Wireless Networks (2021) 27:1413–1430
https://doi.org/10.1007/s11276-020-02526-6

An SDN-based true end-to-end TCP for wireless LAN

Krishna Vijay Kumar Singh[1] · Mayank Pandey[2]

Accepted: 14 December 2020 / Published online: 2 January 2021

Abstract
Segment losses due to intermittent connectivity and mobility lead to sub optimal performance of the Transmission Control Protocol (TCP). This is due to the fact that segment loss is considered as a binary signal for triggering congestion control and retransmission mechanisms at the TCP sender. In wired networks, segments are dropped due to congestion at the routers and the strategy of taking missed acknowledgment as an implicit signal for congestion control performs well. However, in wireless networks, segment losses are primarily due to mobility and transmission errors. Unlike many previous efforts, this paper proposes the design and implementation of Software Defined Network (SDN) assisted TCP which does not require the wireless Access Points (APs) to be TCP-aware and preserves end to end semantics. Further, no changes are required to be done in TCP protocol implementation at the end-hosts. The proposed approach utilizes the programmability provided by the SDN paradigm to intelligently trigger the spurious timeout detection and response algorithms, already implemented in standard TCP. The proposed approach is compared with the standard TCP and SDN assisted Zero Window based approach on Linux kernels using virtual data-plane switches and APs provided by the Mininet-WiFi platform. The implementation results establish the applicability of the proposed approach.

Keywords Mobility · Wireless TCP · SDN · Wi-Fi

1 Introduction

The continuous increase in the usage of mobile devices resulted in many research efforts in the area of wireless and mobile communication. The major chunk of current Internet traffic is originated from these mobile devices. However, these mobile devices often change their point of attachment in the network due to mobility. In view of the above, the main challenge is to provide seamless connectivity during mobility. Wireless connectivity to these mobile devices is either provided by cellular network or Wi-Fi. Cellular network is not included in the scope of this paper, On the other hand in Wi-Fi network end-hosts gets the network connectivity through Wireless Access Points (WAPs or APs). The APs and end-hosts both uses IEEE 802.11 Media Access Control (MAC) protocol [1]. Further, to increase the coverage of Wi-Fi networks the APs are deployed at different locations with the support of wired backbone. Thus the Wi-Fi setup is divided in wireless part (using IEEE 802.11) and wired part (mostly using IEEE 802.3). Moreover, based on the size or the region to be made Wi-Fi enabled, these APs may be part of single sub network or multiple IP sub networks.

If the all APs are in single IP sub network then the mobility of a node is termed as Layer2 mobility. Here, the network layer is not involved during change of attachment as the mobile hosts can keep their existing IP address. On the other hand, if the AP(s) are installed in multiple IP sub networks and then a mobile node may cross the boundary of a IP sub network. This is termed as Layer3 mobility. These different types of mobility situations pose different challenges in providing seamless connectivity. In layer2 mobility, only the MAC layer of the protocol stack is involved. Layer2 mobility is easier to handle than layer-3 mobility where both MAC and network layers are

✉ Krishna Vijay Kumar Singh
krishna.si@smit.smu.edu.in

Mayank Pandey
mayankpandey@mnnit.ac.in

[1] Department of Computer Applications, Sikkim Manipal Institute of Technology, Majitar, Rangpo, East Sikkim 737136, India

[2] CSED, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, UP, India

involved. The difficulty of providing a seamless mobility solution also depends upon the QoS requirements of applications running on mobile devices. Applications that cannot tolerate data loss require connection-oriented (TCP-based) service whereas the applications that can handle some data loss need connection-less (UDP-based) services from the underlying transport layer of the protocol stack. Providing TCP-based services is more difficult than providing UDP-based services during mobility. Further, the mobile host is in the role of a receiver or sender for an ongoing session also impacts the complexity of the solution.

The traditional IP-based Wi-Fi network architectures have very limited capability to support seamless mobility which adversely affects the overall performance. The mobility of nodes brings changes in the topology and the legacy wireless networks are not flexible enough to incorporate these changes in a transparent manner. In traditional networking devices, the rules and policies are configured once, and frequent changes such as dynamic inclusion of routing paths (due to change in topology) are not easy. Software Defined Networking (SDN) has been emerged a new paradigm in last few years. It offers network programmability and adaptability by decoupling the data plane (responsible for forwarding the traffic) and the control plane (responsible for making routing decisions) [2]. In this paradigm, all decision making functionalities are shifted to a logically centralized remote system known as SDN controller. The SDN controller consists of several modules which can be programmed as per the need. Consequently, it provides dynamic inclusion of network policies and rules based on the context.

We have utilized the SDN paradigm to address Layer2, and Layer3 mobility issues in IP-based Wi-Fi networks presented in our previous article [3]. The significant improvement is achieved in terms of handover delay in comparison with Mobile IP [3]. In addition, significant reduction in data loss for UDP-based traffic is also observed. However, the TCP-based applications suffer from long periods of interruptions during handover which resulted in sub-optimal bandwidth utilization. This is because TCP considers every packet loss as an indicator for congestion in the network and reduces the TCP sending rate drastically. In addition to that, the data transmission resumes only after Retransmission Timeout (RTO) period. The RTO gets doubled on each unsuccessful retransmission attempt which results in long period of inactivity.

TCP is originally designed for wired networks where loss of packet/acknowledgment can be considered as a true indicator for congestion. However, this assumption is typically incorrect in the wireless scenario where majority of packet losses happen due to transmission errors and mobility of the end hosts. Triggering congestion control mechanisms in such scenarios leads to unacceptable performance. Several approaches, such as I-TCP [4, 5], Snoop-TCP [5, 6], M-TCP [7], Freeze-TCP [8], and WTCP [9], have been proposed to handle this issue. However, these approaches have certain limitations which resulted in lack of deployable implementations. Some of them violate the end-to-end semantics of TCP whereas some require the APs to be TCP-aware. Further, some approaches require changes in the implementation of TCP protocol which is not a viable option.

In this paper, we present the design and implementation of SDN assisted TCP for enterprise wireless networks. Our approach utilizes the programmability provided by SDN to intelligently avoid the TCP sender from the effect of entering in congestion control phase due to mobility. In our approach, the impact of mobility-induced packet losses (considered as implicit congestion signal) is nullified which avoids the reduction in sending rate. More importantly, this is achieved without doing any changes in the standard TCP/IP protocol implementations at the end hosts. Further, our approach ensures that the end-to-end semantics of TCP is preserved and intermediate nodes (APs) are not required to be TCP-aware. The major contributions of our approach can be listed as below:

1. Design of an SDN assisted, efficient TCP for IEEE 802.11 enabled enterprise scale Wi-Fi networks using intelligent triggering of spurious timeout detection (RFC 3522 [10] and RFC 5682 [11]) and response (RFC 4015 [12]) algorithms.
2. Implementation of the proposed approach on Linux kernels using virtual data plane switches and APs provided by Mininet [13] and Mininet-WiFi [14]platforms.
3. Implementation of modules on Floodlight [15] controller to manage the mobility events, forwarding tables of network devices and subsequent abrupt TCP behaviour during handover.

The remainder of this paper is structured as follows:- In Sect. 2, relevant related works and their limitations are described. Section 3 presents the proposed approach and associated algorithms. This section also provides the implementation details. Section 4 presents the results and associated discussions. Finally, Sect. 5 concludes the paper with its limitations.

2 Related work

The related work section is described with the help of two separate sub-sections. The first sub-section presents the important traditional approaches which have brought significant changes in performance of TCP in a wireless and

mobile environment. The limitations of these efforts are also described and a summary of all these efforts are presented with the help of a Table. These approaches are mainly seen in mid-1990 to mid-2000, that is the reason to recognise all these efforts as traditional efforts. The second sub-section presents the SDN-based efforts to improve the performance of TCP. These approaches demonstrate the leverages of SDN to control and improve the performance of TCP traffic. However, none of these efforts are focused on wireless and mobile environment.

2.1 Traditional Efforts with their Limitations

In this sub-section some of the significant efforts recognized as traditional approach are presented. These approaches are I-TCP [16], Snoop-TCP [6], MTCP [17] and M-TCP [7] etc. The I-TCP [16], MTCP [17] and M-TCP [7] are based on splitting the TCP connection into two. One for wired and other for wireless portion of network. The functionality for wired portion remains same as standard TCP. On the other hand, for wireless portion of network (connections from AP to MN) follows a modified TCP. This modified TCP is designed to handle the issues of wireless network and mobility of clients. The aim of these approaches is to solve the issues such as high bit-error, mobility etc. locally in wireless portion, and not to disseminate information (about packet losses) in wired portion of network.

In I-TCP [16] and MTCP [17], TCP sender receives the acknowledgement from the AP instead of MN. In this manner, sender gets an impression that the packet is transmitted to the intended receiver. However, it may be possible that receiver is disassociated from the network. This breaks the end-to-end semantics of TCP where AP takes the full responsibility of reliable data delivery instead of TCP protocol running at end-hosts.

Unlike I-TCP [16] and MTCP [17], Snoop-TCP [6] does not use split connection approach. Instead, a watchdog agent is executed at the AP which keeps track of sequence number of data packets and acknowledgement numbers of every TCP connection. Also, the AP buffers data packets and maintains soft state of every connection. On detecting packet loss (when duplicate acknowledgements are received), the AP uses link layer retransmissions. It drops the duplicate acknowledgement and retransmits the data packet (presumed lost) from its buffer. In this way, Snoop-TCP is able to maintain end-to-end semantics of TCP. However, in this approach the link layer at the AP has to be TCP aware.

Both M-TCP [7] and Freeze-TCP [8] utilize the concept of Zero Window Advertisements (ZWA) to stall the sender when handover takes place and disconnection is detected. In M-TCP [7], the AP monitors all the packets and acknowledgements similar to Snoop-TCP. Further, the AP does not send an acknowledgement to sender unless an acknowledgement is received from MN. This helps in maintaining the end-to-end semantics. However, AP withholds the acknowledgement for the last byte acknowledged by MN. This withheld acknowledgement is used to send ZWA to stall the sender. However, this ZWA issued for last acknowledgement may also lead to negative usable window at TCP sender.

Unlike M-TCP [7], in Freeze-TCP [8] the ZWA is sent by the MN itself. MN in the role of receiver sends ZWA upon impending disconnection and sends full window advertisement (FWA) upon reconnection. In this approach, TCP running at MN has to be dependent on link layer to get knowledge about impending disconnection. During mobility, link layer is able to determine that the disconnection is about to happen based on the signal strength. After getting information from link layer, TCP at MN triggers ZWA to stall the sender. It may be noted here that further research for improving the TCP in wireless and mobile environment is restricted after mid 2000. This is due to limited support from the traditional networking devices to support the changes in network dynamics due to mobility. The summary of all the above mentioned efforts are presented in Table 1.

