TCP Wave Resilience to Link Changes
A New Transport Layer Approach Towards Dynamic Communication Environments

A. Abdelsalam, M. Luglio, C. Roseti and F. Zampognaro
University of Rome"Tor Vergata", Via del Politecnico 1, 00133, Rome, Italy

Keywords: TCP Wave, Hybrid Networks, Error Recovery, Handover.

Abstract: In case of hybrid access networks, the selected link can suddenly change leading to a vertical handover. A running TCP connection can experience a combination of three potential effects: a bandwidth change, a latency change and an outage interval due to handover operations. In this context, we address a detailed performance analysis of standard TCP, in comparison with a new TCP-based protocol, namely TCP Wave, which mainly replaces traditional window-based transmission paradigm with a proactive burst transmission. Previous studies demonstrate TCP Wave capability to quickly adapt its rate to dynamic link variations, so that its application in the target scenario is considered worth. Performance assessment is carried out using the Network Simulator (Ns-3) over a different set of possible configurations in terms of handover direction, outage duration and selected transport protocol. The achieved results confirm the TCP Wave efficiency in dealing with link changes and provide a high number of interesting hints for drawing requirements of the TCP-based transport protocols operating on future dynamic networks.

1 INTRODUCTION

The next generation networks will simultaneously leverage on connectivity offered by multiple access technology, including satellite, wired networks and mobile terrestrial networks. In this view, as fostered in an ETSI specification (ETSI, 2015), an Intelligent User Gateway (IUG) is installed at the user premises, with the aim to route traffic among the available access technologies according to performance-related policies, or to increase resiliency against potential interruption of service on one or more links. At the other end of the hybrid access network, an Intelligent Network Gateway (ING) is installed to complement operations executed by the IUG and provide interconnection to the public network. The logic adopted in both IUG and ING for applying routing rules can be tailored to the user needs, but in general the criteria adopted is one or a combination of the following:

- QoS/QoE requirements of the processed traffic and running applications;
- capabilities and availability of the access links;
- policies and contracts agreed between operator and subscriber.

The latter two criteria are defined within a process of optimization of the overall available resources, which is managed by an "*orchestrator*" agent, having visibility on the resource allocation status of the available links. It is then possible that the endpoints are unaware of vertical handovers within the access networks, while applications are running and transport protocol end-to-end connections are established. Running TCP/IP flows are mainly affected by three macro-effects:

- a possible change of the available bandwidth;
- a possible change on the end-to-end-latency;
- a possible disconnection time to execute the vertical handover.

These effects, or a sub-set of them, are not limited to the target network architecture, but can be easily faced also in other communication scenarios, e.g.,: i) mobile ad-hoc networks (MANET), where node mobility affects the actual end-to-end rate (depending on the number of crossed hops) and leads to varying network link characteristics together with frequent connection losses (Yuwono, 2013); ii) hard handovers supported in LTE networks, in which a strong signal degradation is experienced upon the *eNode* change, corresponding to a so-called Time To Trigger (TTT) (Zhang et al., 2012). Therefore, it

Abdelsalam, A., Luglio, M., Roseti, C. and Zampognaro, F.
TCPWave Resilience to Link Changes - A New Transport Layer Approach Towards Dynamic Communication Environments.
DOI: 10.5220/0005966700720079
In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 1: DCNET*, pages 72-79
ISBN: 978-989-758-196-0

is evident that newer proposals for variants of traditional TCP/IP protocols must take into account the handling of disconnection periods and/or sudden variations of the link physical characteristics. Inefficiencies are expected for the traditional TCP protocol (Allman et al., 2009), which has been designed assuming communication conditions typical of congested wired networks. In fact, the window-based transmission paradigm has been designed considering losses and/or latency changes as a congestion signal. Clearly, such assumptions may result false in all the aforementioned scenarios. Other TCP variants, such as MPTCP (Bhat and Talmale, 2014) and SCTP (Lee et al., 2006), designed to efficiently tackle multi-homed terminals do not fulfill target requirements for a threefold reason: they need modifications on all the endpoints to manage control sessions, they mainly aim to exploit all the available interfaces in parallel without any centralized optimization scheme, and they focus on endpoints handovers.

With respect to other TCP variants, the recently introduced TCP Wave (Abdelsalam et al., 2015) replaces the traditional window-based transmission with a burst-based transmission, where packet sending process, is decoupled by the ACKnowledgement (ACK) reception timing. In this way, both transmission rate and packet retransmission scheduling can be managed irrespectively from the experienced latency. An exhaustive performance analysis in the described scenario is proposed, through an experimental simulation campaign, with reference TCPs and TCP Wave. The tests reported in this paper aim to reproduce general conditions of a vertical handover between links with quite different physical characteristics, to assess TCPs behavior and to analyze resulting performance.

