will make estimating mistakes for congestion window, and the bottleneck may become idle for long periods. However, loss-based congestion control does not have this problem because it continues to increase the congestion window until the buffer is full.

In summary, there is a lack of a unified solution for performing effective congestion control, sequencing control, and loss recovery at the transport layer with reliable signals of network overload and packet loss for performing congestion control and loss recovery over general, error-prone transmission channels. Such unified solution is highly desirable, as it adapts TCP to a wide range of wireless and, in some cases, wired transmission channels at a minimal deployment cost.

3. Smart TCP sender model

Our unified solution is motivated by the following two observations: Nature of signals: Over general communication channels, there are two prevalent approaches for inferring the occurrence of a packet loss based on: (1) the number of consecutively received duplicate ACKs reaching an adaptively evolved threshold value or, (2) a timer expiration. However, one major concern with the former approach is that a TCP sender may not be able to accumulate enough duplicate ACKs to perform loss recovery because there are insufficient numbers of outstanding packets. Special measures, like limited retransmit [32], would have to be incorporated in order to alleviate the problem by sending extra data packets upon the arrivals of duplicate ACKs. By contrast, a more elegant and robust approach for inferring packet loss is to measure the time elapsed since a packet transmission and refer to the past recent history of RTT.

Separation of signals: Over error-prone channels, signals of packet loss alone can only serve as indirect signals of network congestion. It is necessary to gather additional information to confirm whether an inferred loss can be categorized as a congestive loss.

In place of the conventional approach of activating fast retransmit and fast recovery simultaneously, we propose to postpone the decision on a congestion response for a short time period after a packet retransmission. The motivation behind doing so is two-fold. First, information inferred by the events which happen after a packet retransmission can be incorporated to decide whether a congestive loss has occurred. Such information is a real-time reflection of the network load. Specifically, if an ACK for a packet lags behind a retransmission of that packet by a short duration, it should be treated as a signal of no network congestion. The ACK packet may be for the originally transmitted packet or the retransmitted packet. In the former case, false retransmission, which can hardly be fully eliminated in reality, is detected, thereby eliminating the need for a congestion response. In the latter case, a fairly short round-trip time and thus a lightly loaded network can be inferred.

Second, it is more “affordable” for TCP to trigger a false fast retransmit than a false congestion response. In the event of a false fast retransmit, there is at most one full-sized packet being retransmitted spuriously. However, a significant portion of the available bandwidth will be left unused in the event of a falsely activated congestion response. For TCP variants which activate fast retransmit and fast recovery simultaneously, both measures may be deferred to avoid the penalty of a false congestion response until it is highly likely that a congestive loss has occurred. However, over general error-prone reordering channels, such deferment on fast retransmit may result in an expensive RTO. For example, TCP-DCR postpones fast retransmit by one RTT upon receiving the first duplicate ACK, so that a packet is retransmitted, if needed, about two RTTs after its first transmission. By advancing a packet retransmission before the activation of a congestion response, the risk of triggering an RTO can be more effectively reduced while avoiding the severe performance penalty due to a false congestion response. Nevertheless, while false retransmission can be tolerated with our proposal, it would not be triggered excessively to avoid a significant wastage of the limited bandwidth.

Based on the above observations, we design our smart TCP sender (STS) model, as illustrated in Fig. 1. A new retransmission decision timer \( RD \), is started whenever a new packet \( P \) is injected into the network. If ACK, is received by the TCP sender before \( RD \) expires, \( RD \) will be cancelled. Otherwise, \( P \) will be retransmitted and a congestion response decision timer \( CD \), will be started.

---

1 This timer is different from the conventional retransmission timer, which triggers a packet retransmission and activates the slow start algorithm when it expires. TCP-PR is an exception since it uses a modified retransmission timer so that fast retransmit and fast recovery are activated upon the timer expiration.
$CD_i$ will be cancelled if $ACK_i$ arrives before it expires. Otherwise, the congestion control mechanisms will be activated upon the expiration of $CD_i$. Therefore, the installation of $CD_i$ allows the TCP sender extra time after the packet retransmission to decide whether congestion control shall be activated. $ACK_i$ arriving before the expiration of $CD_i$ shall be treated as an indication of no network congestion. Thus, this eliminates the need for activating any congestion control measures.

The expiration periods of $RD_i$ and $CD_i$, denoted as $\tau_{RD_i}$ and $\tau_{CD_i}$, are evaluated in Sections 3.1 and 3.2, respectively. To facilitate the subsequent discussion, we denote the time instants when $RD_i$ expires as Time 0, and $t$ time units after $RD_i$ expires as Time $t$. On this basis, we define the notations as shown in Tables 1 and 2.

The following assumptions are made to simplify the discussion:

(A1) The size of congestion window and slow start threshold ($ssthresh$) can be expressed in terms of the number of packets. This assumption has been widely adopted in the literature for the analytical studies of the TCP variants, such as TCP Reno [33], TCP Veno [34], RR-TCP [35], and TCP CUBIC [36].

(A2) $P(\text{PA}_i(t)|\text{CL}_i)$ is monotonically increasing with Time $t$ and $\lim_{t \to \infty} P(\text{PA}_i(t)|\text{CL}_i) > 0$. Given that $\text{PL}_i^L$ is lost due to network congestion, whether Packet $P_i$ is acknowledged or not relies on $\text{PL}_i^L$. By allowing more time (i.e., a larger $t$), the chance of $P_i$ being acknowledged does not drop. Furthermore, the chance should be non-zero with a sufficiently large $t$. 

\begin{table}[h]
\centering
\caption{General notations.}
\begin{tabular}{ll}
\hline
\text{Symbol} & \text{Description} \\
\hline
$\epsilon_f$ & Cost of falsely activating a congestion response \\
$\epsilon_d$ & Cost of excessively delaying a congestion response \\
$F(t)$ & Cumulative distribution function of RTT \\
$\delta_{\text{min}}$ & Minimum attainable RTT \\
$P^i$ & First transmitted Packet $P_i$ \\
$P^r$ & Retransmitted Packet $P_i$ \\
$p_c$ & Congestive loss rate \\
$p_l$ & Loss rate \\
$p_n$ & Non-congestive loss rate \\
w & Average cwnd over a TCP session \\
\hline
\end{tabular}
\end{table}
Table 2
Notations for events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( NL, r )</td>
<td>( P_r ) lost due to network congestion</td>
</tr>
<tr>
<td>( NL, o )</td>
<td>( P_r ) lost due to network congestion</td>
</tr>
<tr>
<td>( NL, r )</td>
<td>( P_o ) experiencing non-congestive lost</td>
</tr>
<tr>
<td>( NL, o )</td>
<td>( P_o ) experiencing non-congestive lost</td>
</tr>
<tr>
<td>( R/TX, l )</td>
<td>Fast retransmission of ( P_l )</td>
</tr>
<tr>
<td>( P_{A}(t) )</td>
<td>Either ( P_r ) or ( P_o ) acknowledged by Time ( t ) after ( R/TX, l )</td>
</tr>
<tr>
<td>( P_{A}(t) )</td>
<td>( P_r ) acknowledged by Time ( t ) after retransmission</td>
</tr>
<tr>
<td>( P_{A}(t) )</td>
<td>( P_o ) acknowledged by Time ( t ) after retransmission</td>
</tr>
<tr>
<td>( P_{U}(t) )</td>
<td>( P_r ) is lost</td>
</tr>
<tr>
<td>( P_{U}(t) )</td>
<td>Both ( P_r ) and ( P_o ) unacknowledged by Time ( t ) after ( R/TX, l )</td>
</tr>
<tr>
<td>( P_{U}(t) )</td>
<td>( P_o ) unacknowledged by Time ( t ) after retransmission</td>
</tr>
<tr>
<td>( TO )</td>
<td>Retransmission timeout</td>
</tr>
</tbody>
</table>