Moreover, the inception of SDN technology has given a new hope to handle mobility induced challenges of TCP in a quick and effective manner. The SDN-based efforts presented in next subsection tune the TCP behaviour according to the sudden change in traffic pattern. This has motivated us to apply the leverages of SDN technology to explore the possibility of further improvement of TCP behaviour in wireless and mobile environment. The SDN based efforts for improving TCP protocols are presented in next subsection.

2.1.1 SDN-based efforts with their limitations

After rigorous literature review we find vey few efforts [18–23] which have been introduced for tuning the behaviour of TCP using SDN technology. However, all these approaches are proposed for wired infrastructure-based networks. The approach presented in [19] is targeted for Content Delivery Networks [24] whereas other efforts have been made for data center networks.

In [20], an architecture of Open-TCP is suggested which uses the global network information along with the data transfer statistics monitoring capability of the SDN controller to dynamically adjust the TCP congestion control policies. The congestion control policies are periodically communicated to the end-host to adapt a specific variant of TCP variant through kernel-level adjustment.

Table 1 Summary of traditional efforts for solving issues of TCP in wireless and mobile environment

Name of the approach	Provision to restore traffic after handover	Provision to avoid slow start	Maintaining end-to-end semantics	Maintaining end-user transparency	Limitations
I-TCP [16]	Nil	Yes	No	Yes	Split the TCP connection.
Snoop-TCP [6]	Nil	Yes	Yes	No	AP need to be TCP aware.
MTCP [17]	Nil	Yes	No	Yes	Split the TCP connection.
Freeze-TCP [8]	Yes	Yes	Yes	No	End-User Centred Solution.
M-TCP [7]	Yes	Yes	Yes	Yes	AP need to be TCP aware. ACK of Last segment is withheld by AP. ZWA issued for last ACK may lead to negative usable window at TCP sender

In [22, 23], and [18], approaches for SDN-based incast congestion control policies for the data centers are proposed. In [22], the controller selects a flow based on its age to minimize the data transfer rate. Once, specific data-flow is identified to be elephant the alteration is achieved by reducing the advertising window field of the TCP acknowledgment segment. This adjustment is triggered when the OpenFlow switches sends a explicit notification of congestion which may require special support from switching fabrics. In [23], the queue-length of the congested switch port is sent as a signal for congestion which triggers the controller to invoke an appropriate TCP congestion control policy. Similarly, [18] is one more approach for TCP incast problem known as SDN-based Incast Congestion Control (SICC). This approach uses the arrival rate of SYN/FIN along with the queue length of data plane switches to predict the possible congestion. If a congestion is being monitored from a specific switch then subsequent traffic of the switch is intercepted and their receiver window is reduced to 1 MSS. This helps to deal with the TCP incast problem.

In [21], an architecture of Omniscient-TCP is presented which calculates Bandwidth Delay Product of a route between end-hosts. The TCP retransmission timers and the initial and range of congestion window size for a connection is adjusted as per the Bandwidth Delay Product. In [19], SDN mechanisms are used for efficiently managing the role of TCP Sender from one endpoint to another endpoint. The second endpoint selected can be introduced at any time during the communication based on the traffic pattern. This scheme is proposed keeping in mind the Content Delivery Networks (CDN) where connection endpoints may change for selecting a suitable server to deliver specific content.

Besides, [25] is a recent work focused on the split and join of TCP connection by utilizing SDN. The SDN capability of global view and control over the network is intelligently used for TCP flow-based splitting/aggregation. The TCP flows are split into non-overlapping independent flows based on the idea of MPTCP [26], and the improvement in performance is claimed with the help of SDN. It is known that MPTCP allows multiple sub-flows for simultaneous data transfer using multiple paths in a TCP single connection. The effort [27], also claim to improve the multipath TCP. An SDN-based new architecture namely S-MPTCP is presented which eliminates the several defects of sub-flow selection and deciding routes for each sub-flow of data. Here, the SDN controller manages each sub-flow more effectively using global network information and significant improvement in the TCP throughput has been claimed.

All these efforts have made significant contributions towards improving TCP through SDN. These efforts have motivated us to utilize SDN for tuning TCP in wireless mobile environments too. However, none of these efforts are made for wireless networks. Further, most of these efforts require changes in TCP running at endpoints whereas, in our approach, no such change is needed. To the best of our knowledge, ours earlier work SDN assisted ZWA-based approach presented in [28] is the first attempt to use SDN for tuning the behavior of TCP while maintaining its true end-to-end semantics. The work presented in this paper is improving our earlier work approach presented in [28]. The aim of the paper is to provide a user transparent solution which supports the following feature (1) Standard TCP implementation remain intact (2) The network must be adaptive to reconfigure itself according to the change in association point in due course of mobility

(3) Data Transmission can be re-initiated immediately after the re-association with best permissible data rate. (4) Switches and APs need not be TCP aware.

3 Proposed approach

As discussed before, our approach utilizes the programmability provided by SDN paradigm to tune the behaviour of TCP in wireless and mobile environment. In SDN architecture, the control plane of network devices is managed by a remote controller. In our approach we have utilized Java based Floodlight Controller [15] to manage the network policies. The network devices such as infrastructure switches and APs run the data plane only. These devices run as separate namespaces provided by Mininet Wi-Fi [14] platform. Mininet Wi-Fi is an extension of Mininet [13] platform for facilitating IEEE 802.11 capabilities in emulated network devices.

Figure 1, depicts a demonstrative SDN based topology for explanation of our approach. Here, there are four OpenFlow [29] enabled switches (S1, S2, S3 and S4) where S1, S2 and S3 are connected with OpenFlow enabled APs (AP1, AP2 and AP3). A Mobile Node (MN) moves from the range of one AP to another while maintaining ongoing TCP sessions with a Correspondent Node (CN). The switches and APs are connected to controller via secure OpenFlow channel. Using this channel, controller can add, delete and modify the routes (flows) dynamically as per the context. Also, controller can be equipped with different modules which work in event based manner to handle different events occurring in the network.

TCP has some fundamental procedures such as flow control, congestion control and segment retransmission in case of lack of acknowledgement. Flow control allows the TCP receiver to regulate the sending rate of TCP sender. This is done by using *Advertized Window size* field in TCP header. Figure 2 depicts the TCP flow control window. Here, the segments having sequence numbers between left edge and right edge is the current window size. The shaded area represents the segments which are sent (but acknowledgement is not received) whereas unshaded area depicts the segments which can be sent next (without waiting for acknowledgement). When the acknowledgements are received the left edge slides ahead. TCP implementations maintain three variables namely SND.WND (window size), SND.NXT (sequence number of segment which can be sent next) and SND.UNA (first unacknowledged segment at the left edge). The usable window is calculated as $SND.UNA + SND.WND - SND.NXT$.

In congestion control, TCP sending rate (congestion window) increases almost exponentially in slow start phase and linearly in congestion avoidance phase. These phases are determined using a variable known as slow-start threshold. The event of not receiving acknowledgement within Retransmission Timeout (RTO) triggers the resending of the segment. Further, TCP takes this non receipt of acknowledgement as implicit signal of congestion and drastically reduces its congestion window before entering in slow start phase. The calculation of RTO is a dynamic and adaptive process to decide about the time for which the TCP sender has to wait before issuing retransmission.

Figure 3 depicts the effect of mobility over RTO values at the CN which is in the role of TCP sender. It may be noted that when MN (in the role of TCP receiver) gets disconnected due to mobility, the acknowledgements could not reach to CN. Consequently, the segments are considered to be lost and CN makes an attempt to retransmit the

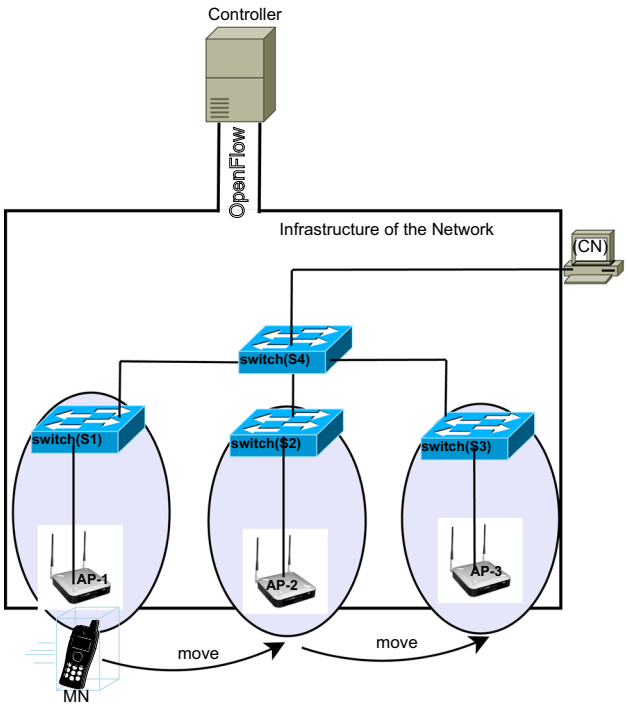


Fig. 1 Demonstrative topology

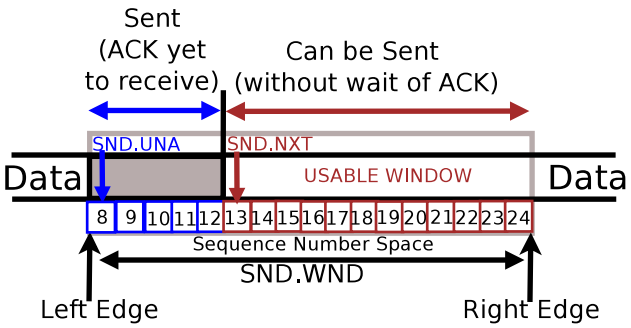


Fig. 2 TCP Flow control using sliding window

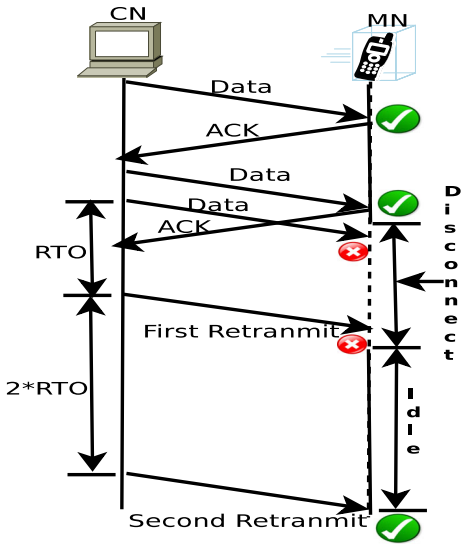


Fig. 3 Effect of mobility on data transmission

same segment after RTO. According to RFC 6298 [30], RTT samples should not be taken while retransmitting, and the RTO value should be set as the double of current RTO value at the time of retransmission. During handover, the disconnection of MN may lead to several successive timeouts. As a result, even if the MN is associated again, it is not possible for CN to resume the transmission. It can be observed from the figure that even after the re-association of MN, the sender and receiver both are idle for significant period (2*RTO).

