

Enhancing Wireless TCP: A Serialized-Timer Approach

Chengdi Lai, Ka-Cheong Leung, and Victor O.K. Li
 Department of Electrical and Electronic Engineering
 The University of Hong Kong
 Pokfulam Road, Hong Kong, China
 E-mail: {laichengdi, kcleung, vli}@eee.hku.hk

Abstract—In wireless networks, TCP performs unsatisfactorily since packet reordering and random 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 propose a novel TCP variant, known as TCP for non-congestive loss (TCP-NCL), to adapt TCP to wireless networks by using more reliable signals of packet loss and network overload for activating packet retransmission and congestion response, separately. TCP-NCL can thus serve as a unified solution for effective congestion control, sequencing control, and loss recovery. Different from the existing unified solutions, the modifications involved in the proposed variant are limited to sender-side TCP only, thereby facilitating possible future wide deployment. The two signals employed are the expirations of two serialized timers (the second timer is started upon the expiration of the first timer). Intuitively, the reliability of the two signals tend to increase with longer timer expiration periods. Yet, excessively long expiration periods of timers will undermine the responsiveness of TCP-NCL against genuine packet loss and network overload. A smart TCP sender model has thus been developed for optimizing the timer expiration periods. Our simulation studies reveal that TCP-NCL is robust against packet reordering as well as random packet loss while maintaining responsiveness against situations with purely congestive loss.

Index Terms—Transmission Control Protocol (TCP), wireless network, congestion control, congestion response decision timer, flow control, loss recovery, packet reordering, random packet loss, retransmission decision timer, sequencing control, TCP-NCL

I. INTRODUCTION

Transmission Control Protocol (TCP) [2], [13], [21] is the most important transport layer protocol in current networks, providing in-order data delivery services to various applications and featuring reliable transmission and *end-to-end* congestion control. In fulfilling the former feature, signals of packet loss have been used for triggering retransmission 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 overloading a congested network.¹ However, how to identify signals and utilize the identified signals to perform effective congestion control and loss recovery has long been a very challenging research problem.

The popular TCP variants, such as TCP Reno [2], [23], 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 forced to reopen from one packet size by employing the slow start algorithm.

A TCP receiver would expect all the data packets received to be consecutively ordered. Otherwise, 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, say, three, 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.

A. Wireless Transmission Channel

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 wireless networks due to the significant level of occurrences of random packet losses and packet reordering [15].

¹Some variants of TCP [7], [20], [25] periodically adjust *cwnd* by estimating the available bandwidth instead of employing the additive-increase-multiplicative-decrease (AIMD) algorithms [13] for congestion control, claiming to be more effective solutions over networks with large bandwidth-delay product. Nevertheless, AIMD algorithms have been proved to be computationally efficient in offering robust performance over a wide range of networks. Thus, many popular TCP variants, including our proposed TCP-NCL in this paper, are designed based on the AIMD algorithms.

As compared with the wired media, the wireless medium provides much more lossy physical links for data transmissions. Signals propagating over wireless channels suffer from degradation, interference, and noise. Packets received may be damaged beyond the recovery capability of error control codes, if any. These packets are thus discarded, leading to the occurrence of random packet losses.

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 behavior, it is in fact persistently observed over modern networks and can be caused by a myriad of reasons [15], [16]. Due to the high transmission error rates and, in some cases, mobility in wireless networks, packet reordering is further increased substantially when the transmission medium evolves from physical cables to wireless. Specifically, four causes of packet reordering are commonly observed over wireless networks, including:

Link layer retransmission (LLRTX): To combat high transmission error rates of wireless channels, some link layer retransmission mechanisms [3], [12] have been proposed to locally retransmit damaged packets at the link layer. As a side effect of local retransmission, the packet order of a flow is altered.

Path change: Over mobile ad-hoc networks (MANETs), a TCP flow traverses a number of wireless nodes. The transmission path of the flow may be altered when some nodes move. The round trip time (RTT) of the connection may change too. Consequently, it is possible that some packets transmitted after the path change arrive at the receiving end before those packets transmitted prior to the path change.

Hand-off: In infrastructure-based networks, the complete transmission coverage of an entire area is achieved cooperatively by a set of base stations. When a mobile client moves from the radio range of one base station to that of another, a hand-off between these two base stations occurs, changing the transmission path for the flow. The resulting variation in RTT may lead to the occurrence of packet reordering.

Packet-level multi-path routing: In wireless mesh networks (WMNs) and many other networks of rich connectivity, multi-path routing have been proposed for load balancing. When it is carried out at the packet level, packets may be reordered due to the difference in time delays of distinct paths.

Hence, in 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.

B. Our Contributions

Our proposed novel TCP variant, known as TCP for non-congestive loss (TCP-NCL), employs more reliable signals of packet loss and network congestion over a general error-prone channel for activating packet retransmission and congestion

response, respectively. Thus, TCP-NCL serves as a unified solution for effective congestion control, sequencing control, and loss recovery. Moreover, the proposed modifications involved are limited to sender-side TCP only, thereby facilitating possible future wide deployment. In the literature, most existing TCP variants only focus on congestion control and loss recovery with the presence of either random packet loss or packet reordering. A few unified solutions, such as ATCP [17], generally require information and modifications beyond the scope of the transport layer, hindering possible future wide deployment.

The two signals of packet loss and network congestion are the expirations of two serialized timers. The first timer is started when a packet is first injected into the network, while the second timer is started upon the expiration of the first timer. A smart TCP sender (STS) model has been constructed for analytically optimizing the timer expiration periods.

Our simulation study reveals that TCP-NCL is robust against packet reordering as well as random packet loss. At the same time, it maintains responsiveness against situations with purely congestive loss as in the case of conventional wired networks.

C. Organization of the paper

This paper is organized as follows. Section II discusses existing solutions for TCP packet reordering and random loss to put TCP-NCL in perspective. Section III develops the STS model, explaining the motivation behind our serialized-timer structure and modeling the optimal timer expiration periods. Section IV presents TCP-NCL based on the analytical results of the STS model. Section V examines the performance of TCP-NCL and compares it with some popular TCP variants. Section VI concludes and discusses some possible extensions of our work.

II. RELATED WORK

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

AVG, DEL, EWMA, and INC in [5], RR-TCP [27], TCP-DCR [4], and TCP-PR [6] propose to abandon the fixed triple duplicate ACKs as signals 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. However, these variants use the same set of signals for activating congestion response, and are incapable of differentiating between congestive and non-congestive losses. They thus have to rely on LLRTX for making the on-going random packet losses over wireless links transparent to the transport layer.

DSACK TCP [10], TCP-DOOR [24], and TCP-Eifel [18] 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 retransmission after activating fast retransmission and fast recovery upon the arrival of triple duplicate ACKs². Upon successful detection, *cwnd* will be recovered immediately or via the slow start process. TCP-DOOR also disables congestion control for a time interval as it assumes out-of-order events are often caused by path change. However, these variants are generally unable to recover *cwnd* unnecessarily reduced in the event of random packet losses. Moreover, as shown by our simulation results, TCP-DOOR tends to excessively disable congestion control actions under persistent packet reordering, resulting in substantial deterioration in performance.

JTCP [26], TCP-Veno [11], and TCP-Westwood (TCP-W) [8] focus on differentiating congestive and non-congestive losses by using the estimated network load upon the arrival of triple duplicate ACKs, which is still treated by them as a credible signal of packet loss for triggering retransmission. One of their major merits is that the binary signal of packet loss no longer dictates the activation of congestion response. Instead, the signal is combined with the estimated network load to decide by how much *cwnd* should be reduced. Yet, the inherent assumption of nearly in-order channel obviously limit the applicability of these variants over wireless networks and many other modern networks, where packet reordering are common. Particularly, persistent reordering may significantly affect the adaptation of *cwnd* to the available network bandwidth by frequently activating minor reductions of *cwnd* and triggering false retransmissions.

TCP-Probing [14] 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 the 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. Again, 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 loading.

The existing unified solutions require information and/or modifications of the network protocol stack beyond the transport layer. ATCP [17] 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.

In summary, there is a lack of a transport layer based unified solution with reliable signals of network overload and packet loss for performing congestion control and loss

²The same design philosophy is also employed by F-RTO [22], which focuses on detecting spurious retransmission triggered by RTO. However, false RTO is not the major reason for performance deterioration over wireless networks. We thus do not include F-RTO as a wireless TCP enhancement.

recovery over general, error-prone transmission channels. Such unified solution is obviously highly desirable, as it adapts TCP to a wide range of wireless and, in some cases, wired transmission channels at a minimal deployment cost.

