Improving Performance of TCP for Wireless Network using SDN

Krishna Vijay Singh
Sikkim Manipal University
Tadong, Sikkim
krishna.si@smit.smu.edu.in

Sakshi Gupta
Motilal Nehru National Institute of Technology
Allahabad, UP
gupta01sakshi@gmail.com

Saurabh Verma
Motilal Nehru National Institute of Technology
Allahabad, UP
saurabh2612.mnnit@gmail.com

Mayank Pandey
Motilal Nehru National Institute of Technology
Allahabad, UP
mayankpandey@mnnit.ac.in

ABSTRACT

This paper addresses the problem of non-optimal performance of Transmission Control Protocol (TCP) over Wi-Fi networks. TCP considers packet loss as a signal for congestion leading to reduced sending rate. This idea works well for the wired networks where majority of the packet losses are due to congestion. However, in wireless networks, packets are mostly lost because of transmission errors and mobility of the end hosts. Also, when packets are lost due to disassociation of the receiver from the network during mobility, sender remains idle for a significant period even after the receiver is re-associated. In this paper, we present the design and implementation of Software Defined Network (SDN) assisted TCP for wireless networks. In our approach, mobility induced packet losses are tackled smartly to avoid reduction in sending rate. Further, the data transfer starts immediately as soon as the receiver is re-associated to the network during mobility. Unlike existing solutions, our SDN assisted TCP does not violate the end-to-end semantics and no modification is required in the TCP protocol running at the end hosts. We have implemented and evaluated our approach using Mininet Wi-fi emulation platform. The results establish the applicability of our approach.

CCS CONCEPTS

• **Networks** → *Network experimentation*;

KEYWORDS

Software Defined Networking, TCP, Wireless TCP, Wireless

ACM Reference Format:
Krishna Vijay Singh, Sakshi Gupta, Saurabh Verma, and Mayank Pandey. 2019. Improving Performance of TCP for Wireless Network using SDN . In *International Conference on Distributed Computing and Networking (ICDCN '19), January 4–7, 2019, Bangalore, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3288599.3288626

ICDCN '19, January 4–7, 2019, Bangalore, India

ACM ISBN 978-1-4503-6094-4/19/01…$15.00
https://doi.org/10.1145/3288599.3288626

1 INTRODUCTION

The last few years have witnessed a significant and continuous increase in video traffic crossing global IP networks. In a recent survey [1], Cisco has estimated that IP video traffic would be 82% of all consumer internet traffic by the year 2021, up from 73% in 2016. Further, a significant percentage of the end hosts receiving this IP video traffic are mobile devices such as smart phones. Although mobile radio technologies such as 3G, 4G are getting more prevalent and cheaper, Wi-Fi still remains a preferable communication alternative for mobile users. Also, all the popular video streaming services (YouTube, Netflix etc.) are implemented using DASH (Dynamic Adaptive Streaming over HTTP) [31] which utilizes TCP as an underlying protocol. Thus, the performance of TCP during mobility in IP based Wi-Fi networks is a key for desired Quality of Experience (QoE).

Generally, the wireless connectivity to mobile devices is provided using Wireless Access Points (WAP or AP in short) running IEEE 802.11 Media Access Control (MAC) protocol [10]. These APs can be installed at different places with the help of backbone wired network and this setup is commonly known as Wi-Fi. Based on the size of the region to be made Wi-Fi enabled, the APs may be installed in a single IP subnet or in different IP subnets. Mobility in IP based Wi-Fi networks can be characterized either as mobility within a subnet or mobility across different subnets. Mobility within a subnet is termed as Layer-2 mobility whereas mobility across different subnets is known as Layer-3 mobility.

The major hurdle in achieving seamless mobility is the inability of the underlying network to tune itself according to mobility induced changes in the topology. In traditional networking devices (switches, routers etc.), both the control plane (for making routing decisions) and the data plane (for forwarding traffic to the next hop) are coupled together, thus making them less flexible. Over the last few years, Software Defined Networks (SDN) has emerged as a new paradigm which decouples the data plane and the control plane. In this paradigm the control plane functionalities are shifted to a logically centralized controller module. The controller module is programmable which facilitates the dynamic inclusion of policies according to the context. These features can be applied over the data plane for tuning the behaviour of network in flexible manner.

Using SDN paradigm, we have provided a solution for layer-2 and layer-3 mobility in IP based Wi-Fi networks in our earlier work [27]. We have observed encouraging results with respect to handover delay compared to Mobile IP [24]. Further, we were able to

reduce the data loss in UDP traffic in our SDN based approach. However, for TCP based applications we have observed large periods of interruptions and non-optimal bandwidth utilization.