Our earlier attempt described in the article [28] considers the above problem. The scheme is published through the article [28] which is an SDN-assisted approach that uses Zero Window Advertisement (ZWA) to freeze the TCP sender on the mobility event and termed as ZWA-based scheme. However, the ZWA-based scheme suffers from some serious issues. Therefore, this paper proposes a spurious-timeout-based scheme that improves the ZWA-based scheme. The brief description of our early effort (ZWA-based scheme) is presented through Sect. 3.1, the limitations of the ZWA-based scheme are presented in Sect. 3.2. The spurious-timeout-based scheme proposed through this manuscript is presented through Sect. 3.3. The further details of implementation for spurious-timeout-based scheme is presented in Sect. 3.4 and section 3.5.

3.1 ZWA-based TCP: an early effort

In our previous attempt [28] an approach using the Zero Window Advertisement (ZWA) concept to improve the performance of TCP in wireless networks is proposed. The basic premise of the approach is to assign the role of TCP receiver to SDN controller during handover when MN is disconnected. After reassociation, SDN controller relinquishes the role of TCP receiver and transfers the buffered segments to MN. Further, during handover, controller is programmed to issue ZWA for avoiding the event of TCP sender entering in to congestion control phase. As a result, the TCP sender freezes its window and its timers and enter in to the persist mode. When the buffered segments at controller are transferred to MN, traffic is restored immediately between MN and CN (TCP sender). It may be noted here that the SDN controller has given responsibility to freeze the sender when receiver MN gets disconnected.

The SDN controller behaves as an intermediary between TCP sender and receiver. Thus, proper synchronization between the sender and receiver lies with controller which is achieved by flow statistics module. The module acquires the exact number of segments transferred to the receiver related to a specific TCP session. This information is necessary track before issuing the ZWA, because acknowledgment is cumulative and once given to TCP sender, it advances the window as per sliding window protocol described in 2. Thus, a provision is made at the controller to figure out the possibility to issue ZWA or not. In any case, if the ZWA cannot be issued for a segment received at the controller then ZWA-scheme is disabled and standard implementation of TCP takes over. This happens because statistics collected from forwarding devices could not ensure about the transfer of after which acknowledgement can be issued. This situation may arrive due to congestion in network or traffic uses a stale route. During the experiments existence of possibility of stale route due to change in association point is found a main reason for above situation.

3.2 Issues and challenges with ZWA-based TCP

SDN assisted ZWA-based TCP faced serious challenges in the event of bit errors. The dominant reason behind this shortcoming is the break of synchrony between sender and receiver. The ZWA along with acknowledgments are issued by SDN controller before the data segment could actually reach to receiver. Further, to prevent negative usable window scenario at sender (RFC793 [31] and RFC1122 [32]), many ZWA segments (with acknowledgments) have to be issued. Specifically, before sender could actually commence the persistence phase (and freeze the timers etc.), these multiple ZWA segments are able to slide the sender window forward in sequence number space. The limitations of ZWA based approach are explained in following paragraph with the help of Fig. 2.

For example, the state of TCP sender just before the handover is depicted in the Fig. 2, where value of SND.UNA=8, SND.NXT=13, SND.WND=16. During the handover the sender sends rest of the permissible segments from usable window (number 13–24). It may be recalled

that issuing ZWA may also lead to negative usable window. Considering the ZWA of segment number 15 which lead to SND.UNA = 16, SND.WND=0 and SND.NEXT=25 (as sender sends all the segments from its permissible space). Now, the usable window becomes minus 9, using the formula $SND.UNA + SND.WND - SND.NXT$ stated earlier. Hence, scheme has been proposed to acknowledge all segments from 13 to 24 where the ZWA for segment 24 can only freeze the sender.

The controller successfully receives all these segment as MN is in disconnected state. As per the scheme proposed in [28] the controller issues an ZWA for all these segments. This ZWA informs the sender about the successful delivery of all segments up to sequence number 24. As a result, the left edge of the sender window is adjusted and segment number 24 is considered as sent and acknowledged. The receiver MN gets its connectivity after successful completion of handover from new association point. The SDN controller in turn is responsible to transfer the segments from its buffer associated with MN. Now suppose, while SDN controller is performing transmission of segment number 13–24 some bit errors are introduced. As we are aware that wireless medium is more prone to bit error.

As explained before, sender slides its window as it gets acknowledgements (ZWA) from SDN controller (which is not performing checksum procedure). But, the receiver is not able send cumulative acknowledgement for segments having errors which can be detected by receiver during checksum procedure. Receiver keeps on sending duplicate acknowledgements for the last correctly received packet which sender has to ignore as its window has already moved ahead. This results in a fatal flaw which leads to connection reset after some time. In some sense, the acknowledgments (ZWA) issued by the SDN controller violates the end-to-end semantics of TCP in such cases. As the damaged segments can only be identified at the end host (receiver). Also, this failure is entirely irrecoverable because the sender clears its buffers for the acknowledged segments. Hence, these segments are not available to the sender, receiver and SDN controller.

Overall, the scheme suffers from two major issues the first issue is with the controller to ensure transfer all previous segment before issuing an acknowledgment through ZWA. If, the controller is not sure then scheme fails and it is reverted. The second is bit-error which is considered to be very serious as it leads to connection reset.

3.3 Spurious-timeout-scheme

Looking from broader prospective, the timeouts which happen at TCP sender because of link-layer retransmission attempts and mobility do not reflect the correct underlying situation. Conventional TCP sender assumes that packet has been lost when the RTO timer expires and enters into retransmission and congestion control. However, the fact is that packet has not been lost, it is just delayed due to dynamically varying conditions of wireless and mobile environments. These timeouts are called *spurious timeouts*. These timeouts would not happen if the sender can wait a little longer for the acknowledgment.

Many efforts have been made to handle the problem of spurious timeouts. These efforts are generally focused on two aspects, the detection mechanisms and procedures for the response once the spurious timeout is detected. The detection mechanisms are utilized for determining whether timeout is spurious or actual, whereas response mechanisms attempt to undo the effect of actions (reduction in congestion window etc.) TCP sender normally performs after timeout.

In the year 2000, the Eifel algorithm [33] was proposed, which has defined the problem of spurious timeouts in TCP and suggested a solution. Further, it is incorporated in an experimental RFC 3522 [10]. This algorithm uses timestamp option (RFC 1323 [34]) to detect the spurious timeouts. The procedure is quite simple. Whenever a segment is retransmitted, the TSV (Time Stamp Value) is stored. When an acknowledgement is received which can ascertain the delivery of the retransmitted segment, its TSER (Time Stamp Echo Reply) value is inspected. If it happens to be smaller than the stored TSV value, this indicates that the received acknowledgement corresponds to the original transmission rather than the retransmission. This provides a hint that the retransmission must have been unnecessary and spurious.

In RFC 5682 [11], another method for detecting spurious timeouts is described. This method is known as Forward RTO-Recovery (F-RTO) algorithm. This method does not require any TCP options, making it suitable even for the old TCP implementations which do not support timestamp options. In this algorithm, when the retransmission timer expires, the sender retransmits the first unacknowledged segment which is a usual process followed in slow start. However, different from the normal operation, TCP sender tries to send new (previously unsent) data in response to the first acknowledgement that arrives after the timeout (given that it advances the window too). Subsequently, if the second acknowledgement (which arrives after the timeout) also advances the window resulting in *acknowledging the data that was not retransmitted*, then the timeout can be declared spurious. In this case, TCP sender comes out of RTO recovery process. On the other hand, if either of the first or second acknowledgements is a duplicate acknowledgement, then it is not certain that the timeout is spurious or not. In this case, TCP sender resumes retransmitting the unacknowledged segments as per the traditional slow start algorithm.

All the recent Linux Kernels (4.4.70, 4.9.30, 4.10.17 and 4.11.3) which we have checked to implement the concept described in RFC 3522 and RFC 5682 based on the availability or non-availability of TCP options at sender. After the detection of spurious timeout, the Eiffel response algorithm (RFC 4015) [12] is utilized as a response procedure. In this procedure, when retransmission timers expires, the current values of SRTT (smoothed round-trip time) [30] and RTTVAR (round-trip time variation) [30] are stored in variable SRTT_prev and RTTVAR_prev respectively. It may be recalled that SRTT and RTTVAR are used for retransmission timeout value calculation as mentioned in RFC 6298 [30].

If the timeout is detected to be spurious, TCP sender takes the help of a flag variable *SpuriousRecovery*. If the value *SpuriousRecovery* is set as SPUR_TO then the sequence number of next segment (SND.NXT) is adjusted to first new unsent segment. This is done to avoid unnecessary GO_BACK_N phase followed in slow start after the retransmit timeout. In addition to this, other different TCP variables such as slow start threshold (ssthresh) and congestion window (cwnd) are restored to the values before timeout. Further the values stored in SRTT_prev and RTTVAR_prev are utilized to estimate the next SRTT values. This is required to cancel out the effect of delay spikes introduced due to handovers.

We have utilized the above mentioned mechanism of detection and response related to spurious timeout in a intelligent manner for our SDN-assisted-spurious TCP. It can be observed that if somehow the TCP sender can be forced to initiate spurious timeout response algorithm after the handover then the data transfer can resume again (with window sizes applicable before the handover). Further, in this situation the sender idle time can also be reduced substantially.

It must be noted here that the TCP sender initiate such measure only if *SpuriousRecovery* is set to SPUR_TO. The default value of *SpuriousRecovery* is set as FALSE on each retransmit timeout event. The value *SpuriousRecovery* changes only if the acknowledgement is received which can advance the window. It means that the sender gets an acknowledgment for the segment or bytes which were not retransmitted after the timeout. This situation is very rare and likelihood of such situation is negligible during mobility.