III. SMART TCP SENDER MODEL

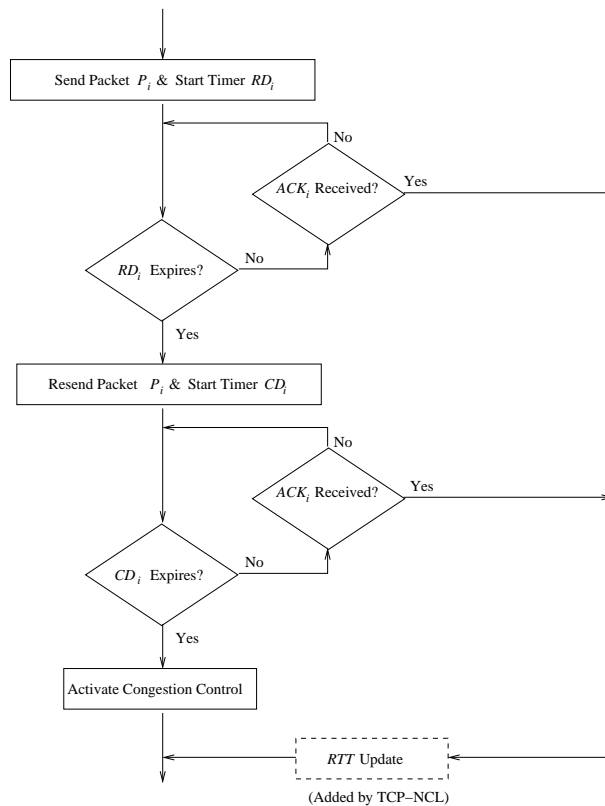


Fig. 1. A flowchart for STS model.

In constructing an aforementioned unified solution, we start from the following two initial observations:

1) Nature of signals: Over general communication channels, there are two prevalent approaches for inferring the occurrence of packet loss based on: 1) the number of consecutively received duplicate ACKs reaches an adaptively evolved threshold value or, 2) a timer³ expires. 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 of the insufficient number of the outstanding packets. Special measures, like limited retransmit [1], would have to be incorporated in order to alleviate the problem by sending extra data packets upon the arrival of duplicate ACKs. By contrast, a more elegant and robust approach for estimating the probability of packet loss is to measure the time elapsed since a packet transmission (or some other similar instances) and refer to the past recent history of RTT.

2) Separation of signals: Over error-prone channels, signals of packet loss alone can only serve as prior signals of network congestion. It is necessary to take additional information into

³The timer is different from the 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.

account to confirm whether the inferred loss can be categorized as a congestive loss.

In place of the conventional approach of activating fast retransmission and fast recovery simultaneously, we propose to delay the decision on whether to activate a congestion response behind that of a packet retransmission for a short time period. The motivation behind doing so is two-fold. First, information inferred by the occurrences of 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 loading condition so that this helps to make an appropriate decision. 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 can either 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 rejecting the necessity of 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 retransmission than a false congestion response. In the event of a false fast retransmission, there is at most one full-sized packet being retransmitted spuriously. However, there is a significant portion of the available bandwidth being left unused in the event of a falsely activated congestion response. For TCP variants which activate fast retransmit and fast recovery simultaneously, they may defer both measures to avoid the penalty of a false congestion response until a congestive loss can be confirmed highly accurately. 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 packet retransmission before the activation of congestion response, the risk of triggering an RTO can be more effectively reduced while avoiding the heavy penalty due to a false congestion response. Nevertheless, while a comparatively higher level of false retransmission can be tolerated with our proposal, it should still be maintained at a minimal level to avoid significant waste 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_i is started whenever a new packet P_i is injected into the network. If ACK_i is received by the TCP sender before RD_i expires, RD_i will be cancelled. Otherwise, P_i will be retransmitted and a congestion response decision timer CD_i will be started.

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,

TABLE I
GENERAL NOTATIONS.

C_S	Number of packets TCP sender misses to send due to SC
C_T	Number of packets TCP sender misses to send due to the extra reduction of TCP sender's throughput introduced by RTO
$F(t)$	Cumulative distribution function of RTT for Packet P_i
G	Goodput of TCP sender
$mrtt_i$	Minimum attainable RTT for Packet P_i
P_i^o	First transmitted Packet P_i
P_i^r	Retransmitted Packet P_i
p_c	Congestive loss rate
p_l	Loss rate
p_w	Non-congestive loss rate
T	Throughput of TCP sender
W	Average $cwnd$ over a TCP session

TABLE II
NOTATIONS FOR EVENTS.

CL_i^o	P_i^o lost due to congestion
CL_i^r	P_i^r lost due to congestion
NL_i^o	P_i^o experiencing non-congestive loss
NL_i^r	P_i^r experiencing non-congestive loss
RTX_i	Fast retransmission of P_i
$PA_i(t)$	Either P_i^o or P_i^r acknowledged by time t after RTX_i
$PA_i^o(t)$	P_i^o acknowledged by time t after retransmission
$PA_i^r(t)$	P_i^r acknowledged by time t after retransmission
$PU_i(t)$	Both P_i^o and P_i^r unacknowledged by time t after RTX_i
$PU_i^o(t)$	P_i^o unacknowledged by time t after retransmission
$PU_i^r(t)$	P_i^r unacknowledged by time t after retransmission
TO	Retransmission timeout

this eliminates the need for activating any congestion control measures.

The expiration periods of CD_i and RD_i (denoted as τ_{RD_i} and τ_{CD_i} , respectively) are evaluated in sections III-A and III-B, respectively. To facilitate the subsequent discussion, we define the notations as shown in Table I and Table II.

The following assumptions are made to simplify the discussion:

- 1) Data packets sent are of equal size. Accordingly, we express the size of congestion window and slow start threshold ($ssthresh$) in terms of the number of packets.
- 2) $P(PA_i(t)|CL_i^o)$ continuously monotonically increases with time t and $\lim_{t \rightarrow \infty} P(PA_i(t)|CL_i^o) > 0$. Given that PL_i^o is lost due to congestion, the chance of Packet P_i being acknowledged relies on P_i^r . By allowing P_i^r more time (i.e., increasing t), the chance of P_i being acknowledged does not drop. Furthermore, the chance should be non-zero, given that P_i^r is allowed sufficient time ($t \rightarrow \infty$),
- 3) Non-congestive loss is assumed to be random. In other words, we do not consider correlated or burst loss for simplicity. Nevertheless, simulation results will show that our proposed TCP variant still offers competent performance under burst loss. Thus, NL_i^o is mutually independent with any events occurring with P_i^r and vice versa. In particular, it follows that:

$$P(NL_i^o \cap PU_i^r(t)) = P(NL_i^o) \cdot P(PU_i^r(t)) \quad (1)$$

A. Retransmission Decision Timer

τ_{RD_i} determines how long we have to wait before activating packet retransmission. The assignment of τ_{RD_i} requires careful balancing between the objectives of prompt retransmission for preventing RTO and avoiding spurious packet retransmission.

Ideally, it should be guaranteed that, when P_i^o is lost due to congestion, P_i^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, RTO will be incurred if and only if P_i^r is lost:

$$P(TO|CL_i^o) = P(PL_i^r) = p_l \quad (2)$$

On the other hand, an excessively small τ_{RD_i} will result in spurious packet retransmissions. Let $F(\tau_{RD_i}) = 1 - \zeta$, where $0 \leq \zeta \leq 1$. The following lemma shows that the efficiency of TCP sender, defined as the ratio of goodput to throughput, drops when ζ increases.⁴

Lemma 1:

$$\frac{G}{T} \simeq [(1 + p_l) + \zeta(1 - p_l)]^{-1} \quad (3)$$

To improve efficiency, we thus have:

$$\zeta \ll 1 \quad (4)$$

Since the retransmission timer only serves as a coarse upper bound for performing loss recovery and congestion control, it is possible for (2) and (4) to be simultaneously satisfied with τ_{RD_i} appropriately set. A closed-form expression for the optimal value of τ_{RD_i} is left as part of the future work. Nevertheless, we assume that (2) and (4) are satisfied in the following discussion.

B. Congestion Response Decision Timer

As discussed earlier, an ACK_i arriving before the expiration of CD_i shall reject the occurrence of a congestive loss for P_i^o . In other words, it is guaranteed that:

$$P(CL_i^o|PA_i(t)) = 0 \quad (5)$$

where $0 \leq t \leq \tau_{CD_i}$.

The following lemmas establish two distinct sufficient and necessary conditions for (5).

Lemma 2: $P(CL_i^o|PA_i(t)) = 0$, where $0 \leq t \leq \tau_{CD_i}$, holds if and only if

$$P(PA_i(t)|CL_i^o) = 0 \quad (6)$$

where $0 \leq t \leq \tau_{CD_i}$.

Lemma 3: $P(CL_i^o|PA_i(t)) = 0$, where $0 \leq t \leq \tau_{CD_i}$, holds if and only if

$$\tau_{CD_i} \leq \tau_u \quad (7)$$

⁴Proofs for all lemmas are relegated to appendix section at the end of this report.

where τ_u denotes a certain upper bound value and $\tau_u \geq mrtt_i$.

Therefore, we shall seek for an optimal solution for τ_{CD_i} within $[0, \tau_u]$. Consider a time period t ($0 \leq t \leq \tau_u$) after the retransmission of packet P_i , when a TCP sender is facing the decision of whether or not to activate a congestion response. The risk associated with a positive decision is that the network may not be congested (i.e. P_i^o 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_i^o 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 incurred.