Table 3
Network configurations for the infrastructure-based wireless network.

<table>
<thead>
<tr>
<th>Figure</th>
<th>bw</th>
<th>dl</th>
<th>pe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 6</td>
<td>1</td>
<td>50</td>
<td>0–10%</td>
</tr>
<tr>
<td>Fig. 8(a)</td>
<td>1</td>
<td>10–320</td>
<td>1%</td>
</tr>
<tr>
<td>Fig. 8(b)</td>
<td>1</td>
<td>10–320</td>
<td>3%</td>
</tr>
<tr>
<td>Fig. 8(c)</td>
<td>1</td>
<td>10–320</td>
<td>5%</td>
</tr>
<tr>
<td>Fig. 9(a)</td>
<td>1–6</td>
<td>50</td>
<td>1%</td>
</tr>
<tr>
<td>Fig. 9(b)</td>
<td>1–6</td>
<td>50</td>
<td>3%</td>
</tr>
<tr>
<td>Fig. 9(c)</td>
<td>1–6</td>
<td>50</td>
<td>5%</td>
</tr>
</tbody>
</table>

(A3) What happens to \( P_r \) is independent of what happens to \( P_o \). In particular,

\[
P(NL, r, TO(u) | (P_{U}(t))) = P(NL, r, TO(u)) - P(P_{U}(t))
\]

(3.1. Retransmission decision timer)

\( \tau_{RD} \) determines how long we have to wait before activating packet retransmission. We make two mild assumptions regarding \( \tau_{RD} \), which will facilitate our discussions on the optimal setting of \( \tau_{CD} \). First, when \( P_r \) is lost due to congestion, \( P_r \) should be promptly transmitted so that the remaining time before RTO is sufficient for it to be acknowledged. This ensures that \( cwnd \) will be halved instead of being reset to one at the onset of network overload. In other words,

(A4) RTO will be incurred if and only if \( P_r \) is lost. Analytically, this is:

\[
P(T/O|CL, r) = P(P_{U}(t)) = p_i
\]

Second, the probability of false fast retransmit is \((1-p_i)(1-\max(\tau_{RD}))\).

To avoid frequent premature fast retransmissions, we should have:

(A5) \( F(\tau_{RD}) \approx 1 \)

(3.2. Congestion response decision timer)

The setting of \( \tau_{CD} \) should guarantee that, if ACK \( k \) arrives before the expiration of \( CD, P_r \) does not experience congestive loss. In other words:

\[
P(CL, r, P_{A}(t)) = 0
\]

for all \( t \in [0, \tau_{CD}] \). In this way, STS does not misinterpret any congestive loss as a non-congestive loss.

Proposition 1. Let \( \tau_{max} \triangleq \sup\{t \geq 0 : P(P_{A}(t)|CL, r) = 0\} \), we then have:

1. \( P(CL, r, P_{A}(t)) = 0 \) if and only if \( t \leq \tau_{max} \).
2. \( \tau_{max} \geq d_{max} \).

Proof. Please refer to Appendix A. \( \blacksquare \)

Now, we seek an optimal solution for \( \tau_{CD} \), within \([0, \tau_{max}]\). Consider a time period \( t \in [0, \tau_{max}] \) after the retransmission of Packet \( P_r \), when a TCP sender must decide whether or not to activate a congestion response. The risk associated with an activation is that the network may not be congested (i.e. \( P_r \) is not dropped due to congestion). Consequently, the spuriously activated congestion response will reduce \( cwnd \) unnecessarily. On the other hand, if the network is indeed overloaded (i.e. \( P_r \) is lost due to congestion) and the activation of congestion response is excessively delayed, the network congestion may be exacerbated and an expensive RTO may be triggered.

To reach a rational decision regarding the optimal \( \tau_{CD} \), we need to quantify and compare the risks associated with the activation and postponement of congestion response. Two metrics are thus introduced as follows.

Definition 1. The expected cost of activating a congestion response, denoted as \( EC_{A}(t) \), is the product of the conditional probability of \( P_r \) not being lost due to congestion, given that \( P_r \) is unacknowledged, and the cost of falsely activating a congestion response, \( c_{f} \).

Definition 2. The expected cost of delaying a congestion response, denoted as \( EC_{D}(t) \), is the product of the conditional probability that a congestive loss and an RTO have occurred, given that \( P_r \) is unacknowledged, and the cost of excessively delaying a congestion response, \( c_{d} \).

For \( 0 \leq t \leq \tau_{max} \), we can show that:

\[
P(CL, r, P_{A}(t)) = P(P_{U}(t)) \cdot P(CL, r|P_{U}(t)) + P(P_{A}(t)) \cdot P(CL, r|P_{A}(t))
\]

(4)

From (3) and (4), we have:

\[
P(CL, r|P_{U}(t)) = 1 - P(CL, r|P_{U}(t)) = 1 - \frac{P(CL, r)}{P(P_{U}(t))}
\]

(5)

\[
P(T/O \cap CL, r|P_{U}(t)) = \frac{P(T/O|CL, r)P(CL, r)}{P(P_{U}(t))}
\]

(6)

Hence we can show that:

\[
P(CL, r|P_{U}(t)) = \frac{p_i(1-(1-p_i)\max(\tau_{RD}))}{p_i + p_o(1-(1-p_i)\max(\tau_{RD}))}
\]

(7)

\[
P(T/O \cap CL, r|P_{U}(t)) = \frac{p_i p_o}{p_i + p_o(1-(1-p_i)\max(\tau_{RD}))}
\]

(8)

Proof. Please refer to Appendices B and C. \( \blacksquare \)

We now derive \( c_{f} \) and \( c_{d} \) as follows.

Definition 3. The cost of falsely activating a congestion response, denoted as \( c_{f} \), is the average number of packets a TCP sender misses to send due to the activation of a congestion response.