2 TCP WAVE DESCRIPTION

TCP Wave (Abdelsalam et al., 2015) is the latest vesion of a protocol family started by TCP Noordwijk (Roseti et al., 2010)(Patriciello et al., 2015), which was designed to exclusively operate between I-PEP agents at the edge of satellite links. TCP Wave kept the "burst transmission" concept of TCP Noordwijk, upgraded with adaptivity to different architectures and to broader target application scenario. TCP Wave reviewed all algorithms and methods with the aim to become a valid alternative to standard TCP flavors in every broadband communication environment (including terrestrial-only links), taking into account the requirements of nowadays traffic characteristics and network evolution. The new design principles include the guarantee of optimal performance under varying network and physical conditions, with adaptation to wide bandwidth and/or delay changes, acceleration of small transfers, quick achievement of high throughput and efficient handling of transmission errors, also in presence of link disconnections, interwork with ACKs as generated by any standard TCP receiver, to guarantee protocol interoperability in any current and future communication scenario.

Loss recovery is compatible with the classic TCP recovery mechanisms, referenced hereafter as "standard TCP" (Allman et al., 2009), so that the main indication of lost packet is indeed the reception of duplicate ACKs (*DupAcks*), which are generated by standard TCP receivers upon out-of-sequence received packets. Similarly to standard TCP, TCP Wave performs a packet retransmission when receiving three DupAcks referred to the same packet but, in contrast, TCP-Wave sender triggers a new *Fast Retransmission (FR) algorithm*, to retransmit the packet assumed lost, without affecting the normal process of burst transmission. To guarantee the stability of this approach, TCP Wave also inherits and updates the *Retransmission Time Out (RTO)* mechanism of standard TCP implementation. The RTO is updated taking as input RTT samples internally calculated by a function working on a burst-basis (one sample per burst) and, when RTO expires, the trigger is intercepted by TCP Wave to perform specific timeout operations. Both the new FR and RTO algorithms, following the principles of the TCP Wave burst-based transmission, rely on an internal timer (Roseti et al., 2010) instead of using ACK reception as clock, making the actual response to losses substantially different from standard TCP. Error recovery algorithms of standard TCPs are widely covered in literature (Allman et al., 2009)(Paxons et al., 2011), whereas the description of TCP Wave error recovery methods was not adequately addressed in previous works (Abdelsalam et al., 2015). Therefore, the next sub-sections provide details on the TCP Wave error recovery algorithms.

2.1 Fast Retransmission (FR) Algorithm

TCP Wave FR algorithm is designed to recover single/multiple packet losses without incurring into RTO expiration. Upon receiving three *DupAcks*, the lost packet is immediately retransmitted, while burst transmission scheduling is kept unaltered, except the reduction by 1 packet to the next scheduled burst, to keep consistent the actual overall transmission rate. This FR algorithm does not limit in practice the maximum number of packet losses that can be recovered

without triggering RTO, because the uninterrupted transmission of "fresh" packet bursts allows generation of all the *DupAcks* needed to report multiple consecutive losses. To opposite, standard TCP during fast retransmit is able to generate just a limited number of "new" packets according to the Congestion Window (*cwnd*) value, increasing the possibility to experience an RTO before the recovery of all the consecutive losses. The retransmitted packet successfully delivered will generate a cumulative ACK covering packets transmitted on multiple bursts. When this happens, TCP Wave sender performs only a single ACK-based statistic update, accounting the cumulative ACK to complete computation related to the ACK train partially received before the loss. As a result, every cumulative ACK will produce at most 1 new RTT sample.

Finally, TCP Wave disables FR during the whole RTO recovery phase, that is until the the transmission reaches the last packet ACKed before the RTO expiration. The rationale relies on the assumption that an RTO occurs when there is a serious problem in the network and than it is considered preferable to incur in a further RTO rather than to recover individual packets.

2.2 Retransmission Time Out (RTO)

Retransmission Timeout (RTO) indicates how long the TCP connection must wait for an ACK related to a transmitted packet, before performing "reset/recovery" actions. This mechanism is tightly coupled with the FR and DupAck mechanism to ensure a robust continuation and management of the connection. Due to the dynamic changes in the network and to the different ways in which receiver can manage the generation of ACKs (cumulative, delayed, etc.), it is difficult to determine the ideal value for RTO timer. Too long RTO is bad for timely determining a major network outage, while too short RTO can lead to frequent unnecessary timeouts. (Paxons et al., 2011) outlines the algorithm implemented by standard TCP protocols to dynamically calculate RTO value. It depends on the measured RTT samples, their variation and it is lower bounded to at least 1 s. This algorithm is implemented by all the current TCP variants and it has been adapted within TCP Wave.