To quantify the risk associated with activating a congestion response, the following metric is introduced. The expected cost of activating a congestion response, $EC_A(t)$, is defined as the product of the conditional probability of P_i^o not being lost due to congestion, given that P_i is unacknowledged, and the throughput reduction due to the activation of a congestion response, C_S . In other words,

$$EC_A(t) = P(\overline{CL_i^o}|PU_i(t)) \cdot C_S \quad (8)$$

Using the same token, the expected cost of delaying a congestion response, $EC_D(t)$, is introduced to quantify the risk associated with delaying congestion response. It is defined as the product of the conditional probability that a congestive loss and an RTO have occurred, given that P_i is unacknowledged, and the extra TCP sender throughput reduction due to RTO, C_T .

$$EC_D(t) = P(TO \cap CL_i^o|PU_i(t)) \cdot C_T \quad (9)$$

To derive $P(\overline{CL_i^o}|PU_i(t))$, we note that:

$$P(CL_i^o) = P(PU_i(t)) \cdot P(CL_i^o|PU_i(t)) + P(PA_i(t)) \cdot P(CL_i^o|PA_i(t)) \quad (10)$$

From (5), (7), and (10), we have:

$$\begin{aligned} P(\overline{CL_i^o}|PU_i(t)) &= 1 - P(CL_i^o|PU_i(t)) \\ &= 1 - \frac{P(CL_i^o)}{P(PU_i(t))} \end{aligned} \quad (11)$$

where $0 \leq t \leq \tau_u$, and $P(PU_i(t))$ can be evaluated by the following lemma.

Lemma 4:

$$P(PU_i(t)) = p_c + [1 - (1 - p_l)F(t)]p_w \quad (12)$$

where $0 \leq t \leq \tau_u$.

From (11) and (12), it follows that:

$$P(\overline{CL_i^o}|PU_i(t)) = \frac{p_w[1 - (1 - p_l)F(t)]}{p_c + p_w[1 - (1 - p_l)F(t)]} \quad (13)$$

where $0 \leq t \leq \tau_u$.

To derive $P(TO \cap CL_i^o|PU_i(t))$, following a similar process to (10) - (11), we can obtain:

$$P(TO \cap CL_i^o|PU_i(t)) = \frac{P(TO|CL_i^o)P(CL_i^o)}{PU_i(t)} \quad (0 \leq t \leq \tau_u) \quad (14)$$

From (2), (12), and (14), it follows that:

$$P(TO \cap CL_i^o | PU_i(t)) = \frac{p_c p_l}{p_c + p_w [1 - (1 - p_l)F(t)]} \quad (15)$$

where $0 \leq t \leq \tau_u$.

To derive C_S , consider the activation of congestion response, when both $cwnd$ and $ssthresh$ are set to $0.5W$. Thus TCP sender is at the congestion avoidance stage to begin. $cwnd$ is incremented by one at every RTT cycle afterwards. Let n_{CA} be the number of cycles during which the TCP sender is in the congestion slow stage with $cwnd$ less than W . Accordingly, we have:

$$n_{CA} = \lfloor 0.5W \rfloor \quad (16)$$

Therefore, when a congestion response is falsely activated, the number of packets the TCP sender misses to send is:

$$\begin{aligned} C_S &= W n_{CA} - \sum_{i=1}^{n_{CA}} (\lfloor 0.5W \rfloor + i - 1) \\ &= 0.5 \lfloor 0.5W \rfloor (\lfloor 0.5W \rfloor + 1) \end{aligned} \quad (17)$$

Finally, to derive C_T , consider the occurrence of an RTO, when $cwnd$ is reset to one and $ssthresh$ is set to $0.5W$. For simplicity, we would assume that 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 $\lceil ssthresh \rceil$, 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:

$$n_{SS} = \lceil \log_2 0.5W \rceil \quad (18)$$

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

$$\begin{aligned} C_T &= W n_{SS} - \sum_{i=1}^{n_{SS}} 2^{i-1} \\ &= \lceil \log_2 0.5W \rceil \cdot W - 2^{\lceil \log_2 0.5W \rceil} + 1 \end{aligned} \quad (19)$$

Hence, from (8), (9), (13), (15), (17) and (19), we have:

$$EC_A(t) = \frac{p_w [1 - (1 - p_l)F(t)]}{2\{p_c + p_w [1 - (1 - p_l)F(t)]\}} \times \lfloor 0.5W \rfloor (\lfloor 0.5W \rfloor + 1) \quad (20)$$

$$EC_D(t) = \frac{p_c p_l}{p_c + p_w [1 - (1 - p_l)F(t)]} \times (\lceil \log_2 0.5W \rceil W - 2^{\lceil \log_2 0.5W \rceil} + 1) \quad (21)$$

where $0 \leq t \leq \tau_u$.

Observe that:

$$\frac{\partial}{\partial t} EC_A(t) \leq 0, \quad \frac{\partial}{\partial t} EC_D(t) \geq 0 \quad (22)$$

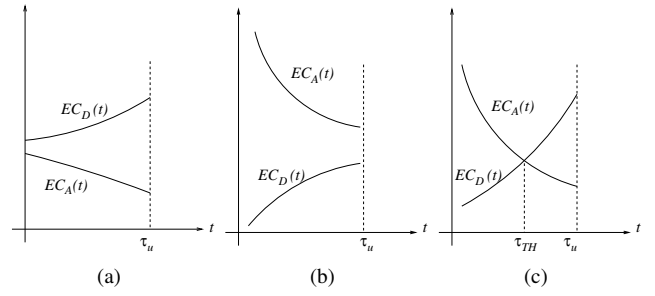


Fig. 2. Evaluation of $\tau_{CD_i}^*$: (a) Case I. (b) Case II. (c) Case III.

Thus, when the activation of congestion response is postponed further, $EC_D(t)$, which quantifies the risk associated with delaying a congestion response, increases, while $EC_A(t)$, which quantifies the risk associated with activating a congestion response, drops. When $EC_A(t)$ is greater than $EC_D(t)$, it is advantageous to set τ_{CD_i} no less than t since the operation cost of activating congestion response is $EC_A(t)$, which is larger than that of deferring it, $EC_D(t)$. The cost may drop when t increases. Similarly, when $EC_D(t)$ is greater than $EC_A(t)$, it is advantageous to set τ_{CD_i} no greater than t since the operation cost of deferring congestion response is $EC_D(t)$, which is larger than that of activating it, $EC_A(t)$. The cost may drop when t decreases.

Therefore, the evaluation of the optimal solution of τ_{CD_i} , $\tau_{CD_i}^*$, subject to the constraint $0 \leq \tau_{CD_i} \leq \tau_u$, can be divided into three cases, as depicted in Fig. 2. Case I arises when $EC_D(t)$ exceeds $EC_A(t)$ for any $t \geq 0$. The gap between the two would keep on increasing when t increases. Thus, the congestion response should be activated at $t = 0$, or set $\tau_{CD_i}^*$ to be zero. Case II arises when $EC_D(t)$ fails to catch up with $EC_A(t)$ for all t such that $0 \leq t \leq \tau_u$. Since we cannot delay a congestion response further according to the prior constraint, we have to set $\tau_{CD_i}^*$ to be τ_u . The final case arises when $EC_D(t)$ catches up with $EC_A(t)$ for some t such that $0 \leq t = \tau_{TH} \leq \tau_u$. $\tau_{CD_i}^*$ thus corresponds to τ_{TH} since $EC_A(t)$ is greater than $EC_D(t)$ prior to τ_{TH} and less than $EC_D(t)$ after τ_{TH} .

In conclusion, $\tau_{CD_i}^*$ is given by:

$$\tau_{CD_i}^* = \begin{cases} 0, & p_c > (p_l \cdot \frac{C_T}{C_S} + 1)^{-1} p_l \\ \tau_u, & p_c < \left\{ \frac{p_l C_T}{[1 - (1 - p_l) \cdot F(\tau_u)] \cdot C_S} + 1 \right\}^{-1} \cdot p_l \\ \tau_{TH}, & \text{otherwise.} \end{cases} \quad (23)$$

where $\tau_{TH} = F^{-1} \left(\frac{1}{1 - p_l} - \frac{C_T p_c}{C_S \cdot (p_l - p_c)} \cdot \frac{p_l}{1 - p_l} \right)$.

IV. TCP-NCL

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

The serialized-timer structure of the SLS model is supplemented by TCP-NCL with the NCL-RTT-Update (NRU) process for maintaining a set of RTT samples so as to estimate $F(t)$. Section IV-A describes the NRU process. Section IV-B

explains the assignment of τ_{RD_i} and τ_{CD_i} in TCP-NCL based on the RTT samples and the implementation of retransmission decision (RD) and congestion response decision (CD) timers using timestamps. The pseudocode for TCP-NCL algorithm is included at the end of this section.

A. NCL-RTT-Update (NRU) Process

The distribution of RTT is time-variant over wireless networks, where the network topology and/or loading may change over a TCP session. The time-varying property of $F(t)$ requires periodic updating of RTT samples, so that the recent RTT samples are recorded and some outdated samples are discarded. We would first define the *maximum RTT record length* as *MRRL*. A data structure *rttRcd* is defined for storing the most recent *MRRL* RTT samples. All older samples will be discarded.

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_i^o or P_i^r . Thus, some additional measures would be needed to make sure that ACK_i is for P_i^o before recording the corresponding RTT sample. We identify the time instances when P_i is first transmitted and retransmitted as $txTime[i]$ and $rtxTime[i]$, respectively. Upon the arrival of ACK_i , if the present time (identified by *now*) exceeds $rtxTime[i]$ by less than $\beta\tau_{CD_i}$ ($0 < \beta < 1$), this infers that P_i^r cannot be acknowledged within such a short interval and thus the RTT sample ($now - txTime[i]$), 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 change or handoff. When the latter occurs, it is likely that ACK_i will not be received before RT_i expires. Yet, the corresponding RTT sample can still be accurately recorded as long as it does not exceed $(\tau_{RD_i} + \beta\tau_{CD_i})$.