TCP was originally made for the wired network and packet losses leading to absence of acknowledgements are considered implicit signals for initializing congestion control and retransmission procedures. Congestion control procedures drastically reduce the sender window (the amount of data that can be sent per RTT). Also, every missed acknowledgement doubles the retransmission timeout period which is responsible for large periods of inactivity.

As a design principle, TCP assumes that congestion has occurred if packets are dropped. This assumption is typically wrong for wireless networks as packets may be dropped due to transmission errors which are more there. Further, mobility itself can induce packet losses. Owing to these issues, numerous efforts such as ITCP [4, 5], SnoopTCP [5, 6], MTCP [8], FreezeTCP [14], WTCP [28] etc. have been made towards this body of knowledge. Some more efforts [23] have claimed to improve the performance of TCP in wireless environment. However, there are certain limitations which hindered the large scale acceptance and implementation in form of a deployable protocol. Some of these efforts either violates the end to end semantics of TCP or require the APs to be TCP aware. Further, in some efforts TCP protocol implementation is required to be changed.

The intent of the work presented in this paper is to utilize the network programmability provided by SDN for the design and implementation of TCP that is suitable for wireless environments. The aim is to intelligently hide mobility induced packet losses so that sender cannot enter in congestion control mode avoiding reduction in the number of packets that can be sent per RTT. Most importantly, this has to be achieved without doing any changes in the currently deployed TCP implementations. Further, the end to end semantics of TCP has to be preserved without making APs TCP aware. In our SDN based approach, we have utilized the concept of Zero Window Advertisement (ZWA) which is traditionally used by TCP for flow control. Further, SDN based ZWA approach is able to activate the sender quickly which otherwise remains idle after movement of end host from one AP to another.

We have implemented the proposed algorithms using Mininet-WiFi [12] and Floodlight SDN Controller [29]. Mininet-WiFi is a fork of Mininet [25] which provides excellent support both for OpenFlow [22] enabled switches and wireless access points. The OpenFlow protocol [22], defined by the Open Networking Foundation (ONF) is the most popular API for communication between a controller and forwarding devices (data plane). Our contributions can be summarized as follows:

(1) Design and implementation of mobility detection mechanism. This mechanism generates notification for a controller regarding disassociation of mobile node from AP and association to another AP. Consequent to this, controller updates the routes in forwarding devices according to the new location.
(2) Design and implementation of two modules in Floodlight controller to avoid the initiation of congestion control mechanisms by TCP running at ends. These modules help the SDN controller to act as mediator and hide the effect of mobility induced packet losses.

The paper is structured as follows. In Section 2, detailed description of proposed approach is presented. Section 3 outlines the relevant related work. Section 4 presents the implementation details, results and associated discussion. Finally, Section 5 concludes the paper with possible future directions.

2 PROPOSED APPROACH

Our proposed approach includes different procedures which have to be executed simultaneously by the SDN controller at different layers of protocol stack. The execution of these procedures is triggered after detecting the mobility of host from one AP to another. The actions associated with these procedures can be listed as:

- Deletion of old routes (on disassociation from old AP) and addition of new routes (on association with new AP) in infrastructure switches and APs to maintain the reachability of mobile host.
- Hiding the mobility induced packet losses to avoid TCP sender from entering into congestion control mode. To do this, SDN controller acts as TCP receiver during disconnection period. Further, when mobile host gets associated with new AP, SDN controller relinquishes the role of TCP receiver.

2.1 Mobility Detection and Route Establishment

The procedures used in our approach are described with the help of an example network architecture depicted in Figure.1. The actual network topology used for implementation is presented in Section 5 where experimental setup is described. In Figure.1, a Floodlight controller and few OpenFlow enabled infrastructure switches and APs are depicted. Further, there is one fixed node termed as Correspondent Node (CN) and one Mobile Node (MN). MN is in the role of receiver for the on-going TCP connection with CN.

One of the major challenge in our proposed approach is to detect the movement of mobile node. In wired scenario, it is easy for the Floodlight controller to detect the disconnection of any node from OpenFlow enabled infrastructure switches. The DeviceManager module with the help of LinkDiscovery module periodically checks the status of switch ports. However, in wireless settings, single wireless interface of AP can be used by multiple clients for network access. Capturing the movement (disconnection) of any node from AP is difficult as there is no evident change in wireless interface properties except some load reduction on AP. In Floodlight controller there is no existing module which can serve this purpose. In our initial effort [27], we have used gratuitous ARP [30] from the mobile node (both in periodic and adaptive manner) for capturing the movement. In the current implementation we have utilized special purpose mobility management frames available in IEEE 802.11 wireless LAN standard.