Consider the activation of a congestion response when both \( cwnd \) and \( ssthresh \) are set to \([0.5 \omega]\). The TCP sender is to start the congestion avoidance phase. \( cwnd \) is incremented by one for every RTT afterwards. Let \( n_{CA} \) be the number of cycles during which the TCP sender is in the

\[\text{due to buffer overflow, the buffer will be kept full until the lost packet is retransmitted.} \]

\( \tau_{max} \) is therefore the sum of transmission delays and the maximum queuing delays. However, this does not apply to the case of multiple flows since some flows may back off earlier than others during buffer overflow.

2 We do not know a closed-form expression for \( \tau_{max} \) in the general case. Consider a single flow over a bottleneck link. When a packet is dropped...
Appendix D as: send due to the occurrence of RTO.

Next, we note that denoted as The Definition 4.

The response is \( E_C \) denotes by Eq. (9).

Similarly, when \( c_{wnd} \) is negative, the number of packets the TCP sender misses to send is:

\[
c_f = \frac{c}{w} n_{CA} - \sum_{i=1}^{\infty} \left(\frac{0.5 w}{w} \right) + i - 1
\]

\[
= 0.5 \left[ 0.5 w \right] \left( 0.5 w + 1 \right)
\]

\[(9)\]

Definition 4. The cost of excessively delaying a congestion response, denoted as \( c_f \), is the average number of packets a TCP sender misses to send due to the occurrence of RTO.

Following a similar logic as in deriving \( c_f \), we can derive \( c_d \) in Appendix D as:

\[
c_d = \left[ \log_2(0.5 w) \right] \cdot w - 2^{\log_2(0.5 w)} + 1
\]

\[(10)\]

Observe that \( \frac{c}{w} E_C(t) \leq 0 \) and \( \frac{c}{w} E_D(t) \geq 0 \). Thus, when the activation of congestion response is postponed further, \( E_C(t) \), which quantifies the risk associated with delaying a congestion response, increases, while \( E_D(t) \), which quantifies the risk associated with activating a congestion response, drops. When \( E_C(t) \) is greater than \( E_D(t) \), it is advantageous to set \( T_{CD} \) no less than \( t \) since the operation cost of activating congestion response is \( E_C(t) \), which is larger than that of deferring it, \( E_D(t) \). The cost may drop when \( t \) increases. Similarly, when \( E_D(t) \) is greater than \( E_C(t) \), it is advantageous to set \( T_{CD} \) no greater than \( t \) since the operation cost of deferring congestion response is \( E_D(t) \), which is larger than that of activating it, \( E_C(t) \). The cost may drop when \( t \) decreases. Therefore, the evaluation of the optimal solution of \( T_{CD} \), \( T_{CD}^* \), subject to the constraint \( 0 \leq T_{CD} \leq T_{max} \), can be divided into three different cases, as depicted in Fig. 2.

Case I arises when \( E_D(t) \) exceeds \( E_C(t) \) for any \( t \). The gap between the two functions would keep on increasing when \( t \) increases. Thus, the congestion response should be activated at \( t = 0 \), or set \( T_{CD}^* \) to be zero. Hence, we have:

\[
p_c > (p_1 \cdot \frac{c_f}{c_d} + 1)^{-1} \cdot p_l \]

\[(11)\]

Case II arises when \( E_D(t) \) fails to catch up with \( E_C(t) \) for all \( t \) such that \( 0 \leq t \leq T_{max} \). Since we cannot delay a congestion response further according to the prior constraint, we have to set \( T_{CD}^* \) to be \( T_{max} \). Hence, we have:

\[
p_c < \left( \frac{p_1 \cdot c_f}{[1 - (1 - p_1) \cdot F(T_{max})] \cdot c_d} + 1 \right)^{-1} \cdot p_l \]

\[(12)\]

The final case arises when \( E_D(t) \) catches up with \( E_C(t) \) for some \( t \) such that \( 0 \leq t = T_{th} \leq T_{max} \). \( T_{CD}^* \) thus corresponds to \( T_{th} \) since \( E_C(t) \) is greater than \( E_D(t) \) prior to \( T_{th} \), and less than \( E_D(t) \) after \( T_{th} \). We summarize the optimal value of \( T_{CD} \) by the following theorem.

Theorem 1. Suppose Assumptions A1-A5 hold. Let \( r_c = \frac{c}{w} \). The optimal value of \( T_{CD} \) within \([0, T_{max}]\) for minimizing the associated expected cost, \( r_c^* \), is given by:

\[
r_c^* = \begin{cases} 
0, & r_c > T_{max} \\
T_{max}, & r_c < T_{min} \\
T_{th}, & T_{min} \leq r_c \leq T_{max}
\end{cases}
\]

\[(13)\]
Fig. 4(b) plots $p_i$ for most connections. It follows that congestion response upon packet retransmission.

Therefore, in the presence of (even a small magnitude of) non-congestive loss, it is generally optimal to postpone the decision of congestion response behind packet retransmission by no less than $d_{min}$, so that further information can be collected after retransmission before deciding whether the network is genuinely congested. Present loss-based TCP variants mostly bundles congestion response with packet retransmission, and behaves similarly as the STS model with $r_{CD} = 0$. They thus suffer from heavily sub-optimal performance.

4. TCP-NCL

The STS model is a limited, idealized TCP sender. First, it does not provide a closed-form expression for the optimal $r_{RD}$. Second, it assumes, among other things, prior knowledge of the RTT distribution $F(i)$, which is impractical. Therefore, we propose TCP-NCL to closely approximate STS.

The serialized-timer structure of the STS model is supplemented by TCP-NCL with the NCL-RTT-Update (NRU) process for maintaining the statistics of the RTT samples so as to estimate $F(i)$. Section 4.1 describes the NRU process. Section 4.2 explains the implementation of the retransmission decision (RD) and congestion response decision (CD) timers using timestamps. The pseudocodes for these procedures are exhibited in Algorithms 1 and 2.

Algorithm 1 Procedure NRU(Packet $P_i$)

1: if $P_i$ is in rcPkts then
2: $\text{rtt} \leftarrow \text{now} - \text{txTime}[i]$
3: else
4: if $(\text{now} - \text{txTime}[i]) < \beta \cdot \text{tauCD}$ then
5: $\text{rtt} \leftarrow \text{now} - \text{txTime}[i] + \text{tauRD}$
6: else
7: return
8: end if
9: end if
10: $\text{maxR} \leftarrow \max$(most recent MRRL RTT samples)
11: $\text{minR} \leftarrow \min$(most recent MRRL RTT samples)
12: $\text{tauRD} \leftarrow \text{maxR} + \text{minR}$
13: $\text{tauCD} \leftarrow \text{minR}$

4.1. NCL-RTT-Update (NRU) process

The NRU process is invoked if an ACK, say ACK$_i$, arrives before the expiration of its corresponding CD timer, CD$_i$. This can be further divided into two cases: ACK$_i$ arrives before or after RD$_i$ expires. In the latter case, there is an ambiguity regarding whether ACK$_i$ is for $P^o_i$ or $P^r_i$. Thus, some additional measures would be needed to make sure that ACK$_i$ is for $P^o_i$ before recording the corresponding RTT sample. We record the time instance for the most recent transmission of $P_i$ in txTime[i]. Upon the arrival of ACK$_i$, if the present time (identified by now) exceeds txTime[i] by less than $\beta \cdot CD$ ($0 < \beta < 1$), it is inferred that $P^r_i$ cannot be acknowledged within such a short interval and thus the RTT sample, $(\text{rttRD} + \text{now} - \text{txTime}[i])$, is inserted into rttRcd. Otherwise, the corresponding RTT sample will be ignored.