B. Assignment of τ_{RD_i} and τ_{CD_i}

τ_{RD_i} is set to $max(rttRcd)$, which denotes the maximum sampled RTT stored in *rttRcd*. Intuitively, this assignment is an appropriate choice, keeping spurious retransmission at a minimal level without excessively delaying fast retransmit. As will be shown by our simulation results, TCP-NCL demonstrates a very efficient usage of network bandwidth in that it almost always attains the highest connection goodput among all our compared TCP variants. This performance can hardly be achieved if either spurious retransmission constitutes a significant wastage of total throughput, or RTO frequently occurs and forces *cwnd* to be reset to one. Thus, the choice of setting τ_{RD_i} as $max(rttRcd)$ seems to be appropriate.

By (23), τ_{CD_i} is set to $min(rttRcd)$, which corresponds to the minimum RTT sample stored in *rttRcd*. It can be shown that $p_c < (p_l \frac{C_T}{C_S} + 1)^{-1} p_l$ when $p_c < p_l \ll 1$. Thus, it is reasonable to assume $\tau_{CD_i}^*$ to be τ_u or τ_{TH} , depending on the value of p_c . A conservative estimation for both τ_u and

τ_{TH} can be set as $mrtt_i$, which in turn leads to our choice of setting τ_{CD_i} as $min(rttRcd)$. A more adaptive approach capable of varying τ_{CD_i} based on the value of p_c is left as future work. The major foreseeable merit of such an approach is the increased responsiveness to network congestion. Nevertheless, as will be shown by our simulation results, TCP-NCL is quite responsive in the presence of congestive loss only.

With our approximation above, at any time instance, the expiration periods of RD/CD timers for all the RD/CD pending packets would be the same. We thus define two variables, namely, *tauRD* and *tauCD*, for keeping the present values of the expiration periods of RD and CD timers, respectively, instead of keeping a unique copy for each packet.

To fulfill the effects of the RD and CD timers, we define a list *rdPkts* for keeping the packets whose RD timers are supposedly pending, and another list *cdPkts* for keeping the packets whose CD timers are supposedly pending. When a packet P_i is first transmitted, it will be added to *rdPkts* with the timestamp $txTime[i]$ attached to it. While it is in the list, if ACK_i arrives or it has been on the list for more than *tauRD*, it will be removed from the list. In the latter case, packet P_i is retransmitted and added to *cdPkts* with the additional timestamp $rtxTime[i]$ attached to it. If ACK_i arrives or P_i has been in *cdPkts* for more than *tauCD*, P_i will be removed from it. In the latter case, the congestion control measures will be activated.

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* will be triggered only once. Thus, when *cwnd* is halved, all the present packets in *rdPkts* and *cdPkts* are exempted from causing *cwnd* to be halved again and will be added to an *ePkts* list. Every time these congestion control measures are activated, *cwnd* will be halved only if the packet causing the activation is not in *ePkts*.

All the three aforementioned lists will be cleared during initialization and upon the occurrence of an RTO.

The pseudocode of the NRU process is as follows:

```

void NRU(Packet Pi) {
if (Pi is in rdPkts
    or now <= beta*tauCD+rtxTime[i]) {
    rtt ← now - txTime[i];
    if (Sizeof(rttRcd) == MRRL)
        remove (rttRcd, oldest RTT sample);
    add (rttRcd, rtt);
    tauCD ← min(rttRcd);
    tauRD ← max(rttRcd);
}}

```

The pseudocode for realizing the RD and CD timers is as follows:

1) Invoked when sending P_i into the network for the first time.

```

txTime[i] ← now;
add (rdPkts, Pi);

```

2) Invoked when a new ACK_i arrives.

```

if (  $P_i$  is in ePkts )
    remove( ePkts ,  $P_i$  );
if (  $P_i$  is in rdPkts ) {
    NRU(  $P_i$  );
    remove( rdPkts ,  $P_i$  );
} else if (  $P_i$  is in cdPkts ) {
    NRU(  $P_i$  );
    remove( cdPkts ,  $P_i$  );
}
//Updates cwnd & send packets accordingly
    
```

3) Invoked when (now - $txTime[i] > tauRD$) for P_i in $rdPkts$, or RD_i expires:

```

remove( rdPkts ,  $P_i$  );
add( cdPkts ,  $P_i$  );
resend(  $P_i$  ); //update rtxTime[i] upon rtx
    
```

4) Invoked when (now - $rtxTime[i] > tauCD$) for P_i in $cdPkts$, or CD_i expires:

```

remove( cdPkts ,  $P_i$  );
if (  $P_i$  not in ePkts ) {
    ssthresh  $\leftarrow$  cwnd/2;
    cwnd  $\leftarrow$  ssthresh;
    append( ePkts , rdPkts );
    append( ePkts , cdPkts );
}
    
```

V. PERFORMANCE EVALUATION

In this section, we present our simulation results. Section V-A examines the performance of TCP-NCL under various combinations of MRRL and β . Section V-B examines the fairness issue of TCP-NCL via a wired network with a dumbbell topology. Section V-C compares the performance of TCP-NCL with other TCP variants under various network configurations. All the simulation experiments have been performed using Network Simulator (ns) 2.29 [9].

A. Parameter Tuning

Three different network topologies, as illustrated in Fig. 8, have been used for parameter tuning: an infrastructure-based wireless network, a multi-hop wireless network, and a wired network with a bottleneck link. We examine the performance of TCP-NCL with different combinations of MRRL and β under random packet losses, persistent packet reordering, and abrupt variations in RTT.

In the infrastructure-based wireless network, a TCP sender and a TCP receiver are connected through some wired and wireless links. Random packet errors from 0 to 15% are deliberately introduced into each wireless link. The link layer retransmission mechanism is disabled to simulate an orderly TCP flow.

In the multi-hop wireless network, a TCP sender is connected to a TCP receiver via four wireless links. Random channel error is introduced into the wireless links with a

random packet error rate ranging from 0 to 15%. The link layer retransmission is enabled to introduce persistent packet reordering. Under high channel error rate, however, local link layer retransmission cannot guarantee successful packet delivery due to the predetermined retransmission limit (set to three in this case). Consequently, TCP will be confronted with both packet reordering and random packet loss.

In the wired network, the TCP connection traverses through a bottleneck link, of which the delay takes on a random value in the interval $[50, maxDy]$ ms and is changed every 20 seconds. $maxDy$ ranges from 100 ms to 700 ms. By doing so, RTT will vary abruptly yet infrequently. We aim to test the robustness of the NRU process with this configuration.

In each test over these three topologies, a total of 20 runs, each lasting 2000 seconds and using different seeds for generating the packet error or link delay, have been performed to compute an average value and a 95% confidence interval of TCP goodput in Mbps. In order to remove the effect of the transient states, only the statistics in the last 1000 seconds in each run are collected for computing the goodput.

The simulation results over the infrastructure-based wireless network are shown in Fig. 3. When β is set to be 1, TCP-NCL suffers serious performance deterioration as channel error rate increases. This is because a TCP-NCL sender with $\beta = 1$ will treat an acknowledgment received before CD_i expires as being triggered by the originally transmitted packet and record the RTT sample accordingly. Yet, non-congestive loss is the dominant reason for packet loss over the current network topology. Thus, acknowledgments for the retransmitted packets are highly likely to arrive at the TCP sender before CD_i expires. When NRU fails to ignore these acknowledgments, the accuracy of the RTT samples would be compromised.

Nevertheless, our simulation results also reveal that significant performance improvement can be immediately achieved with β being set to a value less than 1. When β varies between 0.3 and 0.95, TCP-NCL offers consistently high connection goodput regardless the value for MRRL and the channel error rate.

The simulation results over the multi-hop wireless network are shown in Fig. 4. TCP-NCL essentially maintains identical goodput performance with MRRL and β varying within their correspondingly prescribed sets of values. Therefore, over the multi-hop wireless network, TCP-NCL demonstrates very strong robustness to variations of its preset parameters.

The simulation results over the wired network are shown in Fig. 5. For all values of MRRL, TCP-NCL senders perform slightly worse when $\beta = 1$ as compared with their counterparts with a smaller β . This should be due to the occasional mistakes in differentiating between acknowledgments for the originally transmitted packets and the retransmitted packets.

On the other hand, the value of MRRL begins to affect the performance of TCP-NCL this time. When MRRL is set to 300 or a larger value and β is less than one, TCP-NCL renders approximately the same performance.

When MRRL is set to 250 and β varies between 0.5 and 0.95, TCP-NCL attains the best connection goodput among all the tests over the wired network. As discussed previously, the wired network topology aims to simulate periodic occurrence

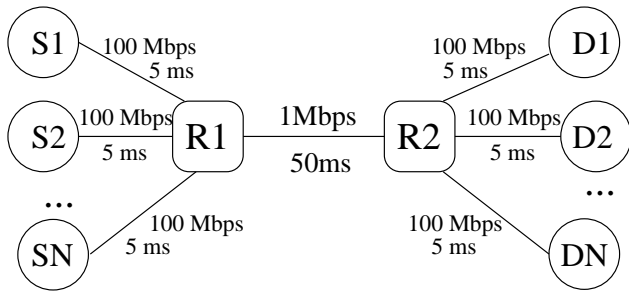


Fig. 6. Wired network with a dumbbell topology.

of abrupt variations in RTT. Whenever the latter takes place, it would be desirable for a TCP sender to replace all the RTT samples in storage with new samples as soon as possible. Apparently, a TCP-NCL sender with a small MRRL are at an advantage of attaining this.