It is known from results of paper [3] that MN remains disconnected for significant time during the handover. Hence, the segments approaching towards the MN is dropped in absence of receiver. The sender expectation to get the acknowledgement for these dropped packet will never be fulfilled. In the given situation sender invokes retransmit timeout and value of *SpuriousRecovery* remain to be FALSE forever.

Looking at this situation we have intelligently programmed our controller to capture and store the segments approaching towards MN. It means similar to the ZWA approach SDN controller pretends as a receiver to capture the data traveling towards MN in its absence. But, it does not issue an acknowledgment for the buffered segments. In spurious-timeout-based scheme data synchronization completely lies with the sender and receiver. However, the SDN controller plays a very vital role to decide the point of time from where the session should be re-initiated and the beginning point (segment number) of established sessions by avoiding the effect of RTO. This is because once the receiver MN confirms its association after the handover the stored segments at the controller are utilized to set *SpuriousRecovery* flag with SPUR_TO and subsequent action in the desired order. The details of the implementation are presented in the next subsection.

3.4 Implementation details

The approach proposed in this paper protects the TCP sender for any such losses due to mobility or bit-error. The proposed protocol includes the following modules: *mobility detection*, *route establishment* and *handler for retransmission timeout*.

– *Mobility detection* module detects mobility (handover) events, such as association and disassociation events at the SDN controller. The controller takes appropriate action depending on the type of mobility event. The mobility event notification messages have to be generated from the OpenFlow-enabled access points. In a wired network, node connection and disconnection (port up/down) are automatically propagated from data-plane to control-plane (SDN controller) using OpenFlow messages. A port status message typically specifies the status of a port along with the reason for the change status (addition, deletion, or modification, in the topology). However, in wireless networks, one wireless link (such as an access point) may serve multiple users on the same channel. Hence, movement of a mobile node does not cause any changes (such as port status) to be communicated to the controller. We have used scapy tool [35] to capture link-layer IEEE 802.11 management frames (such as association and disassociation) received from the MN. It may be noted that the scapy is a packet manipulation tool which enables the users to dissect, modify and send packets in a network. This tool is developed using python and is capable of doing all these manoeuvres interactively. The interactive packet manoeuvring allows to dissect or scan packets from the network, interpret the context and then a new packet with desired parameters can be injected.

– *Route establishment* module establishes a new route according to the recent association point of the mobile node. This module is also used to delete all the flows that are related to MN from infrastructure switches upon disassociation [28]. To determine which of the infrastructure switches contain flows that are related to a disassociated MN, the *Routing* module of the original Floodlight [15] is used. The *routing* module provides list of unique datapath IDs (DPIDs) of the switches that are present on the shortest route between a specified pair of the source (CN) and destination (MN).
– *Timeout handler* module is developed and utilized to store the TCP traffic to protect from mobility-induced losses. A dynamic queue for each TCP connection at the SDN controller is maintained to store the segments on the behalf of disassociated node (MN). It may be noted that queue is instantiated at the time TCP initial connection setup procedure known as three way handshake. This queue starts storing the segments immediately after MN initiates the handover, and continues till the re-association request is received. The Algorithm 1 describes the process of storing the segments at the SDN controller.

Algorithm 1 describes the response of the centralized controller for a data segment related to the disassociated set of end-host reaches to it.

Algorithm 1 :Response of the centralized controller

h

```
1: procedure actionOnPacketInSPT(packet)
2:     for All received packets do
3:         if destination address of the packet ∈ disassociated set of packets then
4:             for packets ∉ Controller.Buffer do
5:                 keep the packets in the SDNController.Buffer
6:             end for
7:         end if
8:     end for
9: end procedure
```

In our approach, a set of dis-associated nodes along with their addresses (such as MAC-address and IP-address) are maintained at the SDN controller. Line-2 and Line-3 of Algorithm 1 shows that controller inspects all the incoming segments against the disassociated set of nodes. The controller also maintains a separate buffer for each disassociated node where the segments destined towards the disassociated node are stored. This buffer is temporary in nature and flushed through *packet_out*. The buffer flush is initiated, once the disassociated node rejoins the network. The Line-4 and Line-5 of the algorithm shows that the segments which are not stored in controller buffer are considered for storage. This is done to avoid the storage of duplicate segments, which may arrive due to retransmission, especially for a long disconnection period.

Later on, when MN is re-associated with another AP, it receives all the buffered segments from the controller. Subsequently, MN (which plays the role of the TCP receiver) issues cumulative acknowledgments that acknowledge all these segments. When these cumulative acknowledgments reach the TCP sender, the following events occur in sequence:

– The TCP sender comes out from the idle mode.
– The spurious timeout is detected, which leads to restoration of the congestion window and slow start threshold to the values before the timeout.
– The TCP sender resumes the segment transfer at full speed as if nothing happened.

3.5 Implementation setup

The network topology emulated and evaluated using Mininet-WiFi [14] and the controller modules are implemented using Floodlight [15]. The necessary support for implementation of Wi-Fi network using IEEE 802.11 media access control is available in Mininet-WiFi. To implement an access point which support OpenFlow interface and install controller driven flow in it. A wired interface of an OpenFlow-enabled switch is turned in to a wireless interfaces using *hostapd daemon*. This enable a switch to act as an OpenFlow-enabled access point. In addition, Mininet-WiFi also provides support for node mobility with the help of various mobility models (such as random walk, random way point etc.,). In our experiments the mobility models are utilized to mimic the required mobility patterns of mobile nodes. The mobility pattern of mobile node results in to the transition of access point from one to another. The network topology used in our experiments are depicted in Fig. 4.

The network topology includes 5 OpenFlow-enabled switches (S1 to S5) and three different OpenFlow-enabled wireless access points, namely AP-1, AP-2 and AP-3. All OpenFlow enabled devices such as switches and APs are directly with the SDN controller through OpenFlow channel using CAT-6 cable with a delay of less than 1 ms. The access points such as AP-1, AP-2 and AP-3 are also connected with three different switches S1, S2 and S3, respectively. In addition, the configuration of wireless channels of AP-1, AP-2 and AP-3 are 1, 6 and 11 respectively. This configuration is assign to avoid the co-channel interference among these APs. All wired links in the given topology have a uniform capacity of 100 Mbps. The delay of each link in the topology varies from 20 to 80 ms.

To measure the performance of the proposed scheme three different traffic conditions are emulated. In the very first traffic condition link delay is kept as 80ms along with the maximum permissible TCP window size 30,000 bytes. It may be noted that the default size for the maximum TCP permissible window is 65535 bytes. We

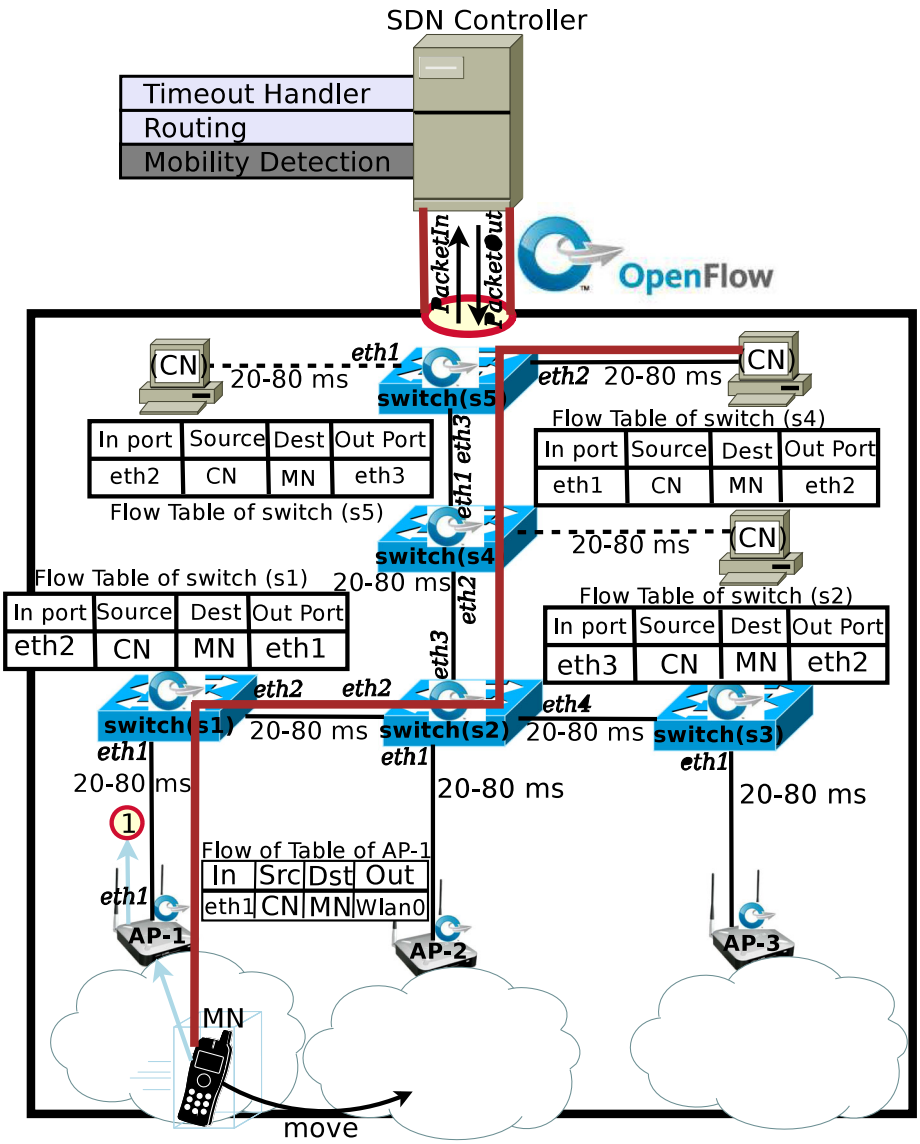


Fig. 4 Implemented network topology with initial route setup towards MN

have reduced it to 30,000 and the link delay is set to 80 ms to demonstrate the performance in very light traffic conditions. The 80 ms link delay on 100 Mbps link gives a space to keep 10Mb of data which is very large compare to the requirement a single TCP connection (30000 bytes). The second condition is considered as medium traffic where the window size is 60,000 equivalent to the default value of TCP window 65,000 bytes with the link delay of 50ms. The third traffic condition is considered as heavy traffic where the window size is increased to 90,000 bytes using the window scale option and link delay is kept 20 ms.