TCP Wave calculates the RTT samples once each transmitted burst, as the time interval between the start of the first packet of a burst and the time of the reception of the first ACK of the corresponding ACK train. When a lost packet is retransmitted through FR algorithm, the resulting RTT value is inflated since RTT is measured with respect to the first packet transmission time. Obviously, in this case, the RTT increase is not a symptom of congestion. Differently from standard TCP, all the RTT measurements including those calculated during retransmissions, are sent to the underlying RTT estimator, which calculates the round-trip time variation (RTTVAR) and sets the RTO value accordingly. This expedient ensures that the estimated RTO value is always consistent with the RTT value seen by TCP Wave, also during retransmission, avoiding unnecessary RTO expirations. Furthermore, since TCP Wave considers an RTO expiration as an indication of a serious problem in the network, either severe congestion or even temporary disconnection, the sender behavior during the *RTO recovery* is quite conservative. Once RTO recovery is triggered, the sender resets the $BURST$ size and the $TxTime$ to the initial values $BURST_0$ and $TxTime_0$ respectively, assuming that network conditions are not well known yet. Then transmission restarts in bursts from the latest ACKnowledged packet before the RTO expiration. All ACK types (normal and duplicate) are monitored during RTO recovery, in order to skip some bursts, if possible, if the ACK is cumulative (i.e., jump in the transmitted sequence number), while $BURST$ size and $TxTime$ are dynamically updated upon the reception of regular ACK trains. Many interactions between FR and RTO algorithm occur. For instance, during RTO recovery, data can be lost again, or , during a drastically changed phase RTO could expire. A remarkable case is when a retransmitted packet is lost again: the sender will receive a quite large number of *DupAcks* associated to it. However, this will not trigger any further retransmission, because in this condition it is not possible to make assumptions on the successful or failed reception of the retransmitted packet. Without receiving a new ACK higher than the current *DupAck*, RTO will expire and TCP Wave will trigger the RTO recovery algorithm. This behavior is consistent with the need of RTO expiration in a situation where a packet is lost twice, which can occur in bad or drastically changed network conditions.

3 PROTOCOL VALIDATION AND PERFORMANCE ASSESSMENT

A baseline version of TCP Noordwijk was implemented in the Network Simulator Ns-3 (Ns-3, 2016) as described in (Patriciello et al., 2015). This version was then used as reference to implement the newest TCP Wave version, including the error recovery algorithms herein discussed. All the TCP Wave algorithms were based on standard calls available in Ns-3 ("*TcpSocketBase*"), closely resembling the

Linux O.S. (Linux, 2015) TCP state-machine model. Therefore, the burst transmission paradigm and all error recovery methods and related operations, do not require any modification for receiver endpoints and TCP sender state-machine, making TCP Wave integration easily applicable to the target real networks.

3.1 Simulation Setup

A setup file written in C++ was used to configure the sender/receiver nodes and network topology for the transfer of application traffic in Ns-3. The reference simulation scenario is shown in fig. 1.

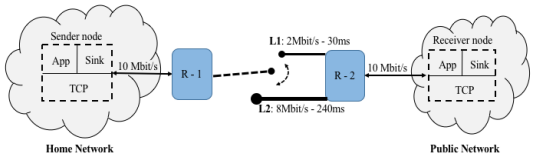

Figure 1: Ns-3 Simulation setup.

A sender/source node is connected to an intermediate Router 1 (*R-1*) through a point-to-point local link. The source node is configured to generate infinite amount of raw data to the transport protocol socket exploiting either TCP Wave or other versions of TCP that are available in the current release of Ns-3 (v3.23): TCP Tahoe, TCP Reno, TCP NewReno, and TCP Westwood+. The data flow is terminated at a Sink agent configured in the receiver node, directly connected through a point-to-point local link to Router 2 (*R-2*). *R-1 – R-2* represents the bottleneck point-to-point link, that can be configured with either a L1 configuration (2 Mbit/s, low latency of 30 ms) or L2 configuration (8 Mbit/s, high latency of 240 ms), as possible consequence of network topology change. In both cases the bottleneck buffer size is assumed equal to 100 TCP/IP packets, which is the default value for the configuration of a queue in Ns-3. All the configuration values have been selected with the aim to reproduce a realistic scenario, while at the same time offering challenging conditions for the transport layer, by changing the bottleneck link configuration (L1 or L2).