Yet, when β is further reduced to 0.3, the goodput of TCP-NCL drops despite the merit of the small value of MRRL. This is because the excessively conservative value of β prevents TCP-NCL from recording RTT samples when there is an abrupt increase in RTT. Consequently, both RD_i and CD_i will not be able to adjust their expiration periods accordingly, thereby triggering spurious packet retransmission and congestion response.

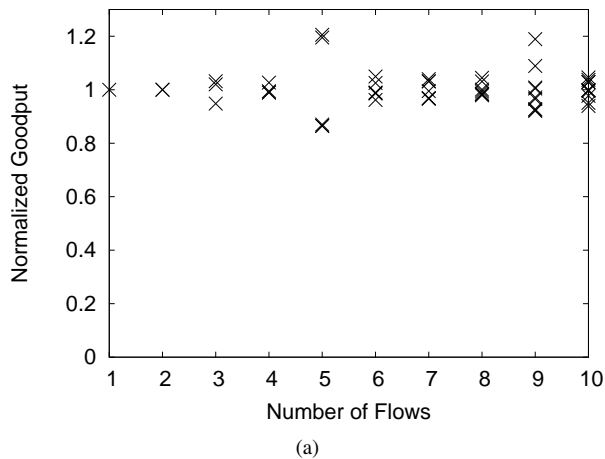
B. Fairness

In the wired network with a dumbbell topology as exhibited in Fig. 6, multiple pairs of TCP-NCL senders and receivers share an error-free in-order bottleneck link, R1-R2. Thus, all occurrences of packet loss are due to network congestion. The number of sender-receiver pairs ranges from one to ten. MRRL is set to 1000 and β is set to 0.8.

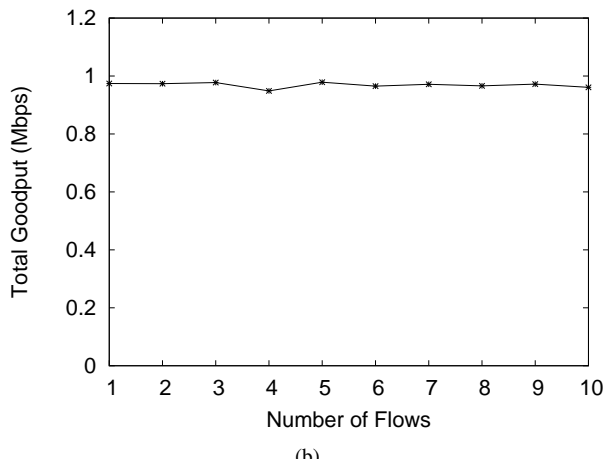
The simulation results are shown in Fig. 7. The normalized goodput of each TCP flow, which is defined as the ratio of its goodput to the average goodput among all flows, is computed and plotted. All the plotted normalized goodputs are approximately one unit. The total goodput in Megabits per second (Mbps) of all TCP flows is plotted in Fig. 7 (b). It remains at a high level despite the increase in the number of flows. Thus, the competing TCP-NCL flows are able to share the bandwidth of a bottleneck link fairly and efficiently. Thus, good responsiveness has been demonstrated in the presence of congestive loss only.

C. Performance Comparison

We compare the performance of TCP-NCL with those of some popular TCP variants, namely, RR-TCP, TCP-DCR, TCP-DOOR, TCP-PR, TCP-Veno, and TCP-W. The first four variants have performed the best in each of the four categories (state reconciliation, threshold adjustment, response postponement, and retransmission by timeout) of TCP enhancements over reordering channels as defined in [16]. TCP-Veno and TCP-W are solutions for random packet losses. The simulated network topologies include the infrastructure-based wireless network, the multi-hop wireless network, the wired network with a bottleneck link, and a MANET.



(a)



(b)

Fig. 7. Goodput performance under a wired network with a dumbbell topology.

In each test over the first three topologies, a total of 20 runs, each lasting 2000 seconds and using different seeds for generating the packet error or link delay, have been performed to compute an average value and a 95% confidence interval of TCP goodput in Mbps. In order to remove the effect of the transient states, only the statistics in the last 1000 seconds in each run are collected for computing the goodput. For the sake of a fair comparison, for TCP-NCL, MRRL is set to 1000 and β is set to 0.8, which yields an average connection goodput performance among all the combinations of MRRL and β tested in Section V-A.

In the MANET, a total of 16 mobile nodes are confined to an area of $1000\ m \times 1000\ m$ with their initial positions uniformly distributed. The data rate of a wireless interface is $2\ Mbps$. IEEE 802.11 serves as the MAC layer protocol with a transmission range of $250\ m$ and a sensing range of $550\ m$. A TCP flow is set up between two selected nodes with dynamic source routing (DSR) as the underlying routing algorithm. The movement pattern follows the random waypoint model with the maximum speed set to $maxSpeed$ and the maximum pausing time set to ten seconds. $maxSpeed$ ranges from $5\ m/s$ to $50\ m/s$. Thus, packet reordering may be incurred when path changes lead to variations in RTT, while burst loss due to

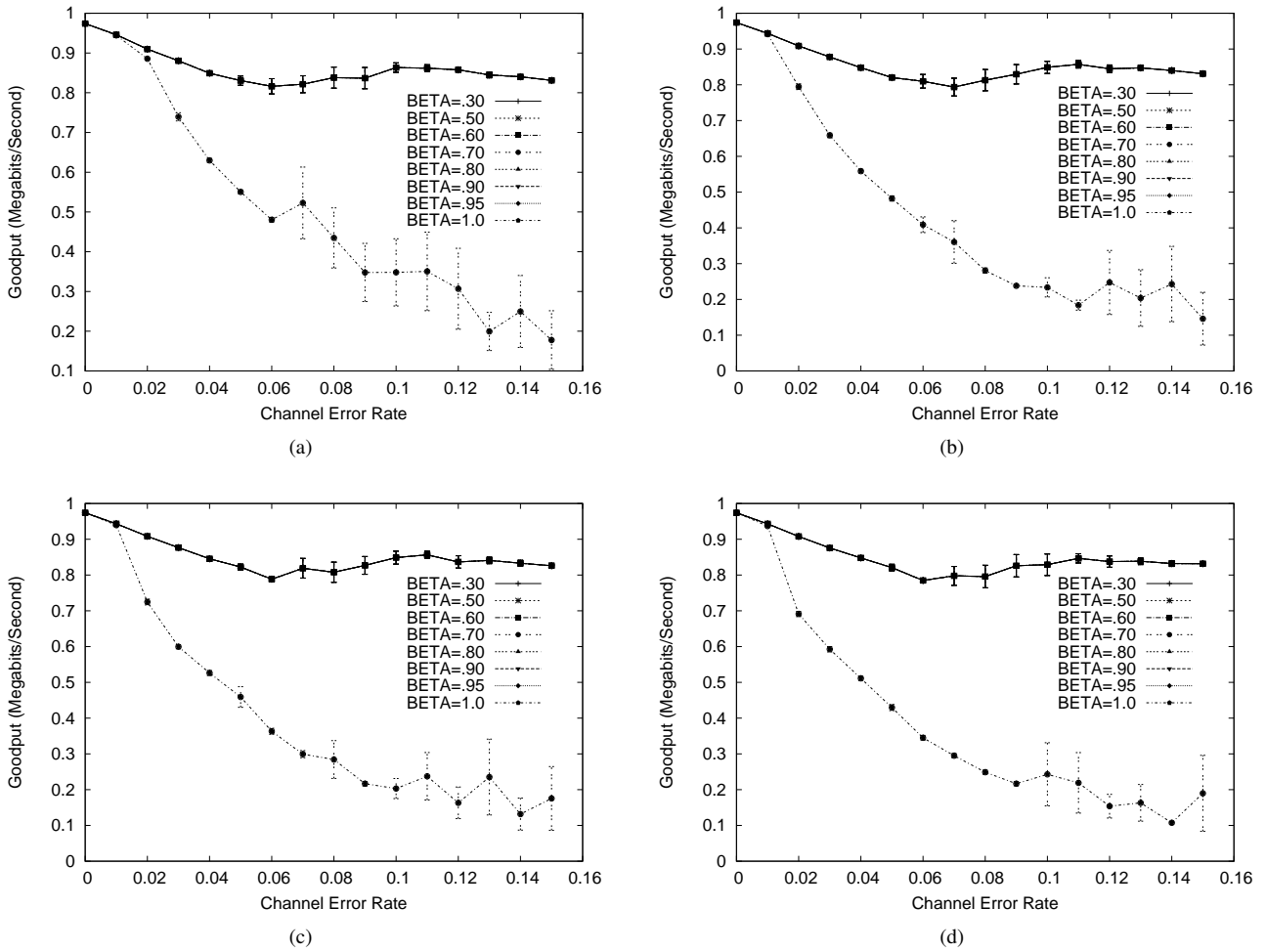


Fig. 3. Goodput performance of TCP-NCL against MRRL under infrastructure-based wireless network: (a) MRRL = 250. (b) MRRL = 500. (c) MRRL = 800. (d) MRRL = 1000.