Through the our experiments, we have learned that high link delay is necessary to anticipate a good performance of the scheme (using ZWA). This is because transfer of disassociation and reassociation notification to the controller requires substantial amount of time (approximately 70 ms). After close inspection we could figure out that this is because in Mininet-WiFi environment where software APs requires sufficiently large amount of time (approximately 70 ms) due to python based module to capture and process IEEE 802.11 management association/disassociation frame. Therefore, due to higher notification delay link delay, controller takes time to manage the mobility event till that time old and stale routes/flows remain effective. Therefore the high link delay is necessary to observe the good performance.

By contrast, the scheme (creating spurious timeouts) that is presented in this paper outperforms over other approach under any delays, starting from 20 ms. Also, according to Fig. 4, the point of an attachment of fixed node(CN) can be changed to analyze the performance of the proposed approach under various hop count between the fixed node (CN) and the mobile node(MN). The total number of correspondent nodes (CN) is two, and the

number of mobile nodes (MN) is 20. All odd numbered MNs are communicated with the first CN, and all even-numbered nodes communicate with the second CN. Two machines are used in the experiment, one is used to implement data-plane (Mininet-WiFi), and the other one is used to implement control-plane(Floodlight). Other details of emulation setups are presented in Table 2 The kernel of the machine is configured to illustrated and analyze the impact of various parameters on proposed approach. The TCP Reno [36] implementation is set in Linux kernel with the SACK option [37] enabled.

To explain the changes in the route of given topology during the mobility of a node MN. The timeline of an important event at the infrastructure layer is presented using Fig. 5. In the given figure T0 represent a point time when MN declares its disassociation due to mobility, T1 represents a point of time where existing flows are deleted from the infrastructure switches with respect to the disassociated node. Similarly, T2 represents a time where MN re-associates it from the network after completion of handover and T3 represents a time when new flows are inserted to the switches corresponding to new attachment point of node MN. Fig. 4 represents the route/flow setup before the handover. The given figure also depicts that MN has declared its impending disassociation through AP-1. The direct colored line from CN to MN depicts the path through which data is moving from CN to MN. On the other hand, Fig. 6 depicts the new flow setup established after the handover (at the time T3) where the mobility of MN is accounted by the SDN controller. The figure also depicts different modules of the SDN controller that are made responsible for capture and process the mobility event to exhibit appropriate changes in the route.

Now, if we analyze the topology with the reference to time point T0, T1, T2 and T3 then the time point T1 is the time where mobility of MN is recorded at the controller and existing flows for carrying the data from CN to MN are deleted. It may be noted here that, from time point T0 to T1 old and stale flows are in effect to carry the data. Figure 6 depicts the stale route through a red crossed line which may take the data traffic to the old location. With the help of experiments, the time interval between T0 and T1 is recorded around 70 ms. Now, if the total link delay between the CN and MN remains less than 70 ms there is a possibility that the entire TCP window of CN gets transmitted through the old and stale route. Consequently, all transmitted segments between T0 and T1 are dropped before the SDN controller could initiate an action. Therefore, in our experimental topology total link delay is kept more than 70 ms. In a few experiments when set the link delay of less than 10 ms, we observed that the proposed scheme exhibit performance similar to the traditional TCP without any improvement. This is because SDN controller could not catch the traffic before it gets delivered to wrong destination.

Iperf tool [38] is utilized for performance evaluation and traffic generation from individual end-hosts. Various parameters, such as the throughput can be analysed directly using iperf. In addition, the changes in the size of the TCP sender's window are made with iperf which are recorded using a shell script running at end-hosts. The Wireshark [39] tool is also used to make deep inspection of traffic along with the behaviour of TCP sender, TCP receiver and SDN controller during handover. The all the segments associated to mobile node (MN) and the correspondent node(CN) were captured and analysed using Wireshark. To get the exact knowledge of TCP traffic, we have captured the traffic through Wireshark from the controller, sender and receiver (station). To capture the Wireshark trace from wireless station or MN, an existing wireless interface (sta1-wlan0) of station is captured in monitor mode. The monitor mode gives the privilege to the user to capture IEEE 802.11 standard frame format.

Table 2 Emulation configuration

Emulation time	120–300 s
Mobility model	Linear (to emulate different disconnection duration)
Speed	0–4 m/s
Emulation plateform	Mininet–WiFi
Type of traffic	TCP
Controller	Floodlight v1.2
Linux Kernel	4.4.70
Machine configuration	CPU Intel(24 Cores), Cache-W, DDR3 32GB RAM
Radio technology	802.11 b/g, Tx-Range: 33 meter, Tx-Power: 20 dBm
Propagation loss model	Two ray ground
Link delay	20–80ms
Number of mobile nodes	20
Congestion window size	30,000–90,000 Bytes

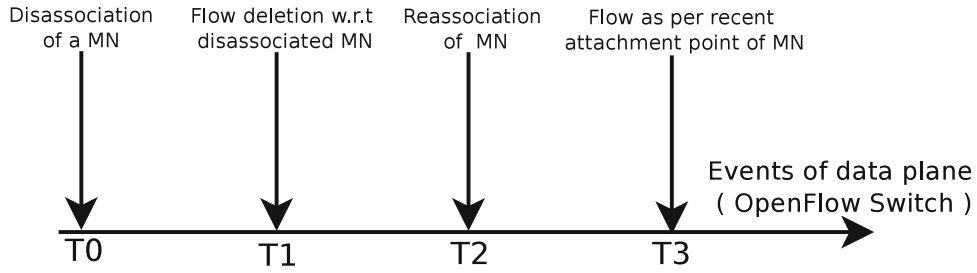


Fig. 5 Timeline of events at OpenFlow Switch due to handover

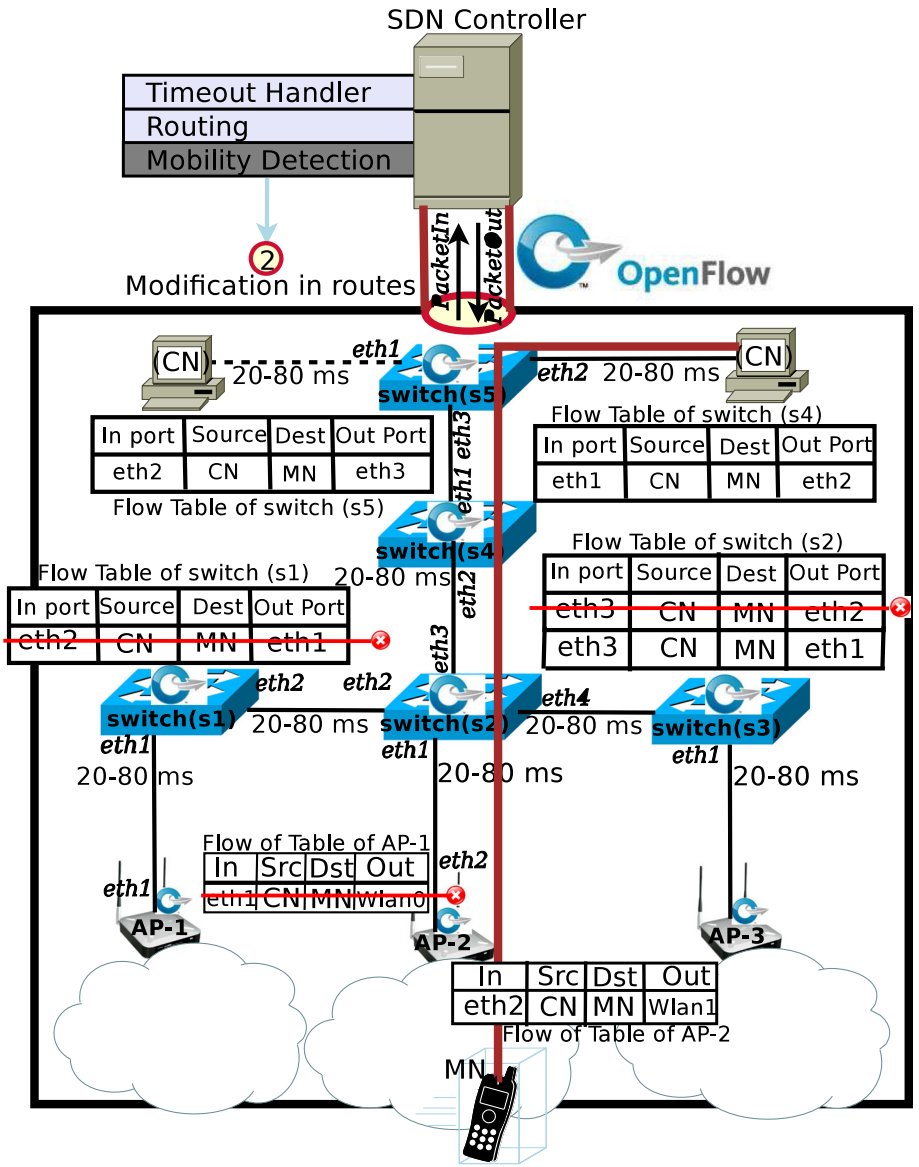


Fig. 6 Implemented network topology with route setup after the handover

4 Results and discussion

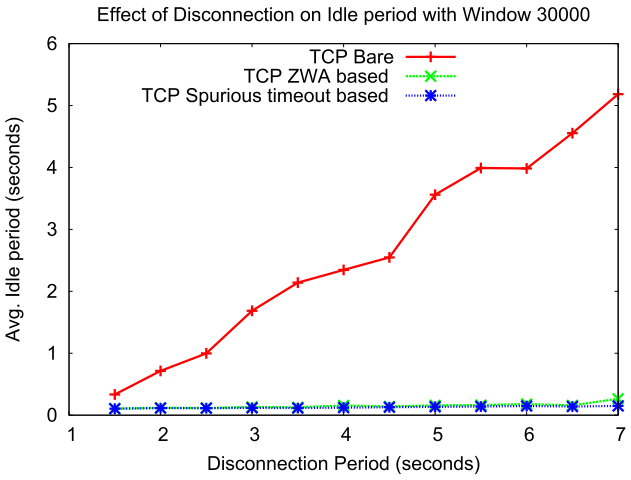
The performance of the proposed approach is measured in terms of TCP idle period, variation in congestion window and the number of retransmissions attempt during the handover. In addition, a throughput snapshot is also given to understand the change in data transfer rate due to handover, the performance of the controller is also measured in terms of the response time (the time that is taken by the controller to respond to a packet_in event). These values are recorded under various disconnection duration. Also, set of results are recorded under various window sizes and link delays between the fixed node CN (in the role of the TCP sender) and the mobile node MN (in the

role of the TCP receiver). Three schemes of TCP are compared, the first approach is referred as bare-TCP where SDN controller provide no support for TCP sessions (it only provides support for mobility as stated in article [3]). The second approach is referred as ZWA-based where controller acts as TCP receiver to send ZWA when MN is disconnected (as stated in article [28]) whereas the third scheme is referred as spurious-timeout-based proposed in this paper.