During the simulation run, handover events are reproduced by triggering a 100% bidirectional loss event for a so-called "*handover time*" (*HT*). This represents the lower layer switching experienced upon a hard vertical handover that is defined in the reference scenario. Simultaneously, the *R-1–R-2* link configuration is switched from L1 to L2 configurations or *vice-versa*. For a preliminary validation and performance evaluation during this type of handover, the case of an *HT* = 2.0 s is considered as baseline. Such an *HT* value has been selected since it is comparable and always larger than the RTO estimation in case of the link with the highest latency (L2), and then it allows the triggering of all the envisaged loss recovery mechanisms. *TCP NewReno* is considered as reference standard TCP. In the last part of the paper, a general overview comparing different values for the *HT* (0.5 s, 2 s and 10 s) and using all available TCPs is provided for completeness. As a general criterion to schedule the handover event in the simulation, we consider a time where the target transport protocol under evaluation shows the maximum eligible rate on the current link (which can be different based on protocol and link characteristics). The rationale is to analyze the recovery dynamics from the steady-state achieved on the "old" link until the restore of the steady-state on the "new" link.

The initial setup for standard TCP is a MTU of 1 kbyte, initial *cwnd* of 1 MSS; threshold for retransmission = 3 *DupAcks*. In addition, the initial values considered for TCP Wave, are a $BURST_0$ of 10 TCP packets, a $TXTime_0$ of 0.5 s and a β of 0.35 s.

3.2 Analysis of TCP Loss Management

In the first test a handover from L1 to L2 at 30 s is triggered, when both TCP NewReno and Wave are in steady-state. Thus, the sender will not receive any ACK from 30 s to 32 s, since *HT* is set to 2 s. In case of TCP NewReno, the sender stops transmission completely at the exact moment it experiences the link outage. This is because, in standard TCP, transmission is *ACK- clocked*: *cwnd* shifts are performed upon new ACK reception. The complete packet transmission and ACK reception timing, as experienced by the TCP NewReno sender, are shown in fig. 2.

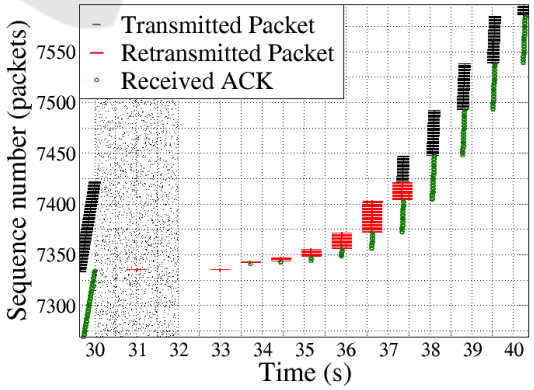

Figure 2: NewReno recovery after handover (L1 to L2).

After the link outage at 30 s, the sender waits for the RTO expiration to perform a retransmission. However, in this scenario, the RTO expires during the outage interval, causing a further loss of the re-

transmitted packet. Thus, the sender doubles its RTO time, as defined in the exponential back-off algorithm (Paxons et al., 2011), and waits for the next RTO expiration. Finally, once the second RTO expires after 3 s, the channel is available. The sender resets *cwnd* to 1 packet and goes into Slow Start phase (with *cwnd* doubled every RTT), until overcoming the Slow Start Theshold (ssthresh), which has been set to 1/2 of the amount of bytes in flight at the RTO expiration. Definitively, the overall RTO recovery strategy of NewReno increases the outage interval unnecessarily, since the sender is always waiting for either an ACK or the RTO expiration to trigger actions.

Fig. 3 shows TCP Wave behavior in the same conditions. After disconnection at 30 s, TCP Wave

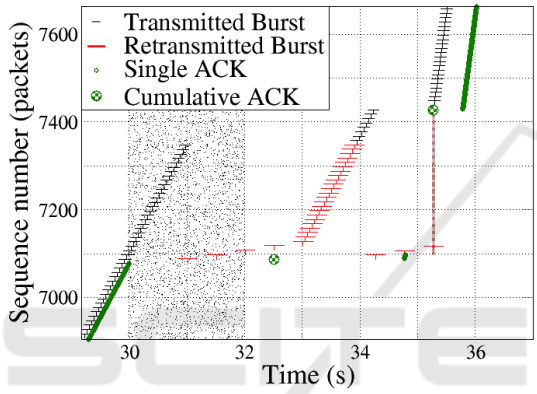

Figure 3: TCP Wave recovery after handover (L1 to L2).