By updating RTT based on ACKs received in this manner, the RTT sampling process is more robust to changes in the network environment, especially to abrupt increases in RTT caused by path changes or handoffs. When the latter occurs, it is likely that ACK will not be received before RTT$_i$ expires. Yet, the corresponding RTT sample can still be accurately recorded as long as it does not exceed $(\text{rd} + \beta \cdot \text{CD})$.

The distribution of RTT is time-variant over heterogeneous wired/wireless networks, where the network topology and/or load may change during a TCP session. Thus, it is important to ensure that the outdated RTT samples can be discarded. To this effect, we define the maximum RTT record length (MRRL). An RTT sample is discarded if it is older than MRRL.

The assignments of $\text{rd}^r$ and $\text{cd}^r$, as will be discussed in Section 4.2, only rely on the maximum and minimum values in updated RTT records. We store these values in maxR and minR, respectively.

In extreme cases, this may require keeping all the most recent MRRL samples in storage. For example, consider when RTT keeps on decreasing.
Algorithm 2 Procedure UpdateList(Packet ACK_i)

1: for each Packet P_j in cdPkts do
2: \( \text{if } \tau_{CD} < (\text{now} - txTime[j]) \) then
3: remove P_j from cdPkts
4: \( \text{activate congestion response} \)
5: end if
6: end for
7: for each Packet P_j acknowledged by ACK_i do
8: if \( P_j \in (rdPkts \text{ or cdPkts}) \) then
9: \( \text{NRU}(P_j) \)
10: remove \( P_j \) from rdPkts/cdPkts
11: end if
12: end for
13: for each Packet \( P_j \) in rdPkts do
14: if \( \tau_{RD} < (\text{now} - txTime[j]) \) then
15: remove \( P_j \) from rdPkts
16: \( \text{enqueue } P_j \text{ to to-be-rtxed} \)
17: end if
18: end for
19: \( \text{if } \text{wnd} < \min(\text{cwnd}, \text{awnd}) \) then
20: while number of outstanding packets \( \leq \text{wnd} \) do
21: if to-be-rtxed is non-empty then
22: \( \text{dequeue the first packet } P_j \text{ from to-be-rtxed} \)
23: add \( P_j \) to cdPkts
24: \( \text{retransmit } P_j \)
25: \( txTime[j] \leftarrow \text{now} \)
26: else
27: \( \text{transmit the next new packet } P_k \)
28: add \( P_k \) to rdPkts
29: \( txTime[k] \leftarrow \text{now} \)
30: end if
31: end while

4.2. Retransmission decision (RD) and congestion response decision (CD) timers

\( \tau_{RD} \) is set to \( \max R + \min R \). We note that this needs to be sufficiently large to avoid premature packet retransmission. Consider a worst-case scenario when there is a congested network and a packet is locally retransmitted at every hop. \( \max R \) serves as an upper bound on the sum of transmission delays and queueing delays along a path, and \( \min R \) approximately represents the sum of transmission delays, which is equal to the sum of retransmission delays. \( \max R + \min R \) then estimates RTT for such worst-case scenario, and can therefore be a reasonable upper bound of RTT in the general case.

On the other hand, \( \tau_{CD} \) adds to the delay in activating congestion response. Such delay constitutes the feedback delay of the Internet congestion control system, and should be upper-bounded to ensure the stability of the system [37]. In general, the stability conditions can be observed if the delay is in the order of multiples (\( \geq 2 \)) of RTTs, which is in turn satisfied by the prescribed setting of \( \tau_{RD} \).

\( \tau_{CD} \) is set to \( \min R \) in approximating the optimal setting provided by Theorem 1, as illustrated in Fig. 3. As discussed at the end of Section 3, \( \tau_c \leq \tau_{\text{max}} \) generally holds in the presence of non-congestive loss, even if the amount of non-congestive loss is small. Thus, this approximation is in general a conservative implementation of the STS model in that \( \tau_{CD} \) is no greater than the optimal value in most cases. This reduces the aggressiveness of TCP-NCL and minimizes its adverse impact on other competing TCP flows. Moreover, it removes the dependence on \( \tau_{\text{max}} \), which is hard to derive for the general case.

With our approximation above, at any time instance, the expiration periods of the RD timer for all the RD pending packets would be the same. We thus define \( \tau_{CD} \) for the current value of the expiration periods of the RD timer instead of maintaining a separate \( \tau_{RD} \) for each packet \( P_j \). Similarly, we define \( \tau_{CD} \) for the current value of the expiration period of the CD timer.

To fulfill the effects of the RD and CD timers without incurring the heavy cost of maintaining one timer per outstanding packet, we define \( rdPkts \) and \( cdPkts \) for the packets whose respective RD and CD timers are supposedly pending. The two lists are updated upon the arrival of an ACK as follows:

1. Activate a congestion response if there are packets that are in \( cdPkts \) for more than \( \tau_{CD} \), and remove these expired packets.
2. Remove an acknowledged packet from \( rdPkts \) and \( cdPkts \) if it is in these lists.
3. Retransmit packets that are in \( rdPkts \) for more than \( \tau_{RD} \), and transfer them from \( rdPkts \) to \( cdPkts \) with the timestamps being set as the time instances when the retransmissions take place.
4. Transmit new packets and add them to \( rdPkts \) with the timestamps set as the time instances when the transmissions take place.

When activating the congestion control measures, we adopt an approach similar to that used in TCP-PR, that packet losses within the same burst are considered as a single signal about the onset of network congestion and the reduction of \( cwnd \) is triggered only once. Thus, when \( cwnd \) is halved, all packets in \( rdPkts \) and \( cdPkts \) will not cause \( cwnd \) to be halved again.

While it may seem costly to store these two lists of packets, this can be done without introducing any additional storage structure in actual deployment. The sk_buff structure in Linux records all the outstanding packets with timestamps, and each packet has several associated flags that are left unused [38]. These flags can thus be set to indicate the membership of a packet in \( rdPkts \) and \( cdPkts \).