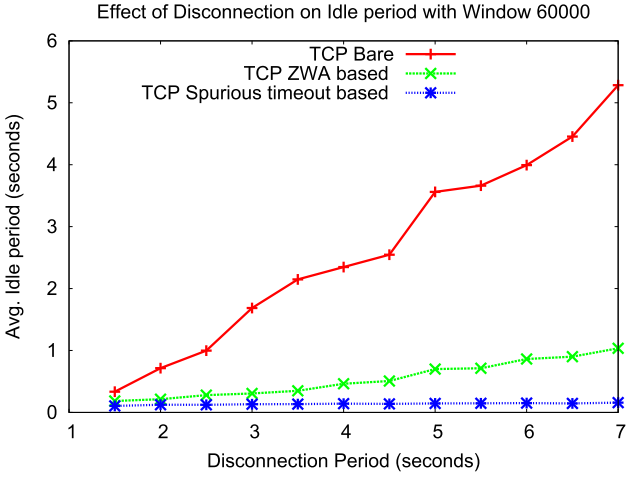
Figure 7 depicts the traffic that was captured using bare-TCP (TCP without getting transport layer support from SDN controller), TCP using ZWA and TCP with spurious-timeout-based approach. In these results, the TCP sender(CN) and receiver(MN) are separated by three hops, and the link delay of each link varies from 20ms to 80 ms. In addition, the disconnection period of the mobile node varies from 1.5 to 7 seconds. According to Fig. 7a, bare TCP leads to a long idle period in comparison to ZWA-based TCP and spurious-timeout-based TCP. The Idle connection period of ZWA-based TCP and spurious-timeout-based TCP is much shorter. The Fig. 7a also depicts that the idle connection periods are the same for the ZWA-based scheme and the spurious-timeout-based scheme. However, according to Fig. 7b and c where the traffic load on the network is increased, the performance of the ZWA-based scheme is degraded whereas the spurious-timeout-based scheme performs consistently well under all traffic load. Further investigations also reveal that successive retransmission attempts by the TCP sender during the disconnection period lead to long idle periods in bare-TCP. However, these successive timeouts does not lead to long idle periods for ZWA-based and spurious-timeout-based scheme. This is because the SDN controller has taken the appropriate measure to re-initiate the data traffic as soon as re-association of MN is reported.

The ZWA-based TCP data transfer performance is poor at times when any segment arrives between times T0 and T1 (when the mobile node is disconnected; refer Figure 5). In this situation, SDN controller relinquishes its control on TCP operations (controller intervention is disabled) and original TCP is activated to revive the TCP connection in a conventional manner. Hence, the SDN controller does not play any significant role in such corner cases and the performance remains the same as that of bare-TCP.

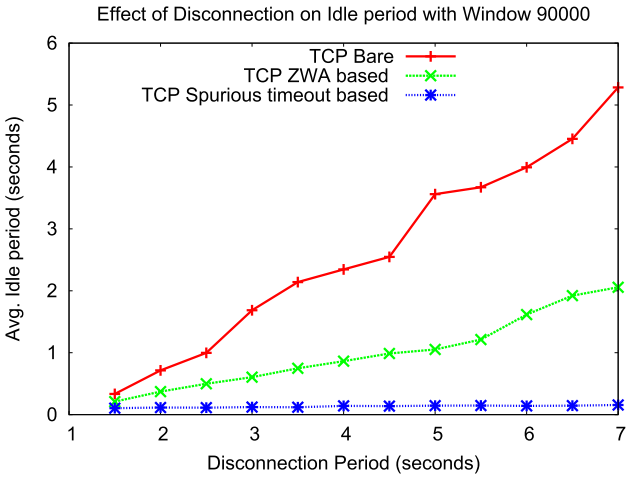
Figure 7 presents the comparative analysis of all the schemes proposed in this paper. The spurious-timeout-based scheme substantially outperforms bare-TCP as well as the ZWA-based scheme. Data transfer is interrupted only when the mobile node is disconnected (1-2 seconds). As soon as the mobile node re-associates itself, data transfer resumes immediately. Effectively, the interruption is only for the period when the mobile node is not connected. In case if a data segment arrives between time



(a) Window 30000, link delay 80ms



(b) Window 60000, link delay 50ms



(c) Window 90000, link delay 20ms

Fig. 7 Comparison in terms of Connection Idle Period

interval T0 to T1 (refer Fig. 5), it is considered to be a lost segment and is retransmitted from the source. In contrast with the ZWA-based scheme, the arrival of any segment to

the data plane between time interval T0 and T1 leads to activation of the conventional TCP with slow start procedure. In the case of the low link delay (such as 20 ms) along with large size of congestion window (greater than 40,000 bytes), the segment arrival rate at the old access point between time interval T0 and T1 becomes high. This is the reason behind the degraded performance of the scheme (ZWA-based) in Fig. 7b, which is further degraded in Fig. 7c.

Figure 8 compares the average size of congestion window to evaluate the performance of proposed approaches on different parameters under the scenario discussed above. Figure 8a shows that ZWA-based and spurious-timeout-based scheme have similar performance and they outperform bare-TCP. Figure 8b and c depicts that when traffic load increases then segment arrival rate between time interval T0 to T1 (refer Fig. 5) increases which leads to reduced cwnd for ZWA-based scheme. However, TCP spurious-timeout-based outperforms other schemes under varying traffic conditions.

Figure 9 compares the performances of the bare-TCP, ZWA-based scheme and spurious-timeout-based scheme with respect to the number of re-transmitted segments. These re-transmitted segments are captured during the handover for each TCP connection. Hence, the TCP receiver collects the segments to be retransmitted for each TCP connection until the sender and receiver resume their data transmission. According to Fig. 9, when we increase the traffic by increasing the window size and reducing the link delay, performance of the ZWA-based scheme degrades. The reason is the same as discussed above:- if a segment arrives between times T0 and T1 (refer Fig. 5), then the ZWA-based scheme adopts the standard mechanism similar to bare-TCP.

These results are expected because, in bare-TCP, retransmission timeouts reduce the size of the sender's congestion window which leads to non-optimal bandwidth utilization. The average size of congestion window (cwnd) is compared using Fig. 8. The figure reveals that average size of cwnd is less in bare-TCP. In contrast the average size of cwnd in ZWA-based and spurious-timeout-based scheme is more. This is because mobility-induced segment losses cannot trigger traditional congestion control mechanisms for ZWA-based or spurious-timeout-based scheme which may lead to the reduction of the sender's cwnd to 1. However, Fig. 8b reveals that in case of heavy traffic condition performance of ZWA-based scheme and TCP-bare is same.

According to Fig. 9, the average number of retransmissions is minimum in ZWA-based scheme because TCP sender is freezed due ZWA segments transferred by the controller. As expected, bare-TCP has the largest average number of retransmissions. For spurious timeout based

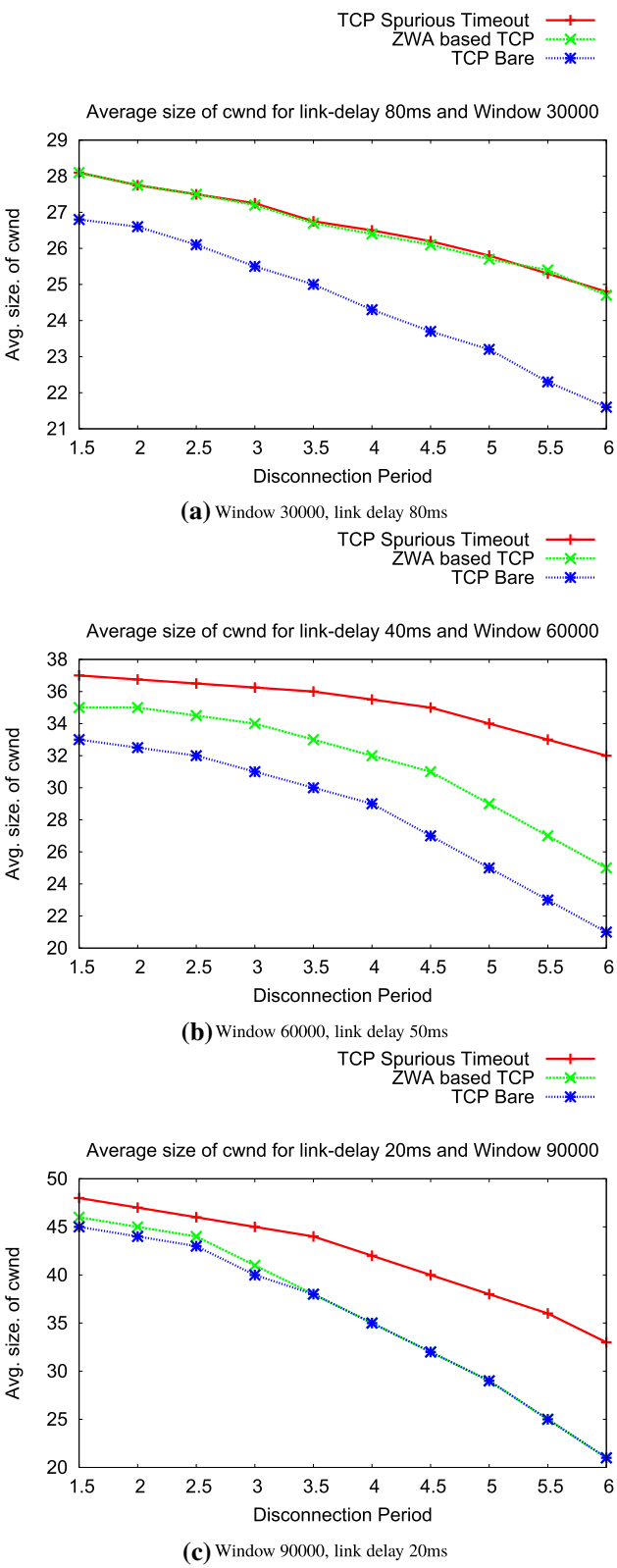


Fig. 8 Comparison in terms of congestion window

scheme, the average number of retransmissions is less than that of bare-TCP because redirection of buffered segments