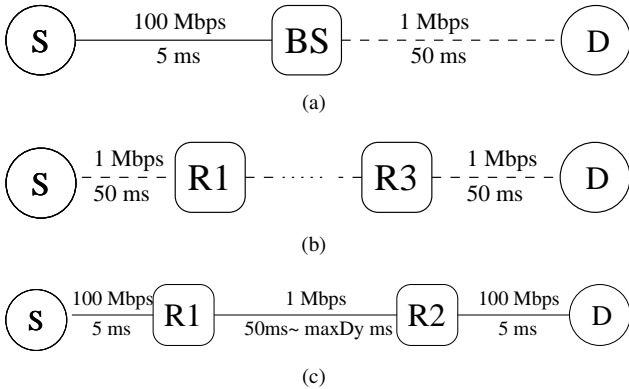


Fig. 8. The network topologies used for performance comparison: (a) Infrastructure-based wireless network. (b) Multi-hop wireless network. (c) Wired network with a bottleneck link.

temporary disconnection may result from the node movement. A total of 80 runs, each lasting 2000 seconds and using different seeds for generating the node movements, have been done for each test to compute an average value and a 95% confidence interval of TCP goodput in Mbps. Again, only the statistics in the last 1000 seconds in each run are collected for computing the goodput.

The simulation results are shown in Fig. 9. In the infrastructure-based wireless network, TCP-NCL essentially maintains a stable goodput level against channel error rates from 0 to 15%, whereas almost all of the other TCP variants experience drastic goodput decrease as the error rate increases. The only exception is TCP-W, which exhibits a relatively elegant performance deterioration. The performance of TCP-NCL should be attributable to its effectiveness in differentiating between congestion loss and random packet loss. In contrast, RR-TCP, TCP-DCR, TCP-DOOR, and TCP-PR exclude the possibility of random packet loss, resulting in under-utilization of network resources.

In the multi-hop wireless network, TCP-DCR, TCP-NCL, and TCP-PR outperform other variants under channel error rate less than 9%, thereby demonstrating robustness to persistent reordering. When the error rate further increases and random losses are no longer transparent to the transport layer, TCP-NCL performs slightly better than TCP-PR while the performance of TCP-DCR is seriously deteriorated. In the latter scenario, the installation of the CD timers again helps TCP-NCL increase the reliability for signals of congestive loss. Another interesting observation concerns TCP-DOOR. According to our obtained statistics on the number of times

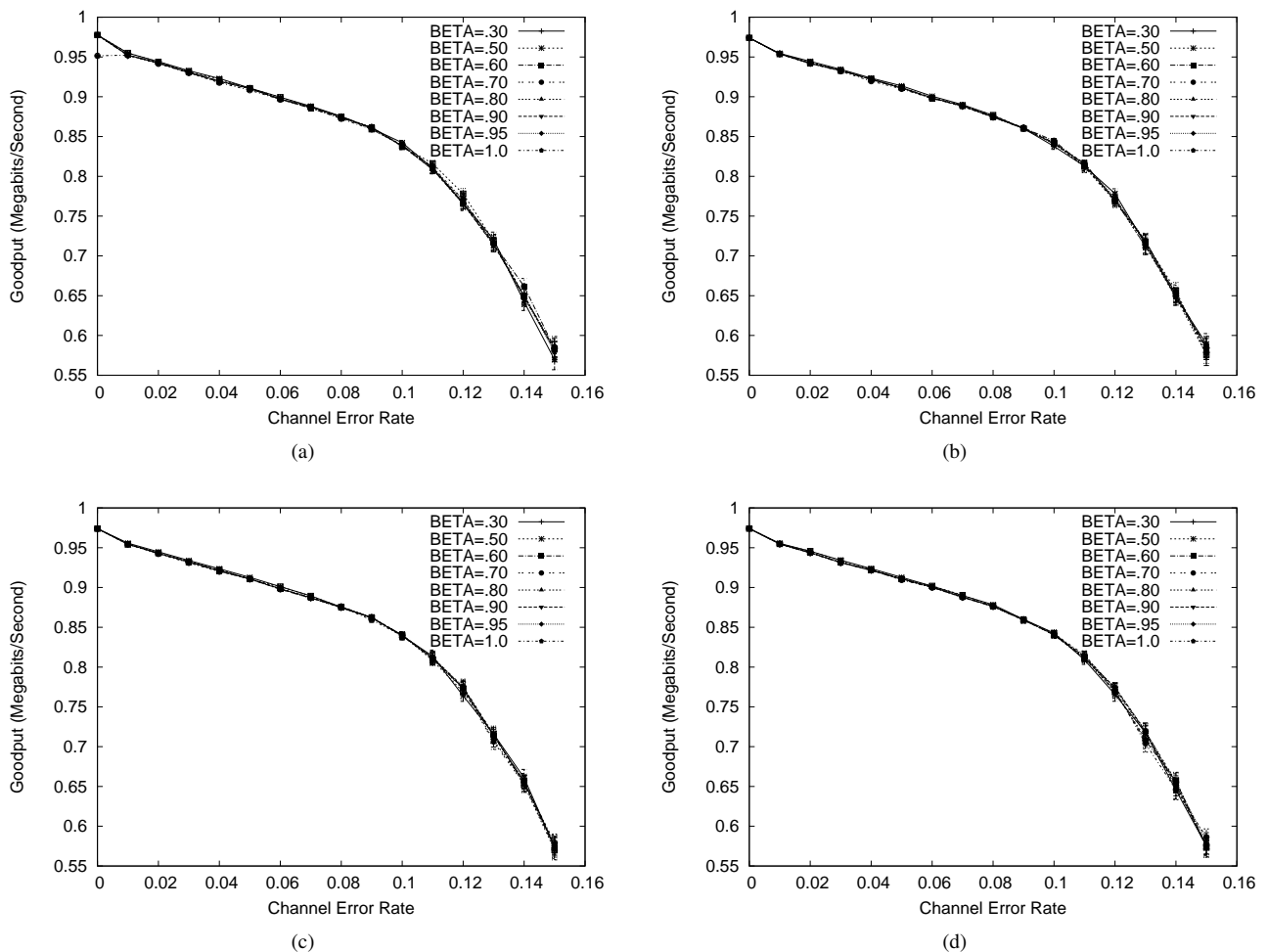


Fig. 4. Goodput performance of TCP-NCL against MRRL under multi-hop wireless network: (a) MRRL = 250. (b) MRRL = 500. (c) MRRL = 800. (d) MRRL = 1000.

fast retransmit and fast recovery are activated, TCP-DOOR freezes fast retransmit and fast recovery for almost the whole TCP session.

In the wired network topology simulating abrupt variations (and thus sometimes increases) in RTT, RR-TCP, TCP-NCL, and TCP-PR offer the best connection goodputs. Thus, with $MRRL$ and β properly set, NRU is partially verified as a comparatively robust RTT sampling process mechanism against abrupt increases in RTT. Otherwise, TCP-NCL should tend to set its timers too conservatively and probably trigger false fast retransmission and congestion response.

In the MANET, all the TCP variants offer severely deteriorated goodput performance (below 15% of the interface rate), which should largely be attributable to the occurrences of non-congestive burst loss caused by disconnections. In theory, it is possible for TCP-NCL to differentiate non-congestive burst loss from congestive loss, given that the corresponding disconnection event only causes the loss of originally transmitted packets within a window and is recovered prior to the retransmission of these packets. During that event, the retransmitted packets will probably be acknowledged in time to prevent a congestion response from being falsely activated. This explains why TCP-NCL generally offers better

performance than other variants under study.

VI. 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 analytical 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 offers significant performance improvement over a wide range of scenarios in wireless/wired networks while maintaining fairness among competing flows.

There are several possible extensions to our work, some of which are listed below:

- 1) implement a closer approximation of the STS model capable of varying τ_{CD_i} based on the value of p_c , as described in Section IV-B;
- 2) develop a distribution model for RTT so as to make TCP-NCL a memoryless algorithm, and;