continues burst transmissions, irrespective of reception of "new" ACK trains. This behavior occurs until RTO expires after about 1 s from the disconnection time. Then, TCP Wave sender starts the retransmissions from the first unacknowledged packet, using the initial burst setting: $BURST_0$ and $TxTime_0$. However, in the simulated scenario the first 2 bursts are lost again since they are retransmitted during the HT period. Despite further losses, burst-transmission paradigm allows to continue the burst sending. Then, the third burst is retransmitted on the new link allowing the reception of the first cumulative ACK at 32.75 s. This cumulative ACK corresponds to a part of the last burst transmitted before the handover event, which was only partially acknowledged before due to the link outage. This cumulative ACKs causes a reset of RTO timer, avoiding further immediate RTO expirations. As a cumulative ACK of a partial burst, its reception is used as well to update TCP Wave sender statistics. In practice, average RTT is not impacted since it was already registered upon the first ACK of the target ACK train, producing a value consistent with the old link latency. Instead, ACK dispersion sample results higher than currently computed average RTT (around 60 ms, since computed on the old link), and under this condition the new sample is discarded. As a consequence, $TxTime$ is still similar to that used on the old link. Therefore, at 33 s retransmission of burst continues at an updated rate close to the L1 bandwidth. In this phase, also "fresh" packets are transmitted just after the retransmission of all those lost during outage (at 33.8 s and exceeding sequence number 7400). Again, at 34.25 s, RTO expires due to loss of the first two bursts sent during *HT* period. The RTO recovery is then re-executed, starting-off from packets acknowledged at 32.75 s by the cumulative ACK. Differently from first RTO, retransmissions are all successfully received, generating ACK trains such as the one at 34.75, which allows a first update of the ACK-based statistics for the new link. However, the effect of such an update is not applied on the next burst, since it was already scheduled. Afterwards, two main effects are evident: 1) a cumulative ACK referred to the large number of burst proactively transmitted with success between the two RTO expirations is received, leading a huge "jump" in the transmission sequence number; 2) $TxTime$ is updated according to ACK-based statistics previously computed. The latter effect allows to suddenly achieve the maximum rate allowed on the new link, as confirmed in section 3.3. Comparing packet sequence number of fig. 2 and fig. 3, it is possible to notice that after the activation of the new link at 32 s, TCP NewReno transmits about 250 packets over 8 s, while TCP Wave transmits more than 500 packets in less than 4 s, obtaining an evident performance improvement.

In the second simulation setup, the link handover is from L2 to L1. In this event, the new link is characterized by a bandwidth-delay product much smaller than the old one, increasing the risk to overload the new link during the recovery operations. For TCP NewReno, the handover occurs at 300 s, instead than 30 s, since it requires longer time to achieve the maximum allowed rate (8 Mbit/s) due to the large latency affecting the *cwnd* increases. On the other hand, for TCP Wave the handover time is scheduled at 30 s as before, since it is able to achieve the maximum rate upon the reception of the first ACK trains (1-2 RTT), which allow a proper update of the $TxTime$ variable.

Fig. 5 shows the TCP NewReno behavior when recovering from the outage event at 300 s. As in the previous handover case, the stop of ACK reception implies the corresponding interruption on sending new packets, waiting for the RTO expiration. RTO expires at 301 s when outage is still in place, so that retransmission of the first unacknowledged packet fails. Therefore, RTO value is doubled and it ex-

pires again as in the previous test, at 303 s. As usual, TCP NewReno reacts to RTO resetting *cwnd* to 1 and *ssthresh* to half of the bytes in flight when first RTO was triggered. In this second attempt, the retransmission of the first unacknowledged packet is successfully performed, leading to a twofold effect: 1) it triggers a cumulative ACK accounting all packets received before handover (which in turn causes a jump in the transmission sequence number); 2) it causes a doubling of *cwnd*, since Slow Start algorithm is running. The latter effect recursively occurs until current *cwnd* overcomes the *ssthresh* value. In the target scenario, *ssthresh* is set to a value proportional to the bandwidth delay product of the old link (L2), which is much higher than the one of the new link (L1). As a consequence, TCP NewReno performs an exponential increase of *cwnd* on RTT basis, which leads at exceeding the bandwidth-delay product of the new link, causing a significant bottleneck buffer overflow and consequently the loss of multiple TCP packets.