4.3. RTO timer

We keep the RTO timer in the design of TCP-NCL so as to perform worst-case recovery. Upon the expiration of the RTO timer, \( cwnd \) is reset to one, all outstanding packets are retransmitted, and \( rdPkts \) and \( cdPkts \) are cleared. When the expiration period of the RTO timer is less than \( \tau_{CD} \), we increase it to the latter. This prevents the worst-case recovery from being prematurely performed (i.e. before the congestion response installed in the serialized-timer structure is activated).

5. Performance evaluation

In this section, we present our simulation results. We have conducted extensive simulation experiments to evaluate the performance of TCP-NCL using Network Simulator (ns) Version 2.29 [13].

We report the simulation results of TCP-NCL with MRRL set to 1000 and \( \beta \) set to 0.8. Nevertheless, we find that the performance of TCP-NCL is not very sensitive to the protocol parameters. The results are similar for MRRL ranging from 500 to 2000 and \( \beta \) ranging from 0.7 to 0.95. We have also included a TCP variant degenerated from TCP-NCL, known as TCP-NCL-RD, for comparison. TCP-NCL-RD differs from TCP-NCL in that it disables the CD timer by setting the expiration period of the CD timer to zero. Thus, it proactively postpones packet retransmission until the expiration of the RD timer, when it decides that a packet is lost due to congestion and thus activates congestion response and packet retransmission simultaneously. We examine the effect of the serialized-timer structure by comparing TCP-NCL with TCP-NCL-RD.
The performance curves of six TCP variants in the literature, namely RR-TCP [15], TCP-DCR [16], TCP-PR [17], TCPW [23], and TCP BBR [31] are for comparison. The former five variants are replicated from our on-going study [39]. The former three variants attain the best goodput performance against packet reordering according to our simulation-based comparison in [40], and TCPW is a well-known solution for handling wireless packet losses in the literature, whereas TCP BBR is a newly proposed rate-based congestion control variant by Google. The simulation code of TCP BBR in NS2 is chosen from [41].

Three simulated network topologies are used for comparison between different TCP variants, including the infrastructure-based wireless network, the multi-hop wireless network, and the wired network with a dumbbell topology, as illustrated in Fig. 5. The infrastructure-based wireless network, similar to [23], presents an error-prone wireless link connecting a base station and a mobile node. In the multi-hop wireless network, a connection traverses over a number of wireless links. LLRTX is enabled at each wireless link so that packet reordering would be introduced. It can be viewed as a generalization of the single-hop reordering link in [16]. The dumbbell topology is introduced for examining the effectiveness of bandwidth sharing among connections using different TCP variants. Unless specified otherwise, the queue management algorithm used for each link buffer is drop tail. The packet size is 1000 bytes. More detailed network configurations will be described in the following.

Sections 5.1 and 5.2 examine and compare the performance of all the aforementioned TCP variants over the infrastructure-based wireless network and the multi-hop wireless network, respectively. Section 5.3 examines the responsiveness of TCP-NCL against congestive loss over the wired network with a dumbbell topology. Section 5.4 discusses further implications and limitations of our simulation study.

5.1. Non-congestive loss

The infrastructure-based wireless network is illustrated in Fig. 5(a). A TCP sender and a TCP receiver are connected through a wired link with bandwidth 100 Mbps and propagation delay 5 ms, and a wireless link with bandwidth $b_w$ Mbps and propagation delay $d_l$ ms. Random packet errors with rate $p_e$ are deliberately introduced into the wireless link. Packets experiencing transmission errors are discarded at the receiving end.

In each test over the topology, a total of 20 runs, each lasting 2000 s and using different seeds for generating the packet error, have been performed to compute an average value and a 95% confidence interval of the TCP goodput. In order to remove the effect of the transient behaviour in the simulation, only the statistics for the last 1000 s in each run are collected for computing the goodput.
Figs. 6, 8 and 9 plot the connection goodputs of the TCP variants against $p_e$, $d_l$, and $b_w$, respectively. In each plot, the legends for TCP variants are sorted in the order of their relative levels of goodput at the rightmost data points. In most cases, the confidence intervals are too narrow to be distinguishable. The configuration settings are summarized in Table 3.

Fig. 6 exhibits the connection goodputs of the TCP variants against the packet error rate, $p_e$. TCP-NCL attains the best performance for both settings. At $p_e = 1\%$, TCP-NCL attains around 15\% performance improvement over TCP-NCL-RD. The performance gains further increase as $p_e$ increases, and reach around 100\% at $p_e = 10\%$. These verify the effectiveness of the installation of the CD timer in combating wireless losses. TCPW attains the second best performance. It performs similarly as TCP-NCL for $p_e$ up to 2\%, but performs slightly worse as $p_e$ further increases. TCP-NCL-RD, RR-TCP, TCP-DCR, and TCP-PR demonstrate similar drastic decrease in goodput as $p_e$ increases. They do not take non-congestive packet loss into account, resulting in severe under-utilization of network resources.

Fig. 8 exhibits the connection goodputs of the TCP variants against the propagation delay of the wireless link, $d_l$, for $p_e = 1\%$, 3\%, or 5\%. All TCP variants experience performance degradation to different extents as $d_l$ increases. TCP-NCL and TCPW attain much higher connection goodput than other TCP variants. TCP-NCL-RD, RR-TCP, TCP-DCR, and TCP-PR demonstrate similar severe performance deterioration as $d_l$ increases. To fully utilize the wireless link capacity, the size of congestion window ($cwnd$) has to be at least the bandwidth-delay product, namely, the product of $b_w$ and the round-trip propagation delay. Such delay increases linearly as $d_l$ increases. With a larger value of $d_l$, it takes a TCP connection longer to resume $cwnd$ to reach the product upon spurious reductions of $cwnd$ due to non-congestive losses.
a high packet error rate. For example, TCP-NCL outperforms TCPW for $pe \geq 5\%$, $dl = 50$, and $bw = 1$ in Fig. 6, and for $pe = 5\%$, $dl \leq 160$, and $bw = 1$ in Fig. 8(c).

(2) When both perform spurious reductions of cwnd due to non-congestive losses, TCP-NCL tends to reduce cwnd more abruptly than TCPW. This can be exemplified in a scenario with a high bandwidth-delay product and a low packet error rate. For example, TCPW outperforms TCP-NCL with for $pe = 1\%$, $dl \geq 80$, and $bw = 1$ in Fig. 8(a), and for $pe = 1\%$, $dl = 50$, and $bw \geq 2$ in Fig. 9(a).

(3) The aforementioned two inferences can counteract each other in a scenario with a high bandwidth-delay product and a high packet error rate. For example, TCP-NCL performs similarly as TCPW for $pe = 5\%$, $dl \geq 100$, and $bw = 1$ in Fig. 8(b) and for $pe = 5\%$, $dl = 50$, and $bw \geq 2$ in Fig. 9(c).