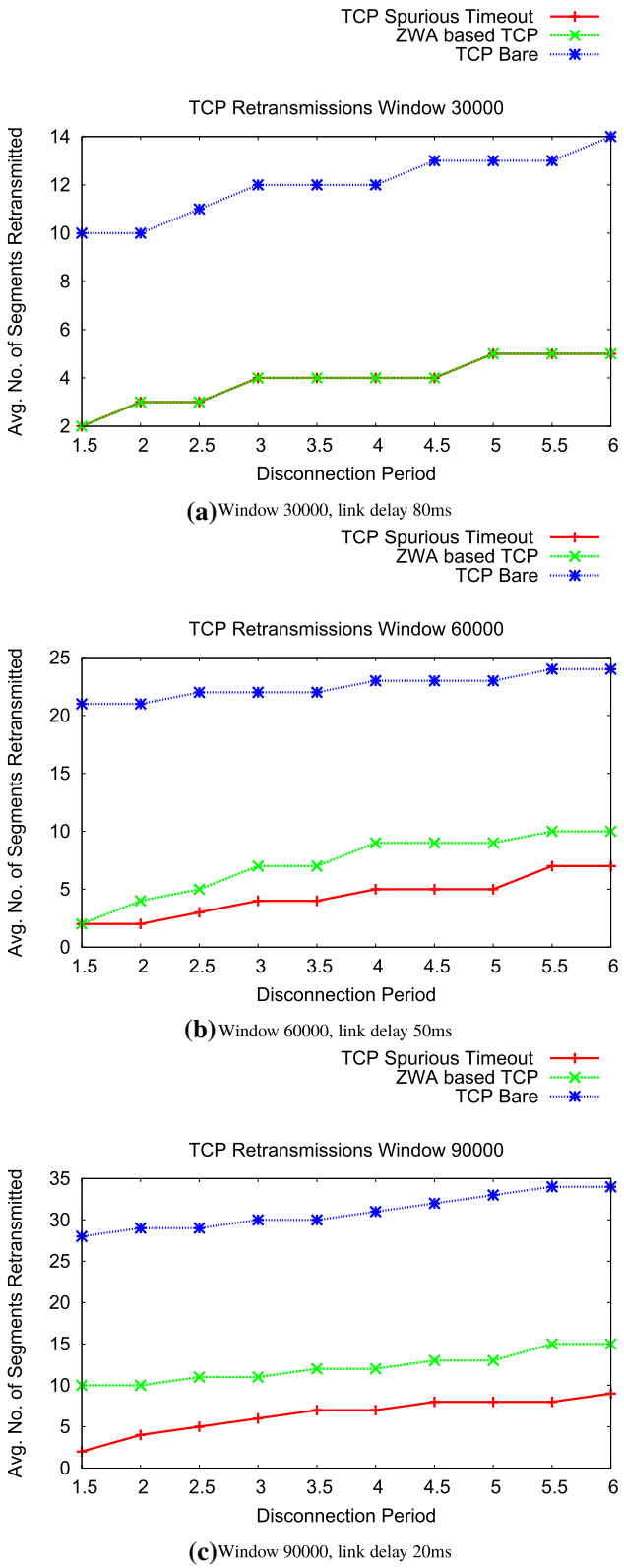


Fig. 9 Comparison in terms of retransmission

towards TCP receiver leads to issuing of cumulative acknowledgements. This in turn, enables the sender to restart the transmission ahead of the left edge of the sender's window. On the other hand, in bare-TCP the transmission begins from the left edge of the sender's window after the receiver has re-associated.

For understanding of the effect of handover on data transfer the Fig. 10 compares the rate of data transfer for TCP-bare and spurious-timeout-based schemes under heavy traffic conditions (with link delay 20 ms, window size 90000 bytes and disconnection period 3 sec). The transfer rate of TCP-bare and ZWA-based schemes are found to be same in heavy traffic conditions, hence the ZWA-based scheme is not included in the comparison. It must be noted that during the time interval T0 to T1 as depicted in figure even if a single TCP segment takes an old and stale route then ZWA-based scheme fails to improve the performance. On the other hand, the spurious-timeout-based scheme works fine even if a few segments are lost due to the existence of old and stale routes during the time interval T0 to T1 and very few segments are captured by the controller during the handover period.

Figure 10 depicts the data transfer rate, the time interval 50–68 s depicted on the x-axis refers to the period before handover (before T0 as explained in the Fig. 5). The MN declares its disassociation between 68 to 69 seconds, during that period the sudden decrease in the rate of data transfer can be observed from the figure. Once MN declares its disassociation no data transfer is observed at MN. The halt time period for data transfer is from 69 to 73 s, in case of spurious-timeout-based scheme and halt time is from 69 to 74 s, in case of TCP-bare. Therefore traffic halt time for TCP-bare is relatively high compare to spurious-timeout-based scheme. It can also be observed that the Spurious-timeout-based scheme immediately attain the maximum transfer rate on completion of the handover. On

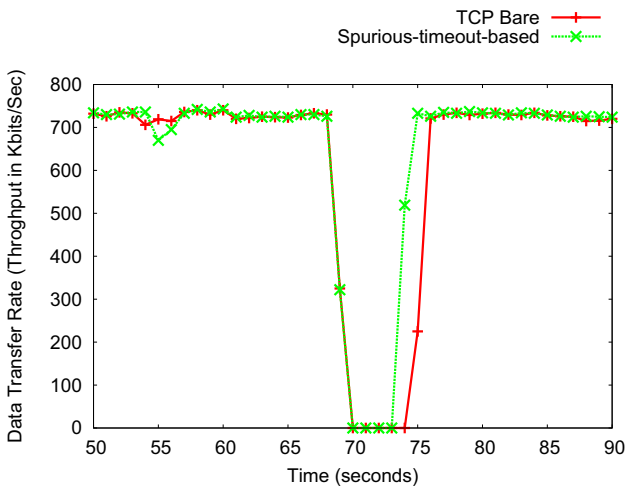


Fig. 10 A snapshot of comparative of data traffic

the other hand TCP-bare scheme consumes relatively more time.

In summary, the performance of the existing ZWA-based scheme is limited as it fails in some scenario where link delay is less and load of traffic is more. However, in the scenarios where the ZWA-based scheme performs well, it outperforms bare-TCP and performs comparatively similar to the spurious-timeout-based scheme. Clearly, the spurious-timeout-based scheme has no such restriction and substantially outperforms over bare-TCP and ZWA-based scheme under all traffic conditions.

4.1 Overhead analysis

Both presented approaches incur substantial overhead in maintaining the states (and corresponding buffered segments) for on-going TCP connections. However, seeing the current proxy/NAT servers, which can maintain the states of 30–35 thousand connections simultaneously, we believe that it can be handled. Also, experience of working with CAPWAP [40] wireless controller provides enough confidence that this overhead can be handled by a good server. According to the test bed evaluation presented in article [41], the Floodlight controller can handle 1,500,000 flows per second; even if there are 1,000,000 hosts per switch and 256 switches connected to each controller. Also, depending on the QoS requirements of the applications, SDN-assisted TCP support can be provided on need basis.

Considering the overhead at the controller, we measured the response time of each packet_in event. Figure 11 depicts the response time of the SDN controller. The performance is measured under different load that varies by increasing the density of nodes and increasing the number of TCP connections between CNs and MNs. The density of mobile nodes is expanded to 50 along with 5 dedicated CN to serve the MN. Additionally up to 60 pairs of iperf based

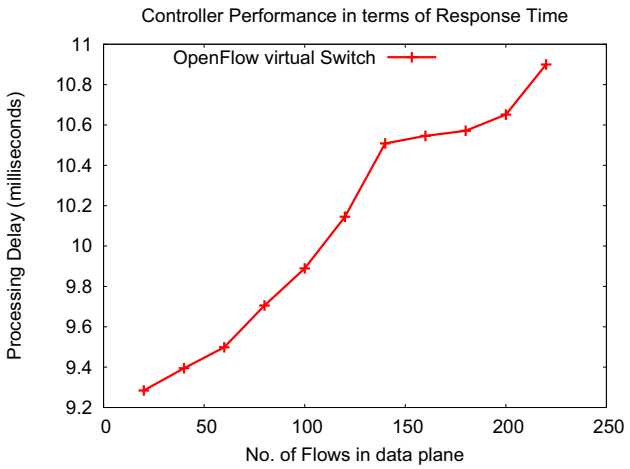


Fig. 11 Controller response time

fixed node TCP connections are also build to verify the scalability. It is known that the density of nodes also leads to more flows in the data plane and the increasing number of mobile nodes leads to frequent *packet_in* events. Figure 11 depicts that controller response time 9.2–11 ms by varying the amount flows in the data plane.

The SDN controller does not encounter any difficulties in responding to the mobility events of up to 215 nodes. However, The mechanism in which the data plane devices are emulated in mininet-WiFi it becomes cumbersome for us to accommodate 215 virtual hosts using Linux namespace and encounter error while expanding more. This is due to the sharing of system resources among various virtual hosts.

5 Conclusions

We have presented the design and implementation of our SDN-assisted TCP for IP-based wireless networks. In our approach, the SDN controller act as a mediator to tune the behavior of TCP during mobility in wireless environments. We have intelligently utilized the existing concept of avoiding spurious timeout to revive the state of the sender after the handover. The intervention of SDN controller assists the sender in preventing the effect of congestion control due to mobility and the mobile node (in the role of the receiver) can quickly re-initiate the TCP traffic after regaining the network connectivity. Our proposed design and its implementation exhibits following features (1) No change is needed in TCP implementation at the end hosts (2) The presence of controller is transparent to the end hosts, (3) Data Transmission can be re-initiated immediately after the re-association with the same rate as before the dis-association and (4) The end-to-end semantics of TCP is preserved. (5) The implementation is achieved with IEEE 802.11 radio channels.

The performance of the proposed protocol has been measured using an elementary topology and a limited number of mobile nodes (MNs). The results that are presented here suffer from a lack of test bed implementation. The scapy interface used to generate mobility events from the data plane requires approximately 30–60ms (time interval to process the IEEE disassociation frame and generate an appropriate notification for the controller). This is because of scapy uses python language which adds some extra delay processing. The firmware based solution may generate such notification faster in comparison of scapy based solution. For, an example in hardware-based SDN switches and APs, flow deletion for underlying changes in the data plane (between times T0 and T1 in Fig. 5) can be completed within 10 ms [42, 41]. Furthermore, after reassociation of the node, fresh flows can be added (between

times T1 and T2 in Fig. 5) within 10 ms [42, 41]. However, from T0 to T1, the existence of an old route causes data loss. In our scenario, data loss refers to the segments that are neither received by the receiver nor intercepted by the SDN controller.