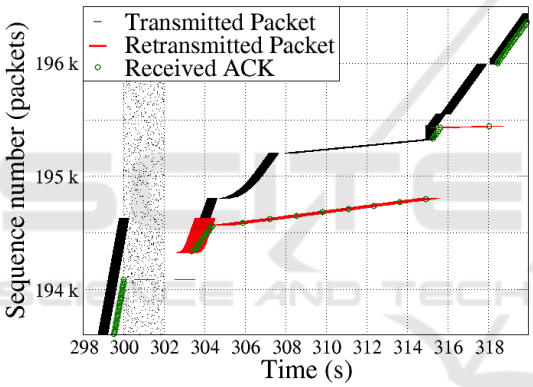

Figure 4: NewReno recovery after handover (L2 to L1).

Then, TCP NewReno enters in Fast Retransmit and Fast Recovery phase (Allman et al., 2009), where lost packets are recovered one by one as soon as the corresponding *DupAcks* are received. As a result, TCP NewReno is able to recover 1 lost packet per RTT. This phase covers the time interval from about 305 s to 315 s in fig. 4. Fast Retransmit and Fast Recovery envisages the halving of the current cwnd, although it can be temporary inflated upon the reception of DupAcks. This process allows the transmission of new packets in parallel to the retransmissions according to *cwnd* inflation/deflation. At 315 s, when all the losses due to the buffer overflow are recovered, TCP NewReno resets cwnd to the value computed at the beginning of the Fast Retransmit and Fast Recovery. Again, such a value is still too big with respect to the new bandwidth delay product, then causing a further buffer overflow, with a lower number of losses. In definitive, the new steady-state is recovered after

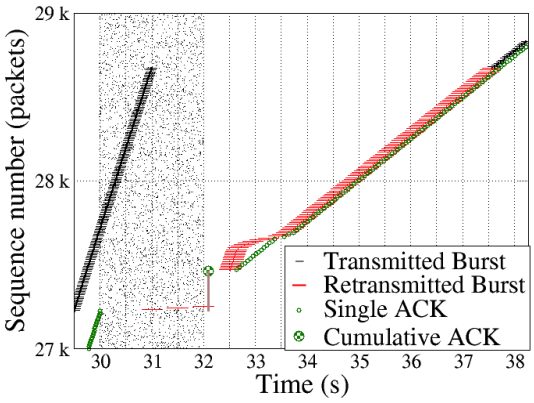

Figure 5: TCP Wave recovery after handover (L2 to L1).

about 17 s from the activation of the new link.

TCP Wave behavior during the same handover scenario is reported in fig. 5. As observed in the previous handover scenario (fig. 3), TCP Wave transmissions continues also after the old link outage, because it is not dependent on the ACK reception. However, the absence of ACKs leads to a unavoidable RTO expiration at 31 s. TCP Wave then behaves as well as during handover from L1 to L2: initial settings are restored and used to retransmit bursts, starting from the first unacknowledged packet, and the first two retransmitted bursts are lost again since they are sent during outage. Again, as in the previous case, the third retransmitted burst arrives to the destination triggering a cumulative ACK. Such a cumulative ACK completes the reception of an ACK train partially received before handover, covering also several subsequent ACK trains related to bursts successfully received. Therefore, it leads to a significant jump in the transmission sequence number, updating ACK train dispersion and RTT with values compliant with the characteristics of the old link. As a consequence, at 32.5 s, the burst retransmissions are performed at a rate similar to that experienced over the old link, which is too high for the new link bandwidth-delay product. Differently from TCP NewReno, TCP Wave is able to promptly adjust the rate monitoring RTT increases (details are provided in (Abdelsalam et al., 2015)). Therefore, after just 1.5 s from the activation of the new link, TCP Wave achieves the steady-state and in less than 6 s recovers from all the losses.

3.3 Analysis of the TCP rate

This sub-section complements the analysis provided in the previous one, showing trends of the rate as measured at the sender side transport layer in the same simulation setup. To support the result comparison, time scale has been normalized at the handover time

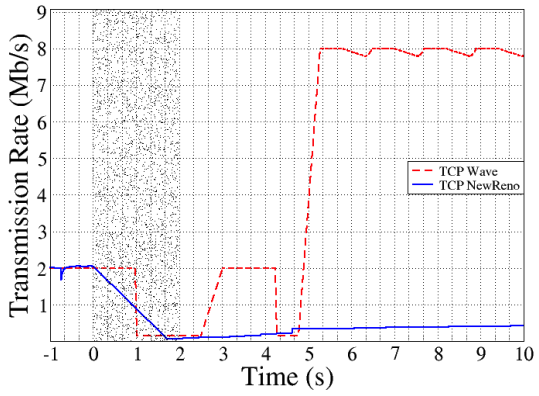

Figure 6: TCP transmission rate: handover from L1 to L2.