5.2. Packet reordering

In the multi-hop wireless network as illustrated in Fig. 5(b), a TCP sender is connected to a TCP receiver via six wireless links. Random channel error is introduced into the wireless links with a random packet error rate ranging from 0 to 10%. Packets experiencing transmission errors are discarded at the receiving end of the wireless link. Yet, the setting differs from the infrastructure-based wireless network in that a packet discarded at the link layer is locally retransmitted around 110 ms after its most recent (re)transmission. The retransmission period of 110 ms is the round-trip propagation and transmission delay over a wireless link. The retransmission for the same packet is performed up to three times. This greatly enhances the chance of successful packet delivery at each hop. Yet, packet reordering will be introduced since the retransmitted packets can be intermixed with some later transmitted packets. The intensity of packet reordering tends to increase as the packet error rate increases. Moreover, under a high packet error rate, some packets may still fail to be correctly delivered after three link-layer retransmissions. Consequently, TCP will be confronted with both packet reordering and non-congestive packet loss.

Fig. 7 exhibits the connection goodputs of the TCP variants against the packet error rate over four-hop and six-hop wireless networks. Over the four-hop connection as exhibited in Fig. 7(a), TCP-NCL, TCP-PR, and TCP-DCR outperform other variants for $pe \geq 7\%$, thereby demonstrating their robustness against persistent reordering. TCP-NCL-RD performs slightly worse. It activates a congestion response if a packet is not acknowledged within a time period of $twaRDi = maxR + minR$ after a packet transmission. The time lag is generally shorter than those of TCP-NCL, TCP-PR, and TCP-DCR, leading to a higher chance of a premature congestion response. When $pe$ further increases beyond $7\%$ and non-congestive losses are no longer transparent to the transport layer, TCP-NCL and TCP-PR performs better than TCP-DCR and TCP BBR. Both TCP-NCL and TCP-PR rely purely on timers for inferring packet loss and network congestion, whereas TCP-DCR relies partly on duplicate ACK. In the presence of packet reordering, duplicate ACK conveys little information about the network status, and thus less reliable than timers.

Similar inferences can be made for the six-hop connection as shown in Fig. 7(b). TCP-NCL and TCP-PR outperform other TCP variants as inferred from the simulation results for the four-hop connection.

5.3. Congestive loss

In the wired network with a dumbbell topology as illustrated in Fig. 5(c), $N$ pairs of TCP senders and receivers are sharing a wired bottleneck link with the bandwidth of 5 Mbps, propagation delay of 50 ms, and buffer size of 50 packets. For $i = 1, 2, \ldots, N$, one TCP connection is set up from Sender $S_i$ to Receiver $D_i$. We examine the responsiveness of TCP-NCL against congestive loss over this network. In general, if a TCP variant, say, TCP-X, is competently responsive against

---

**Fig. 10.** Goodput to fair bandwidth share ratio (GBR) of SACK TCP over wired network with a dumbbell topology.

**Fig. 11.** Jain’s fairness index ($J$) over wired network with a dumbbell topology.

---

In the presence of non-congestive losses, TCPW tends to trigger spurious reductions of cwnd more frequently than TCP-NCL. This can be exemplified in a scenario with a low bandwidth-delay product and throughput constraints. Packet losses are more likely to be due to congestion in this scenario. Therefore, we can make three inferences in comparing TCP-NCL and TCPW:

1. In the presence of non-congestive losses, TCPW tends to trigger spurious reductions of cwnd more frequently than TCP-NCL. This can be exemplified in a scenario with a low bandwidth-delay product and throughput constraints. Packet losses are more likely to be due to congestion in this scenario. Therefore, we can make three inferences in comparing TCP-NCL and TCPW:

---

**Fig. 7(a)** exhibits the connection goodputs of the TCP variants against the bandwidth of the wireless link $bw$, for $pe = 1\%$, 3\%, or 5\%. TCP-NCL and TCPW manage to maintain a linear increase in goodput as $bw$ increases up to 4 Mbps, thereby maintaining a steady bandwidth utilization. All other TCP variants maintain similar connection goodput across different values of $bw$. This essentially corresponds to a decrease in bandwidth utilization as $bw$ increases. Similar inferences made from Fig. 8 can apply here. The bandwidth-delay product varies linearly with $bw$. With a larger value of $bw$, it takes a TCP connection longer to recover from spurious cwnd reductions due to non-congestive losses.

Up to now, we can observe that TCP-NCL attains significant performance gain over TCP-NCL-RD across different configurations of $bw$, $dl$, and positive $pe$, by merely the installation of the CD timer. The serialized-timer structure thus proves to be very effective in shielding the impact of non-congestive losses.

We now draw a closer comparison between TCP-NCL and TCPW. TCP-NCL tries to construct more reliable signals of a congestive loss by differentiating between a congestive loss and a non-congestive loss, and halves cwnd when inferring a congestive loss. TCPW does not perform such differentiation. Rather, it reduces cwnd to its estimated available network capacity upon each inferred loss. Therefore, we can make three inferences in comparing TCP-NCL and TCPW:

1. In the presence of non-congestive losses, TCPW tends to trigger spurious reductions of cwnd more frequently than TCP-NCL. This can be exemplified in a scenario with a low bandwidth-delay product and throughput constraints. Packet losses are more likely to be due to congestion in this scenario. Therefore, we can make three inferences in comparing TCP-NCL and TCPW:
the occurrence of a congestive loss, it is expected to fulfill the following two objectives:

1. **TCP-friendliness**: A connection driven by the standardized TCP, such as SACK TCP, can maintain a similar goodput when its competing connections are driven by TCP-X as when its competing connections are driven by the standardized TCP. This ensures that applications running on top of the standardized TCP variant does not suffer when TCP-X is deployed.

2. **Intra-protocol fairness**: Multiple competing connections driven by TCP-X are able to share links fairly as reflected by their similar throughput attained.\(^4\)

For evaluating the TCP-friendliness of TCP-NCL, we conduct two tests. In the first test, all connections are driven by SACK TCP. In the second test, half of the connections are still driven by SACK TCP, and the other half of the connections are driven by TCP-NCL. In each test, suppose Connections \(1, 2, \ldots, M\) are SACK TCP connections and \(M + 1, M + 2, \ldots, N\) are non-SACK TCP connections (and thus \(M = N\) and \(M + \frac{N}{2}\) in the first and second tests, respectively), we compute the goodput to bandwidth share ratio (GBR) as \(\frac{\sum_{i=1}^{N} g_{pi}}{C}\), where \(g_{pi}\) (\(i = 1, 2, \ldots, N\)) and \(C\) denote the goodput of Connection \(i\) and the capacity of the bottleneck link, respectively. GBR is scaled by the number of SACK TCP connections \(M\), and is thus comparable across the three tests. Moreover, in the first test, multiple SACK TCP connections should be able to share the bandwidth efficiently to generate the goodput, thereby attaining GBR close to one. By examining GBR obtained in the second test and comparing it with the benchmarked GBR obtained in the first test, we can evaluate how the presence of TCP-NCL affects the standardized TCP and thus the TCP-friendliness of TCP-NCL.