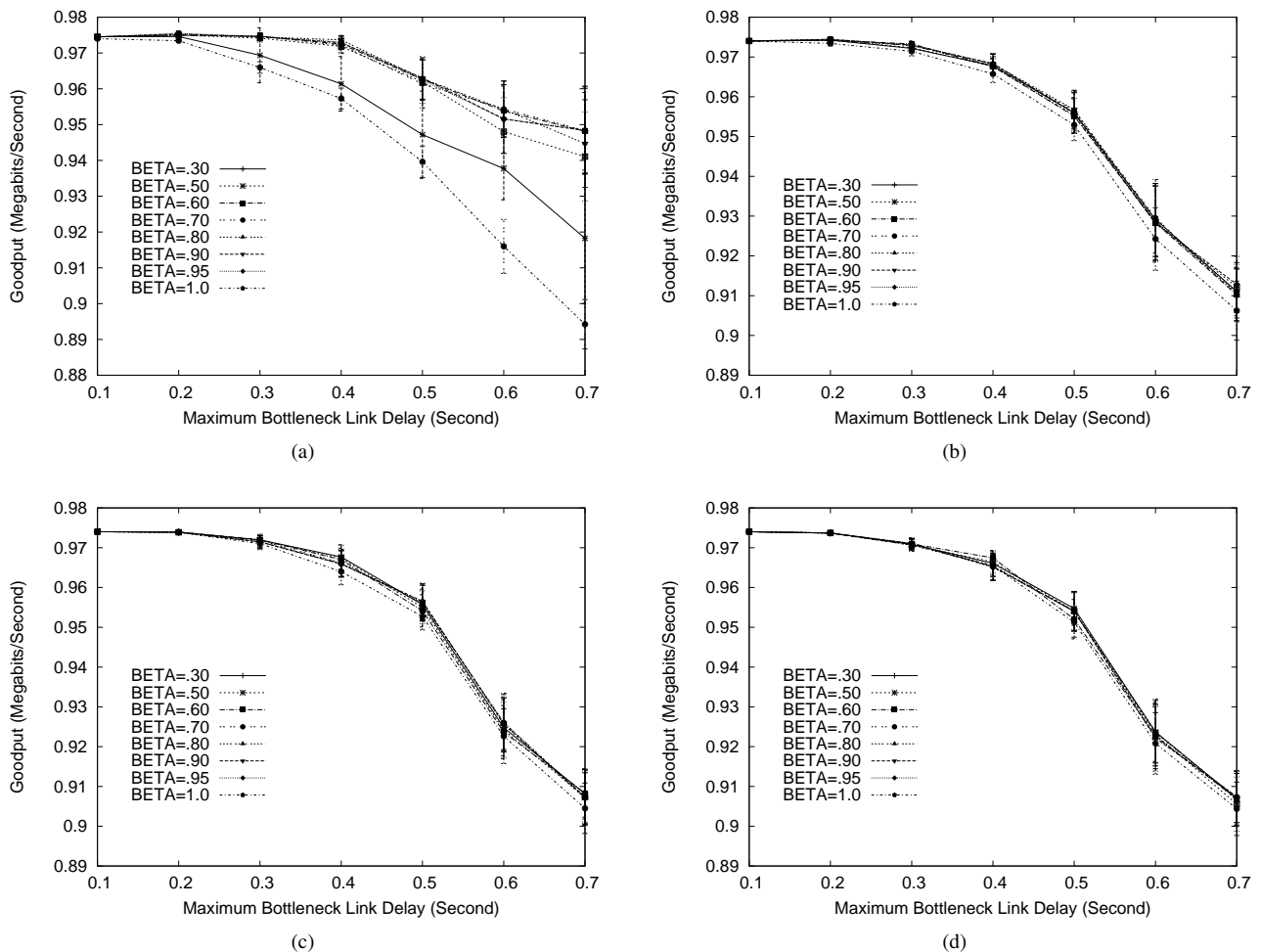


Fig. 5. Goodput performance of TCP-NCL against MRRL under wired network with a bottleneck link: (a) MRRL = 250. (b) MRRL = 500. (c) MRRL = 800. (d) MRRL = 1000.

3) implement and examine the performance of TCP-NCL on experimental testbeds.

REFERENCES

[1] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCPs Loss Recovery using Limited Transmit. *IETF RFC 3042*, Jan. 2001.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *IETF RFC 2581*, Apr. 1999.

[3] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Trans. Networking*, Vol. 5, No. 6, pp. 756-769, Dec. 1997.

[4] S. Bhandarkar and A.L.N. Reddy. TCP-DCR: Making TCP Robust to Non-Congestion Events. *Lecture Notes in Computer Science*, Vol. 3042, pp. 712-724, May 2004.

[5] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM SIGCOMM Computer Comm. Rev.*, Vol. 32, Issue 1, pp. 20-30, Jan. 2002.

[6] S. Bohacek, J.P. Hespanha, J. Lee, C. Lim, and K. Obraczka. A New TCP for Persistent Packet Reordering. *IEEE/ACM Trans. Networking*, Vol. 14, No. 2, pp. 369-382, Apr. 2006.

[7] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE J. Selected Areas in Comm.*, Vol 13, No. 8, pp. 1465-1480, Oct. 1995.

[8] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks*, Vol. 8, No. 5, pp. 467-479, 2002.

[9] K. Fall and K. Varadhan. The ns Manual (formerly ns Notes and Documentation). *The VINT Project*, 6 January 2009.

[10] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgment (SACK) Option for TCP *IETF RFC 2883*, July 2000.

[11] C.P. Fu and S.C. Liew. TCP VenO: TCP Enhancement for Transmission over Wireless Access Networks. *IEEE J. Selected Areas in Comm.*, Vol. 21, No. 2, pp. 216-228, Feb. 2003.

[12] F. Hu and N. Sharma. Enhancing Wireless Internet Performance. *IEEE Comm. Surveys and Tutorials*, Vol. 4, No. 1, pp. 2-15, Dec. 2002.

[13] V. Jacobson Congestion Avoidance and Control. *ACM SIGCOMM Computer Comm. Rev.*, Vol. 25, No. 1, pp. 157-187, Jan. 1995.

[14] A. Lahanas and V. Tsaoussidis. Improving TCP Performance over Networks with Wireless Components using 'Probing Devices'. *Int'l. J. Commun. Systems*, Vol. 15, No. 6, pp. 495-511, July/Aug. 2002.

[15] K.-C. Leung and V.O.K. Li. Transmission Control Protocol (TCP) in Wireless Networks: Issues, Approaches, and Challenges. *IEEE Comm. Surveys and Tutorials*, Vol. 8, No. 4, pp. 64-79, Fourth Quarter 2006.

[16] K.-C. Leung, V.O.K. Li, and D. Yang. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Trans. on Parallel and Distributed Systems*, Vol. 18, No. 4, pp. 522-535, Apr. 2007.

[17] J. Liu and S. Singh. ATCP: TCP for Mobile Ad Hoc Networks. *IEEE J. Selected Areas in Comm.*, Vol. 19, No. 7, pp. 1300-1315, July 2001.

[18] R. Ludwig and R.H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM SIGCOMM Computer Comm. Rev.*, Vol. 30, Issue 1, pp. 30-36, Jan. 2000.

[19] D. Mitzel. Overview of 2000 IAB Wireless Internetworking Workshop. *RFC 3002, Network Working Group*, Dec. 2000.

[20] C. Parsa and J. Garcia-Luna-Aceves. Improving TCP congestion control over Internets with heterogeneous transmission media. *Proceedings of IEEE ICNP*, pp. 213-221, Oct./Nov. 1999.

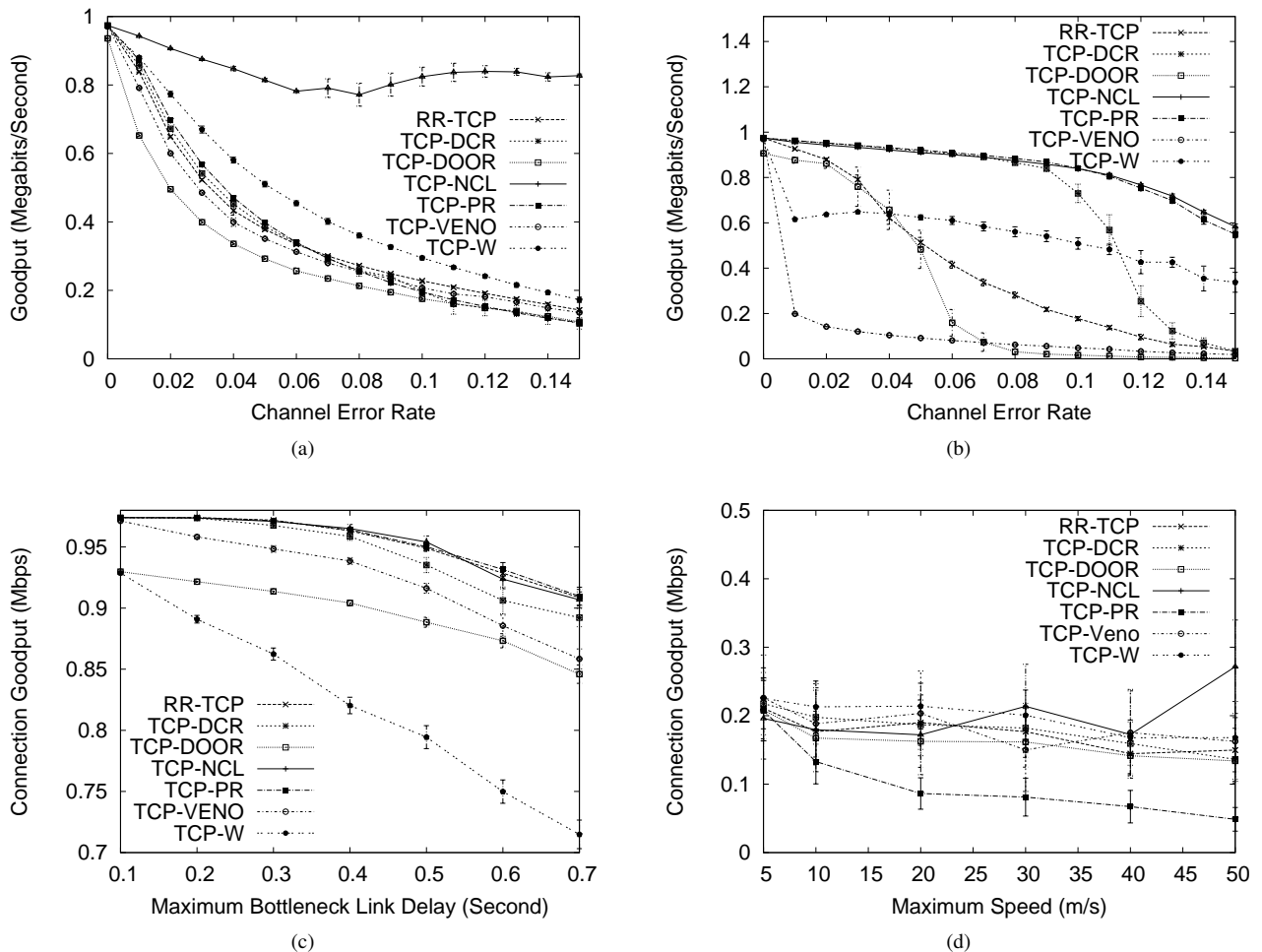


Fig. 9. Goodput performance under various topologies: (a) Infrastructure-based wireless network. (b) Multi-hop wireless network. (c) Wired network with a bottleneck link. (d) MANET.