($t = 0$) in a significant time range. Fig. 6 compares TCP NewReno and TCP Wave during handover from L1 to L2.

After the handover and the triggering of the second RTO, TCP NewReno rate is reset to a minimum value corresponding to a $cwnd = 1$ and its growth is first exponential (Slow Start) and then linear once $cwnd > ssthresh$ (Congestion Avoidance). The effect on the rate is a very slow growth towards the maximum available bandwidth of 8 Mbit/s (more than 4 minutes from L2 activation are needed to achieve the new steady-state). To opposite, TCP Wave exhibits the capability to quickly recover from RTO thanks to the proactive generation of bursts also during the outage period. The first cumulative ACK received (see fig. 3) allows to restore the transmission over the new link allowing at glance 2 Mbit/s rate, while the second cumulative ACK allows to properly estimate the new available bandwidth with a rapid ramp-up to 8 Mbit/s. Definitively, the maximum rate over the new link is achieved within 3.2 s from its activation, which is a great improvement if compared with the 4 minutes needed by NewReno.

Fig. 7 shows the rate comparison in case of handover from L2 to L1, when a shrinkage of the available bandwidth is experienced by the transport protocol after the outage period. After L1 activation, TCP NewReno restarts running Slow Start algorithm with a *ssthresh* set to half of the amount of bytes in flight before handover, which is too big if compared to L1 bandwidth-delay product. The consequence is a quick growth of the rate much above the available bandwidth with inevitable multiple losses due to buffer overflow. TCP NewReno starts recovery of such losses one by one through the Fast Retransmit Fast Recovery (FR-FR) algorithm. This implies a halving of the *cwnd* that limits the actual rate. Simultaneously, the reception of *DupACKs* within the recovery phase allows a temporary inflation of the *cwnd* allowing the transmission of new packets accordingly at a very low rate. Therefore, At 45 s the recovery phase ends with the successful retransmission of the lost packets and TCP NewReno performs a shift of *cwnd* together with its resetting at the values set at the beginning of FR-FR (still oversized for L1). This causes further losses and a new FR-FR phase. As a result, the final steady-state is achieved around 48 s (16 s after the L1 activation).

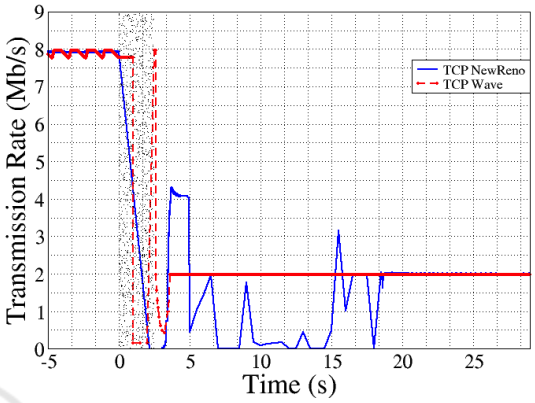

Figure 7: TCP transmission rate: handover from L2 to L1.

During the outage period, TCP Wave instead continues its proactive burst transmission although the lack of ACKs, until the RTO expires and then burst transmission is resumed using initial setting. Initially, TCP Wave restore transmission at the rate computed for the old link (8 Mbit/s) on the reception of the first cumulative ACK (at about 2.5 s). Then, the next ACK train report information on the new link characteristics and allows to properly set transmission parameters to achieve the new optimal rate (2 Mbit/s). Eventually, the new steady-state is achieved within only 1.5 s since the L1 activation.

3.4 Performance Summary

To complete performance analysis carried out in the previous sub-sections, test configurations have been extended with more TCP variants and different outage values for handover. In particular, the tests have been repeated with TCP Reno, TCP Tahoe and TCP Westwood+ (all implemented in the Ns-3 core) and configuring *HT* with two additional values: 0.5 s and 10 s. The latter values have been selected with the aim to assess the efficiency of the transport protocol recovery procedures in case of short network disconnections with short-term loss events (*HT* = 0.5 s), and with severe channel outages (*HT* = 10 s). As a synthetic performance indicator, the recovery time is considered, intended as the time between the reactivation of the new link after handover and the achievement of the steady-state. Result summary is presented in the

Table 1: Recovery time of different TCP Versions vs. disconnection period.