Also, we expect the performance of our proposed approach to be better in a test bed. The presence of multiple processor cores helps the JAVA-based Floodlight controller support many mobility events. Many requests can be efficiently handled by utilizing the multi-threading attribute of the Floodlight controller. However, maintaining a separate dynamic queue to store and manage the segments for each TCP connection lead to extra memory and control overhead on the SDN controller. The extensive performance evaluation of the memory load (memory overhead for buffering the data at the controller) in the presence of a dense population of mobile nodes is beyond the scope of this paper.

References

1. Society, I.C. (1997). Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE Standard 80211-1997 http://ci.nii.ac.jp/naid/10011885684/en/.
2. Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, *16*(3), 1617–1634. https://doi.org/10.1109/SURV.2014.012214.00180.
3. Singh, K. V. K., & Pandey, M. (2016). Software-defined mobility in ip based Wi-fi networks: Design proposal and future directions. In *2016 IEEE international conference on advanced networks and telecommunications systems (ANTS), IEEE*, pp 1–6.
4. Bakre, A., & Badrinath, B. R. (1995). I-tcp: Indirect tcp for mobile hosts. In: Proceedings of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society, IEEE Computer Society, Washington, DC, USA, ICDCS '95, pp 136, http://dl.acm.org/citation.cfm?id=876885.880054.
5. Balakrishnan, H., Padmanabhan, V. N., Seshan, S., & Katz, R. H. (1996). A comparison of mechanisms for improving tcp performance over wireless links. *SIGCOMM Computer Communication Review*, *26*(4), 256–269. https://doi.org/10.1145/248157.248179.
6. Balakrishnan, H., Seshan, S., & Katz, R. H. (1995). Improving reliable transport and handoff performance in cellular wireless networks. *Wireless Network*, *1*(4), 469–481. https://doi.org/10.1007/BF01985757.
7. Brown, K., & Singh, S. (1997). M-tcp: Tcp for mobile cellular networks. *SIGCOMM Computer Communication Review*, *27*(5), 19–43. https://doi.org/10.1145/269790.269794.
8. Goff, T., Moronski, J., Phatak, D. S., & Gupta, V. (2000). Freeze-tcp: A true end-to-end tcp enhancement mechanism for mobile environments. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, IEEE3*, 1537–1545.
9. Sinha, P., Nandagopal, T., Venkitaraman, N., Sivakumar, R., & Bharghavan, V. (2002). Wtcp: A reliable transport protocol for wireless wide-area networks. *Wireless Network*, *8*(2/3), 301–316. https://doi.org/10.1023/A:1013702428498.
10. Ludwig, R., & Meyer, M. (2003). The Eifel Detection Algorithm for TCP. RFC 3522, https://doi.org/10.17487/RFC7323, http://www.rfc-editor.org/info/rfc3522.
11. Sarolahti, P., Kojo, M., Yamamoto, K., & Hata, M. (2009). Forward rto-recovery (f-rto): An algorithm for detecting spurious retransmission timeouts with tcp. In *RFC 5682, IETF, RFC*.
12. Gurtov, A., & Reiner, L. (2005). The Eifel response algorithm for TCP. RFC 4015, https://doi.org/10.17487/RFC4015, https://rfc-editor.org/rfc/rfc4015.txt.
13. promoted ONF (2014) Mininet. [Online]. *Available at*http://mininet.org/.
14. Fontes, R., Afzal, S., Brito, S., Santos, M., & Esteve, C. (2015). Mininet wifi: Emulating software defined wireless networks. In: 2nd International workshop on management of SDN and NFV systems, 2015 (ManSDN/NFV 2015), IEEE, Barcelona, Spain.
15. Networks, B.S. (2015). Floodlight. [Online]. *Available at*http://www.projectfloodlight.org/.
16. Bakre, A., & Badrinath, B. R. (1995). I-tcp: Indirect tcp for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems, ACM, IEEE Computer Society, Washington, DC, USA, ICDCS '95*, pp 136, http://dl.acm.org/citation.cfm?id=876885.880054.
17. Yavatkar, R., & Bhagawat, N. (1994). Improving end-to-end performance of tcp over mobile internetworks. In *1994 First Workshop on Mobile Computing Systems and Applications, IEEE*, pp 146–152.
18. Abdelmoniem, A. M., Bensaou, B., & Abu, A. J. (2017). Sicc: Sdn-based incast congestion control for data centers. In: 2017 IEEE International Conference on Communications (ICC), IEEE, pp 1–6, https://doi.org/10.1109/ICC.2017.7996826.
19. Binder, A., Boros, T., & Kotuliak, I. (2015). A sdn based method of tcp connection handover. In I. Khalil, E. Neuhold, A. M. Tjoa, L. D. Xu, & I. You (Eds.), *Information and communication technology* (pp. 13–19). Cham: Springer.
20. Ghobadi, M., Yeganeh, S. H., & Ganjali, Y. (2012). Rethinking end-to-end congestion control in software-defined networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks, ACM, ACM, New York, NY, USA*, HotNets-XI, pp 61–66, https://doi.org/10.1145/2390231.2390242.
21. Jouet, S., Perkins, C., & Pezaros, D. (2016). Otcp: Sdn-managed congestion control for data center networks. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, IEEE*, pp 171–179, https://doi.org/10.1109/NOMS.2016.7502810.
22. Lu, Y., & Zhu, S. (2015). Sdn-based tcp congestion control in data center networks. In: 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), IEEE, pp 1–7, https://doi.org/10.1109/PCCC.2015.7410275.
23. Lu, Y., Fan, X., & Qian, L. (2017). Eqf: An explicit queue-length feedback for tcp congestion control in datacenter networks. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, IEEE, pp 69–74, https://doi.org/10.1109/CBD.2017.20.
24. Vakali, A., & Pallis, G. (2003). Content delivery networks: status and trends. *IEEE Internet Computing*, *7*(6), 68–74.
25. Guo, W., Mahendran, V., & Radhakrishnan, S. (2018). Join and spilt tcp for sdn networks: Architecture, implementation, and evaluation. *Computer Networks*, *137*, 160–172.
26. Ford, A., Raiciu, C., Handley, M., Barre, S., Iyengar, J., et al. (2011). Architectural guidelines for multipath tcp development. *IETF, Informational RFC*, *6182*, 2070–1721.
27. Liu, Y., Qin, X., Zhu, T., Chen, X., & Wei, G. (2019). Improve mptcp with sdn: From the perspective of resource pooling. *Journal of Network and Computer Applications*,. https://doi.org/10.1016/J.JNCA.2019.05.015.

28. Singh, K.V., Gupta, S., Verma, S., & Pandey, M. (2019). Improving performance of tcp for wireless network using sdn. In *Proceedings of the 20th International Conference on Distributed Computing and Networking, ACM, ACM, New York, NY, USA, ICDCN '19*, pp 267–276, https://doi.org/10.1145/3288599.3288626.
29. ONF (2011). Openflow switch specification, version 1.1.0 (wire protocol) [Online]. *Available at*http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf, version 1.1.0 (Wire Protocol).
30. Paxson, V., Allman, M., Chu, J., & Sargent, M. (2011). Rfc 6298. Computing TCP's retransmission Timer.
31. Postel, J., et al. (1981). *Transmission control protocol rfc 793*. Informational RFC: IETF.
32. Braden, R. (1989). *Rfc1122: Requirements for internet hosts-communication layers*. Informational RFC: IETF.
33. Ludwig, R., & Katz, R. H. (2000). The eifel algorithm: Making tcp robust against spurious retransmissions. *SIGCOMM Computer Communication Review*, *30*(1), 30–36. https://doi.org/10.1145/505688.505692.
34. Jacobson, V., Braden, R., & Borman, D. (1992). Rfc1323: Tcp extensions for high performance. IETF RFC.
35. Scapy tool. [Online]. (2006). *Available at*https://scapy.net/.
36. Padhye, J., Firoiu, V., Towsley, D. F., & Kurose, J. F. (2000). Modeling tcp reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, *8*(2), 133–145.
37. Floyd, S., Mahdavi, J., Mathis, M. & Romanow, D. A. (1996). TCP Selective Acknowledgment Options. RFC 2018, https://doi.org/10.17487/RFC2018, https://rfc-editor.org/rfc/rfc2018.txt.
38. Network, E.S. (2014). Iperf. [Online]. *Available at*https://iperf.fr/.
39. Combs, G. (2019). Wireshark. [Online]. *Available at*https://www.wireshark.org/.
40. Calhoun, P., Montemurro, M., Stanley, D., et al. (2009). Control and provisioning of wireless access points (capwap) protocol binding for iEEE 802.11. IETF RFC5416.
41. Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., Smeliansky, R. (2013). Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central Eastern European Software Engineering Conference in Russia, ACM, ACM, New York, NY, USA, CEE-SECR '13*, pp 1:1–1:6, https://doi.org/10.1145/2556610.2556621.
42. Kobayashi, M., Seetharaman, S., Parulkar, G., Appenzeller, G., Little, J., Reijendam, J., et al. (2014). Maturing of openflow and software-defined networking through deployments. *Computer Networks*, *61*, 151–175. https://doi.org/10.1016/j.bjp.2013.10.011, http://www.sciencedirect.com/science/article/pii/S138912861300371X, special issue on Future Internet Testbeds - Part I.

Krishna Vijay Kumar Singh is an Assistant Professor in the Department of Computer Applications at Sikkim Manipal Institute of Technology (SMIT), Majitar. East Sikkim, India. He holds M.Tech degree in Computer Science (2011) from Sikkim Manipal Institute of Technology (SMIT), Majitar, Sikkim, India. Currently he is pursuing Ph.D. from Motilal Nehru National Institute of Technology Allahabad, India. He is the recipient of research fellowship under Quality Improvement Program (QIP) sponsored by AICTE, Govt. of India. His area of interests are wireless mobile networks, Software Defined Networking (SDN). He has three research publications to his credits for alleviating the problems of wireless network using SDN.

Dr. Mayank Pandey is an Associate Professor in the Department of Computer Science and Engineering at Motilal Nehru National Institute of Technology Allahabad (MNNIT), Allahabad, India. He has teaching experience of 16 years. He holds M.Tech degree in Computer Science (2002) from IIT, Kharagpur, India. He received his Doctorate in Computer Science (2012) from MNNIT Allahabad. He is the recipient of Young Faculty Research Fellowship from MeiTY, Govt. of India. His research interests are in the field of Computer Networks, Wireless and Mobile Networks, Distributed Computing, P2P Networks, Software Defined Networking (SDN) and Formal Methods. He has more than 25 research publications to his credits. He is a member of IEEE and ACM.