Fig. 10 plots GBR against \(N\). We see that GBR experiences only slight changes across the two tests, implying minimal impact on SACK TCP due to TCP-NCL. Moreover, when SACK TCP competes with TCP-NCL, GBR slightly increases as the number of flows increases. This is because the number of outstanding data packets for each flow decreases as the number of flows increases. When the network becomes congested, it generally takes SACK TCP longer to accumulate enough duplicate ACK to activate congestion response. Thus, SACK TCP becomes less responsive against congestion. On the other hand, TCP-NCL relies on timers for activating congestion response, the responsiveness of which is generally independent of the external network environment. Hence, as the number of flows increase, SACK TCP tends to be slightly more aggressive than TCP-NCL.

For measuring the intra-protocol fairness, we introduce the Jain’s fairness index, \(J\) \(^{42}\), as the performance metric. A set of \(N\) connections, namely, Connections \(1, 2, \ldots, N\), share a bottleneck link with throughputs \(t_{p1}, t_{p2}, \ldots, t_{pN}\), respectively. \(J\) is defined as:

\[
J = \frac{(\sum_{i=1}^{N} t_{pi})^2}{N \sum_{i=1}^{N} t_{pi}^2}
\]  

(17)

By construction, \(J\) is bounded between zero and one, and attains one only when all throughputs are equal. Furthermore, it equals \(\frac{1}{N}\) when only \(k\) out of the \(N\) connections share the bandwidth equally and the remaining \((N - k)\) connections receive a zero bandwidth.

We perform two simulation tests with all connections driven by SACK TCP and TCP-NCL, respectively. Fig. 11 plots \(J\) against \(N\). We observe that both SACK TCP and TCP-NCL attain \(J\) close to one. This demonstrates that TCP-NCL is capable of maintaining similar intra-protocol fairness as SACK TCP.

5.4. Further discussion

**Scope of comparison**: In essence, TCP-NCL is an optimized TCP variant over heterogeneous wired/wireless networks, within the design space of end-to-end, loss-based TCP enhancement. Actually, most of TCP variants are classified as loss-based TCP and delay-based TCP. Although delay-based TCP can detect and handle the incipient congestion earlier than loss-based protocols in the high bandwidth-delay product network, it is difficult to obtain accurate delay from network measurements. Hence, in this paper, we focus on the extension of the loss-based approaches. The literature on TCP is vast. Due to the limited scope of the paper, we can thus include only those TCP variants in the same design space for comparison. Particularly, some ECN-based and delay-based TCP variants are not included, such as \([29]\). Yet, we have compared the performance of TCP-NCL with TCP BBR, which is a newly proposed TCP variant by Google. TCP BBR can work well in high-speed and wide-area networks built using commodity switches. However, it may not perform well in certain wireless networks. Our simulation results show that TCP BBR can perform unsatisfactorily when the packet error rate increases. TCP BBR uses the product of the estimated maximum bandwidth and estimated RTT to define the estimated bandwidth-delay product (BDP). The transmission rate is calculated based on the estimated BDP. However, as seen in the simulation results, the estimated maximum bandwidth in wireless link with packet error is less than the actual bandwidth. Hence, the performance of TCP BBR drops with the wireless packet errors.

**Operational region**: We have tested the performance of TCP-NCL under non-congestive loss and packet reordering in Sections 5.1 and 5.2, respectively. We have also stress tested TCP-NCL under multiple (\(\geq 8\) con)secutive packet losses, which can possibly arise during link break. We observe that TCP-NCL suffers severe performance degradation (along with other TCP variants in comparison). Nevertheless, in such scenarios, we note that the minimal assumption on the network (that it can deliver most of the data packets in any order) is violated. By requiring end-to-end protocols to perform well in such scenarios, we may go to the other extreme by pushing the necessary network functionalities from the network core to the network edges.

**Fairness over general networks**: We have examined whether TCP-NCL can share bandwidth fairly with SACK TCP in conventional wired networks in Section 5.3, as is customary in the literature. Yet, whether TCP variants can share bandwidth fairly in the presence of packet reordering and/or congestive loss remains an open issue. We find that, in some cases, it is impossible for a standardized TCP variant, such as SACK TCP, to remain intact when its competing flows are upgraded to become more robust against packet reordering. This is because the link utilization is improved as a result of the upgrade, which in turn increases the level of congestive loss rates seen by all TCP flows. Interested readers can refer to \([43]\) for our systematic study in this respect.

6. Conclusions

In this paper, we have proposed a novel TCP variant, known as TCP-NCL, as a unified solution for performing loss recovery, sequencing control, and congestion control over general, error-prone channels. In particular, we propose the use of two serialized timers for obtaining more reliable signals for packet loss and network overload separately. The STS model has been constructed based on the concept of expected cost and closed-form expressions are derived as references for setting the timer expiration periods. We note that the timers are mostly determined intuitively in existing work. Our simulation investigations reveal that TCP-NCL is capable of attaining significant performance improvement over general, error-prone channels. It also demonstrates competent responsiveness against congestive loss by maintaining effective congestion avoidance and good TCP-friendliness.
Some recent research on wireless TCP focus on performing end-to-end loss recovery via erasure coding or network coding \[44,45\], the design of which is generally orthogonal to the congestion control component of TCP. It is thus feasible to combine TCP-NCL with these variants. Such combination can potentially improve both the efficiency of utilizing network capacity and the efficiency of loss recovery, and is worth further investigation. For future research, we may also consider to extend our work to wireless networks with multiple sources. Another possible extension is to generalize the usage of “expected cost”, which exploits the available information for optimal decision-making from a probabilistic perspective, so as to design TCP to perform effectively in heterogeneous environments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Proof of Proposition 1

**Proof.** 1. Observe that:

\[
P(C_{L_i}^n|PA_i(t))P(PA_i(t)) = P(PA_i(t)|C_{L_i}^n)P(C_{L_i}^n)
\]

and

\[
P(PA_i(t)) \geq P(PA_i(t|0)) = (1 - \rho_i)(1 - \zeta) > 0
\]

Thus, \(P(C_{L_i}^n|PA_i(t)) = 0\) if only if \(P(PA_i(t)|C_{L_i}^n) = 0\). By construction, the latter holds if only if \(t \leq \tau_{\text{max}}\). The first part of the proposition follows.