	HT=0.5 s		HT=2 s		HT=10 s	
TCP variants	*L1→L2*	*L2→L1*	*L1→L2*	*L2→L1*	*L1→L2*	*L2→L1*
Wave	81*	327**	3.2	1.6	5.6	1.6
Reno	271	6.7	270	7.6	270	10.7
NewReno	272	16	267	17	275	21
Tahoe	274	5.5	291	3.6	295	7.3
Westwood+	282	5	282	5.6	287	32

Table 1.

In case of medium and long *HT*, TCP Wave significantly outperforms all the other TCP variants. To opposite, in case of short *HT*, TCP Wave recovery time drastically increases, although in case of L1→L2 handover (*) it still outperforms all the other TCP versions. In the L2→L1 handover case, TCP Wave performance (**) is affected by a relevant impairment, showing a recovery time much higher than in the other cases, including also configurations with longer outages. The rationale relies on the fact that *HT*=0.5 s still leads to multiple losses (all the burst in flight), while TCP Wave does not fall into timeout since its proactive burst transmission, irrespective of the outage duration, allows the generation of *DupACKs* over the new link before the RTO expiration. In other words, differently from the ACK-clocked protocols, TCP Wave triggers a timeout only when *HT* is higher than RTO. Therefore, in this condition, TCP Wave triggers Fast Retransmit algorithm to recover one by one all the packets lost at once during *HT*. This implies the recovery of one lost packet per RTT and explains why the worst performance is experienced with L2→L1 handover: the number of burst transmitted during outage and then lost is higher over L2. As a conclusion, we can state that in this particular case, an RTO expiration brings performance advantages. Furthermore, the sooner RTO expires the higher is performance benefit, as demonstrated by the fact that TCP Tahoe or TCP Reno, which have limited (or no) capability to recover from multiple losses without triggering RTO, outperforms TCP NewReno in these conditions.

4 CONCLUSION

The work presented in this paper provides a detailed analysis of TCP performance during a vertical handover where bottleneck physical characteristics can suddenly change. As a general conclusion, traditional window-based TCP is not suited to efficiently face such communication scenarios showing impaired performance overall the simulated configurations. The new TCP Wave, exploiting a novel communication paradigm based on a proactive burst transmission, demonstrated its ability to quickly adapt its rate to network changes, greatly outperforming the other TCP implementations in most of simulated configurations. The only issue for TCP Wave is in the case of outage intervals shorter than RTO value. The proactive TCP Wave transmission also during outage avoids timeout expiration and leads to recover from a high number of losses exploiting Fast Retransmission algorithm, which has been tailored for efficiently managing network congestion events, and then results inefficient to manage the considered loss scenario.

REFERENCES

Abdelsalam, A., Luglio, M., Roseti, C., and Zampognaro, F. (2015). A burst-approach for transmission of TCP traffic over dvb-rcs2 links. In *2015 IEEE 20th International Workshop on Computer Aided Modelling and Design of Communication Links and Networks*.

Allman, M., Paxson, V., and Blanton, E. (2009). TCP congestion control. In *Network Working Group, RFC 5681*.

Bhat, P. and Talmale, G. (2014). MPTCP combining congestion window adaptation and packet scheduling for multi-homed device. In *2014 International Conference for Convergence of Technology (I2CT)*.

ETSI, ciao, c. (2015). *Satellite Earth Stations and Systems (SES); Hybrid FSS satellite/terrestrial network architecture for high speed broadband access*. TR 103 272.

Lee, K., Nam, S., and In-Mun, B. (2006). SCTP efficient flow control during handover. In *IEEE Wireless Communications and Networking Conference*.

Linux (2015). Programmer's manual, http://man7.org.

Ns-3 (2016). Ns-3 web page, https://www.nsnam.org/.

Patriciello, N., Casoni, M., Grazia, C. A., and Klapez, M. (2015). Implementation and validation of TCP options and congestion control algorithms for ns-3. In *Workshop on Ns-3 (WNS3) 2015*.

Paxons, V., Allman, M., Chu, J., and Sargent, M. (2011). Rfc 6298, computing tcp's retransmission timer.

Roseti, C., Luglio, M., and Zampognaro, F. (2010). Analysis and performance evaluation of a burst-based TCP for satellite DVB RCS links. In *IEEE/ACM Transactions on Networking, Vol. 18, Issue 3*.

Yuwono, A. (2013). Performance analysis of cubic and yeah TCP implementation using BATMAND over MANET. In *2013 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (ROBIONETICS)*.

Zhang, L., Okamawari, T., and Fujii, T. (2012). Experimental analysis of TCP and UDP during LTE handover. In *IEEE 75th Vehicular Technology Conference (VTC)*.