In IEEE 802.11 wireless LAN standard, mobile hosts use deauthentication or disassociation frames to declare their impending disconnection from AP. Similarly, association frames are used to establish connection with AP. However, the challenge is to propagate these frames towards controller in timely manner to trigger the flow deletion/addition process. OpenFlow standard currently

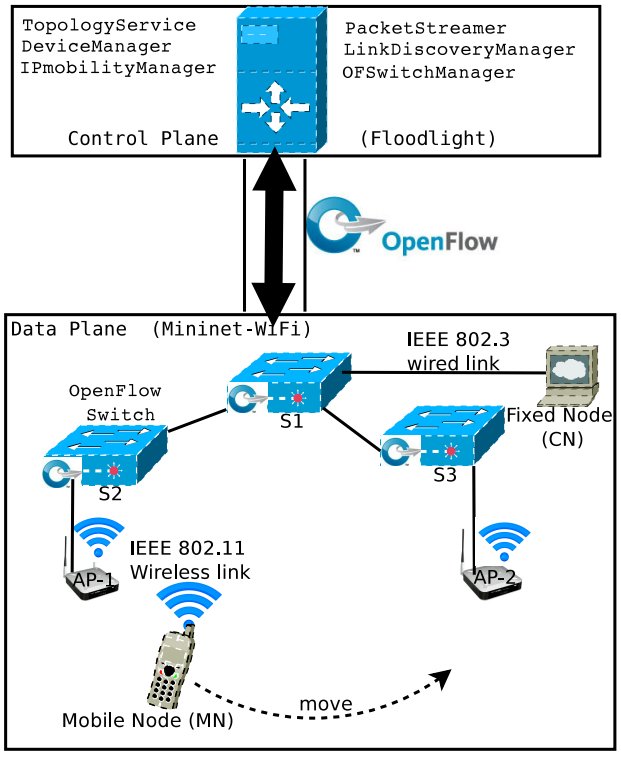


Figure 1: Example Network Topology

does not have support for 802.11 wireless frames. Also, all the available SDN controllers including Floodlight do not provide any procedure or module to capture IEEE 802.11 management frames. Ideally, OpenFlow enabled AP should have a provision to identify disassociation/association frames and forward these frames to controller. Although Mininet-WiFi provides a platform to emulate OpenFlow enabled APs, it does not have facility to capture and forward management frames towards controller.

We have written a patch for Mininet-WiFi which captures these frames and sends notification to the controller about connection or disconnection of hosts. Further, we have written a module *IPMobilityManager* which uses *DeviceManager* module of Floodlight to receive and process such notifications. The *DeviceManager* module of Floodlight maintains information base of wired hosts connected in network. This information base includes IP and MAC address, and point of attachment (Datapath ID of switch and port) of wired hosts. We have augmented *DeviceManager* module to include information about wireless hosts too. Similar to infrastructure switches a DPID is assigned to all the APs also. Our implementation ensures to have updated list of wireless hosts attached to any AP. This list is in form of a mapping between DPID of AP and, MAC and IP addresses of associated hosts. This mapping gets updated at every event of association or disassociation accordingly.

The association or disassociation notifications are forwarded to controller using *packet_in* event. This *packet_in* event is created intelligently at the infrastructure switch attached to AP. The notifications are in form of IP packets which are programmed to have IP and MAC addresses which cannot match with any existing flows in the switch. This automatically generates *packet_in* event which forwards notification to controller. The actions performed by our *IPMobilityManager* module at the reception of this *packet_in* event is described with the help of Algorithm.1.

Algorithm 1 Action performed by SDN controller on movement detection

```
1:  procedure actionOnMovement(messageType)
2:      if messageType == disassociation then
3:          remove device from the device manager information base;
4:          ofp_flow_mod_command to delete flows for MN;
5:          ▷ delete the flows related to disassociated MN from all infrastructure switches.
6:      else if messageType == re-association then
7:          device is added in the information base of device manager ;
8:          ofp_flow_add_command to add flows for MN ;
9:          ▷ Flows are added in infrastructure switches for associated MN to maintain its reachability.
10:     end if
11: end procedure
```

It may be noted that in Line-2 and Line-6 of Algorithm.1, controller takes actions depending on the notification message type. If message type is disassociation then flow deletion process is initiated by controller (Line-4) which deletes all the flows related to MN from infrastructure switches. To know which of the infrastructure switches contain flows related to MN, *Routing* module of Floodlight is used. This module provides unique datapath identification (DPID) of switches present on the shortest route between CN and MN. Similarly, if notification message type is association then flow addition process is initiated by controller (Line-8). This adds flows in the infrastructure switches lying on the shortest route between CN and current location of MN.

2.2 SDN assisted ZWA based TCP

The experimental results, described in [27] states that, SDN based mobility detection and route establishment is very fast, in comparison of non-SDN based approach using legacy switches. In our experiments, stated above, the mobile node in the role of receiver which moves from one AP to another. In the case of UDP traffic, the performance is much better than the legacy network. In the case of TCP traffic, the performance is again better but overall throughput is not as per the expectation.