2. Noting that \(P(PA_i|d_{\text{min}}|C_{L_i}^n) = 0\) since:

\[
P(PA_i|d_{\text{min}}|C_{L_i}^n) = \frac{P(C_{L_i}^n \cap PA_i|d_{\text{min}})}{P(C_{L_i}^n)} \xrightarrow{\text{as} \ z \to 0} \frac{P(PA_i|d_{\text{min}})}{P(C_{L_i}^n)}
\]

\[
\Rightarrow P(PA_i|d_{\text{min}}|C_{L_i}^n) = 0\]

Appendix B. Proof of (7)

**Proof.** \(PU_i(t)\) and \(PA_i(t)\) are complementary events. By the law of total probability:

\[
P(C_{L_i}^n) = P(PU_i(t) \cap C_{L_i}^n)P(PU_i(t)) + P(\overline{PA_i(t)} \cap C_{L_i}^n)P(\overline{PA_i(t)}).
\]

For \(0 \leq t \leq \tau_{\text{max}}\), since \(P(C_{L_i}^n|PA_i(t)) = 0\) by Proposition 1, this reduces to:

\[
P(C_{L_i}^n) = P(PU_i(t) \cap C_{L_i}^n)P(PU_i(t))
\]

It follows that:

\[
P(\overline{C_{L_i}^n}|PU_i(t)) = 1 - P(C_{L_i}^n|PU_i(t))
\]

\[
= 1 - \frac{P(C_{L_i}^n)}{P(\overline{PU_i(t)})}
\]

To derive \(P(\overline{PU_i(t)}|\overline{PU_i(t)})\) in (B.3), we note that a packet is unacknowledged if and only if both its originally transmitted and retransmitted copies are unacknowledged. In other words,

\[
PU_i(t) = PU_i^r(t) \cap PU_i^r(t)
\]

and thus:

\[
P(\overline{PU_i(t)})
\]

\[
= P(\overline{PU_i^r(t)} \cap \overline{PU_i^r(t)})
\]

\[
= P(\overline{PU_i^r(t)} \cap \overline{PU_i^r(t)}|\overline{PU_i(t)})P(\overline{PU_i(t)})
\]

\[
+ P(\overline{PU_i^r(t)} \cap \overline{PU_i^r(t)}|\overline{PU_i(t)})P(\overline{PU_i(t)})
\]

\[
(B.5)
\]

Now, we claim that \(P(\overline{PU_i^r(t)} \cap \overline{PU_i^r(t)}|\overline{PU_i(t)}) \approx 0\) since:

\[
P(\overline{PU_i^r(t)} \cap \overline{PU_i^r(t)}|\overline{PU_i(t)}) \leq P(\overline{PU_i^r(t)}|\overline{PU_i(t)})
\]

\[
= 1 - P(t + t_{\text{RTT}})
\]

\[
\leq \zeta \ll 1
\]

It follows that:

\[
P(\overline{PU_i(t)}) \approx P(\overline{PU_i^r(t)} \cap PU_i^r(t) | PL_i^c)P(PL_i^c)
\]

\[
(B.6)
\]

where

\[
P(\overline{PU_i^r(t)} | PL_i^c) \approx 0
\]

\[
(B.7)
\]

Recall that \(PL_i^c\) denotes the loss of Packet \(P_i^c\), which can be due to either congestive or non-congestive loss, i.e., \(PL_i^c = CL_i^c \cup NL_i^c\). Thus,

\[
P(\overline{PU_i(t)}) = P(\overline{PU_i^r(t)} | CL_i^c \cup NL_i^c)
\]

\[
= P(\overline{PU_i^r(t)} | CL_i^c) + P(\overline{PU_i^r(t)} | NL_i^c)
\]

\[
= P(C_{L_i}^n)P(\overline{PU_i^r(t)} | CL_i^c) + P(\overline{PU_i^r(t)} | NL_i^c)
\]

\[
(B.10)
\]

where \(CL_i^c\) and \(NL_i^c\) are mutually exclusive events. Recall that \(\tau_{\text{max}} = \sup \{t|\overline{PA_i|d_{\text{min}}|C_{L_i}^n} = 0\}\). Thus, for \(0 \leq t \leq \tau_{\text{max}}\),

\[
P(\overline{PU_i^r(t)} | CL_i^c) \geq P(U_i(t)|CL_i^c)
\]

\[
= 1 - P(PA_i(t)|CL_i^c) = 1
\]

\[
(B.11)
\]

\[
\Rightarrow P(\overline{PU_i^r(t)} | CL_i^c) = 0
\]

\[
(B.12)
\]

(B.10) is therefore further simplified as:

\[
P(\overline{PU_i(t)}) \approx P(\overline{PL_i^c})P(PL_i^c)
\]

\[
(B.13)
\]

where \(P(\overline{PU_i(t)})\) can be expanded as:

\[
P(\overline{PU_i(t)}) = P(\overline{PU_i^r(t)}|PL_i^c)P(PL_i^c)
\]

\[
+ P(\overline{PU_i^r(t)}|CL_i^c)P(CL_i^c)
\]

\[
= \rho_i + [1 - (1 - \rho_i)]F(t)\rho_i
\]

Hence,

\[
P(\overline{PU_i(t)}) = \rho_i + [1 - (1 - \rho_i)]F(t)\rho_i
\]

\[
(B.14)
\]

where \(0 \leq t \leq \tau_{\text{max}}\). This leads to (7) when substituting back to (B.3).  □

Appendix C. Proof of (8)

**Proof.** Following a similar derivation as (B.1)–(B.3), we can obtain:

\[
P(\overline{TO} \cap CL_i^c|\overline{PL_i^c}) = \frac{P(TO|CL_i^c)P(PL_i^c)}{P(\overline{PL_i^c})}
\]

\[
(C.1)
\]

where \(0 \leq t \leq \tau_{\text{max}}\). (8) follows by substituting (2) and (B.15) into (C.1).  □
Appendix D. Derivation of the cost of excessively delaying a congestion response, $c_d$

Consider the occurrence of an RTO when $cwnd$ is reset to one and $ssthresh$ is set to 0.5 $w$. For simplicity, we would assume that an ACK arrives before the occurrence of another RTO. Thus, after being reset, the TCP sender is in the slow start stage and $w$ will double itself at every RTT round as long as it is less than $ssthresh$. Once $w$ reaches $ssthresh$, the TCP sender leaves the slow start stage and enter the same congestion avoidance stage. Let $n_{SS}$ be the number of cycles during which the TCP sender is in the slow start stage. We have:

$$n_{SS} = \lceil \log_2(0.5w) \rceil$$

(D.1)

Upon a genuine congestion loss, it is generally desirable for a TCP sender to halve $cwnd$ and directly enter the congestion avoidance stage. Therefore, if an RTO occurs and $cwnd$ is reset to one, the throughput of the TCP sender is penalized when the slow start phase is activated. The number of packets the TCP sender misses to send is:

$$c_d = w - n_{SS} - \sum_{i=1}^{n_{SS}} 2^{i-1} = \lfloor \log_2(0.5w) \rfloor \cdot w - 2^{\lfloor \log_2(0.5w) \rfloor} + 1$$

(D.2)

References