- [21] J. Postel. Transmission Control Protocol. *Request for Comments*, RFC 793, Protocol Specification, DARPA Internet Program, Sep. 1981.
- [22] P. Sarolahti and M. Kojo. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP). *IETF RFC 4138*, Aug. 2005.
- [23] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *IETF RFC 2001*, Jan. 1997.
- [24] F. Wang and Y. Zhang. Improving TCP Performance over Mobile Ad-Hoc Networks with Out-Of-Order Detection and Response. *Proceedings of ACM MOBIHOC 2002*, pp. 217-225, June 2002.
- [25] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. Networking*, Vol. 14, No. 6, pp.1246-1259, Dec. 2006.
- [26] E. H.-K. Wu and M.-Z. Chen. JTCP: Jitter-Based TCP for Heterogeneous Wireless Network. *IEEE J. Selected Areas Comm.*, Vol. 22, No. 4, pp. 757-766, May 2004.
- [27] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. *Proceedings of IEEE ICNP 2003*, pp. 95-106, Nov. 2003.

APPENDIX

Lemma 1:

$$\frac{G}{T} \simeq [(1 + p_l) + \zeta(1 - p_l)]^{-1}$$

Proof: A fast retransmit of packet P_i will be activated upon the expiration of RD_i , given P_i is not yet acknowledged.

Thus,

$$\begin{aligned} P(RTX_i) &= P(PU_i^o(0)) \\ &= P(PU_i^o(0)|PL_i^o)P(PL_i^o) \\ &\quad + P(PU_i^o(0)|\overline{PL_i^o})P(\overline{PL_i^o}) \\ &= p_l + \zeta(1 - p_l) \end{aligned} \quad (24)$$

In addition to fast retransmit, packet retransmission can also be activated by RTO. Yet, with fast retransmit promptly activated, the occurrences of RTO will be maintained at a minimal level. Thus, the difference between throughput and goodput, without considering the protocol overhead, should mainly be attributable to fast retransmit. Accordingly, we have:

$$T \simeq [1 + P(RTX_i)]G \quad (25)$$

Hence,

$$\frac{G}{T} \simeq [(1 + p_l) + \zeta(1 - p_l)]^{-1}$$

Lemma 2: $P(CL_i^o|PA_i(t)) = 0$, where $0 \leq t \leq \tau_{CD_i}$, holds if and only if

$$P(PA_i(t)|CL_i^o) = 0$$

where $0 \leq t \leq \tau_{CD_i}$.

Proof:

Observe that:

$$P(CL_i^o|PA_i(t))P(PA_i(t)) = P(PA_i(t)|CL_i^o)P(CL_i^o) \quad (26)$$

At the same time, we note that:

$$P(PA_i(t)) \geq P(PA_i^r(0)) = (1 - p_l)(1 - \zeta) > 0 \quad (27)$$

for $t \geq 0$, and

$$P(CL_i^o) = p_c > 0 \quad (28)$$

Hence, we establish

$$P(CL_i^o|PA_i(t)) = 0 \Leftrightarrow P(PA_i(t)|CL_i^o) = 0 \quad (29)$$

where $0 \leq t \leq \tau_{CD_i}$. ■

Lemma 3: $P(CL_i^o|PA_i(t)) = 0$, where $0 \leq t \leq \tau_{CD_i}$, holds if and only if

$$\tau_{CD_i} \leq \tau_u$$

where τ_u denotes a certain upper bound value and $\tau_u \geq mrtt_i$.

Proof:

Applying the result of Lemma 2, it would be equivalent to prove: $P(PA_i(t)|CL_i^o) = 0$, where $0 \leq t \leq \tau_{CD_i}$, holds if and only if $\tau_{CD_i} \leq \tau_u$.

Applying $PA_i(t) = PA_i^o(t) \cup PA_i^r(t)$ (i.e., P_i is acknowledged if P_i^o and/or P_i^r is acknowledged), we have:

$$\begin{aligned} P(PA_i(t)|CL_i^o) &= P(PA_i^o(t) \cup PA_i^r(t)|CL_i^o) \\ &\leq P(PA_i^o(t)|CL_i^o) + \frac{P(PA_i^r(t))}{P(CL_i^o)} \end{aligned} \quad (30)$$

Noting that $P(PA_i^o(t)|CL_i^o) = 0$ since the first transmitted packet cannot be acknowledged if lost due to congestion, we obtain:

$$P(PA_i(t)|CL_i^o) \leq \frac{P(PA_i^r(t))}{P(CL_i^o)} \quad (31)$$

At $t=0$, $P(PA_i^r(0)) = 0$ since the retransmitted packet cannot be acknowledged immediately upon its transmission. Thus,

$$P(PA_i(0)|CL_i^o) = 0 \quad (32)$$

By assumption, $P(PA_i(t)|CL_i^o)$ continuously monotonically increases with time t and $\lim_{t \rightarrow \infty} P(PA_i(t)|CL_i^o) > 0$. Thus, there exists a value, denoted as τ_u , such that $P(PA_i(t)|CL_i^o) = 0$ for $t \leq \tau_u$ and $P(PA_i(t)|CL_i^o) > 0$ for $t > \tau_u$. The result of the lemma follows.

Moreover, since $P(PA_i(mrtt_i)|CL_i^o) \leq \frac{P(PA_i^r(mrtt_i))}{P(CL_i^o)}$ by (31) and $PA_i^r(mrtt_i) = 0$, we have $P(PA_i(mrtt_i)|CL_i^o) = 0$. Thus, $mrtt_i \leq \tau_u$. ■

Lemma 4:

$$P(PU_i(t)) = p_c + [1 - (1 - p_l)F(t)]p_w$$

where $0 \leq t \leq \tau_u$.

Proof:

$$\begin{aligned} &P(PU_i(t)) \\ &= P(PU_i^o(t) \cap PU_i^r(t)) \\ &= P(PU_i^o(t) \cap PU_i^r(t)|\overline{PL_i^o})P(\overline{PL_i^o}) \\ &\quad + P(PU_i^o(t) \cap PU_i^r(t)|PL_i^o)P(PL_i^o) \end{aligned} \quad (33)$$

Observe that:

$$\begin{aligned} P(PU_i^o(t) \cap PU_i^r(t)|\overline{PL_i^o}) &\leq P(PU_i^o(t)|\overline{PL_i^o}) \\ &= 1 - F(t + \tau_{RD_i}) \\ &\leq \zeta \end{aligned} \quad (34)$$

for $t \geq 0$.

From (4) and (34), we have:

$$P(PU_i^o(t) \cap PU_i^r(t)|\overline{PL_i^o}) \simeq 0 \quad (35)$$

for $t \geq 0$.

On the other hand, noting that:

$$\begin{aligned} &P(PU_i^o(t) \cap PU_i^r(t)|PL_i^o) \\ &= P(PU_i^r(t)|PL_i^o) - P(PU_i^r(t) \cap PA_i^o(t)|PL_i^o) \end{aligned} \quad (36)$$

and

$$P(PA_i^o(t) \cap PU_i^r(t)|PL_i^o) \leq P(PA_i^o(t)|PL_i^o) = 0 \quad (37)$$

we have:

$$P(PU_i^o(t) \cap PU_i^r(t)|PL_i^o) = P(PU_i^r(t)|PL_i^o) \quad (38)$$

From (33), (35), and (38), it follows that:

$$\begin{aligned} P(PU_i(t)) &\simeq P(PU_i^r(t) \cap PL_i^o) \\ &= P(PU_i^r(t) \cap CL_i^o) + P(PU_i^r(t) \cap NL_i^o) \end{aligned} \quad (39)$$

for $t \geq 0$.

From (1), (6), and (39), it follows that:

$$P(PU_i(t)) \simeq P(CL_i^o) + P(PU_i^r(t)) \cdot P(NL_i^o) \quad (40)$$

where $0 \leq t \leq \tau_u$.

Observe that:

$$\begin{aligned} P(PU_i^r(t)) &= P(PU_i^r(t)|PL_i^o)P(PL_i^o) \\ &\quad + P(PU_i^r(t)|\overline{PL_i^o})P(\overline{PL_i^o}) \\ &= p_l + [1 - F(t)](1 - p_l) \end{aligned} \quad (41)$$

Hence, substituting (41) into (40) and rearranging, we obtain:

$$P(PU_i(t)) = p_c + [1 - (1 - p_l)F(t)]p_w$$

where $0 \leq t \leq \tau_u$. ■