After close inspection we concluded that in enterprise scale topologies (such as our experimental topology), the retransmission timeout (RTO) values are very less (minimum 200 ms[26]). Normally, the disconnection period is 1.8-2 seconds in SDN based route establishment, when transmission ranges of adjacent APs overlap with each other. This disconnection period increases in the case of non-overlapping transmission ranges.

TCP sender does not know that receiver is disconnected and it continues to send the segments. The absence of acknowledgement from MN triggers the RTO at TCP sender. Taking this missed acknowledgement as the signal of congestion, TCP sender reduces its window drastically. Further, when TCP sender retransmits the

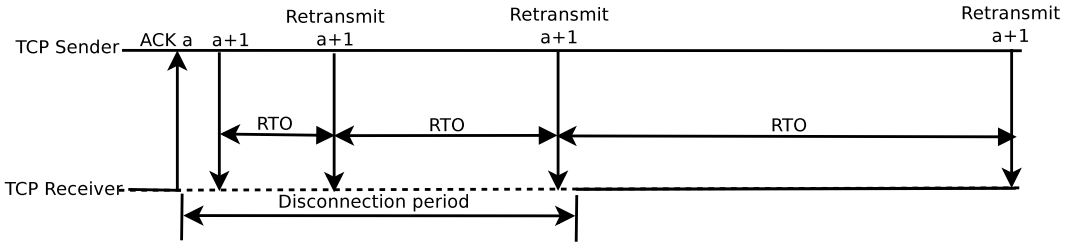


Figure 2: Successive timeout.

unacknowledged segment after timeout, it may trigger one more RTO (if MN is still not associated with new AP). This process may continue and after each successive timeout, RTO period is doubled. This may lead to a situation depicted in Figure.2.

It may be noted here that data transfer cannot be resumed even after re-association and route establishment. The successive retransmission attempts double the RTO value each time, forcing the TCP sender to remain idle. To summarize, the reasons for non-optimal TCP performance are (1) successive timeout leading to long idle period of TCP sender and (2) TCP sender triggering congestion control by reducing its window drastically after a timeout.

It is required to hide the mobility induced packet losses for optimal TCP performance. We propose a scheme to achieve this with SDN based approach. The SDN controller acts as TCP receiver temporarily during disconnection period. Further, Zero Window Advertisement (ZWA) mechanism of TCP is utilized to freeze the sender during the disconnection. This scheme is described next with its associated challenges, limitations and their solution.

In addition to reliable in-order delivery, TCP provides both congestion control and flow control. It uses a version of sliding window mechanism to achieve these objectives. The sender's window size is determined as the minimum of congestion window and flow control window. Congestion window is adjusted based on the implicit feedback from network in form of acknowledgements or missed acknowledgements. Flow-control window is set according to the advertised remaining buffer space from the receiver. Figure.3 depicts the variant of sliding window that TCP uses for flow control.

Here, the window size is represented by *SND.WND*. It gives the number of unacknowledged packets (between left edge and right edge of the window) at any time. *SND.UNA* represents first unacknowledged packet whereas *SND.NXT* represents the sequence number of next packet which can be sent without waiting for any acknowledgement. The usable window size is calculated as given in equation (1).

$$usable\_window = SND.UNA + SND.WND - SND.NXT \quad (1)$$

Accordingly, the usable window size in Figure.3 is $20 + 17 - 29 = 8$. Under normal operating conditions there can be two situations; (1) if the packets inside window (SND.WND) remains unacknowledged, the right edge is fixed, or (2) it moves towards right along with left edge as packets are acknowledged. The total size of the window (SND.WND) is advertised by receiver based on the remaining buffer space available at its end. In some situations, if receiver side process is slower than sender side, the receiver may run out of the buffer space eventually. In this case, receiver advertises a window of size zero. On receiving a zero window advertisement, sender should enter in persist mode and freeze all its retransmit timers. In persist mode, sender continuously sends zero window probes (ZWP) until receiver advertises non-zero window (the interval between successive probes grows exponentially).

We have utilized the zero window advertisement mechanism of TCP in our approach. We have already noted in Algorithm.1 that on the movement of mobile node MN, all the flows related to it are deleted from all the switches. After this event, any in flight packet which is destined for this disassociated node MN is forwarded to controller via *packet_in* method. At the controller, we have coded a module which acts as TCP receiver to issue a zero window advertisement after receiving any packet destined for the disassociated node.

Algorithm.2 describes the actions of controller when a segment is received for a disassociated node.

Algorithm 2 : Action by SDN controller when segment received for disassociated node

```
1:  procedure actionOnPacketInZWA(segment)
2:      for All incoming segments do
3:          if destination address of segment ∈ disassociated_address_list
    then
4:              for segments ∉ Controller.Buffer do
5:                  Store segment in Controller.Buffer
6:                  Generate ZWA for segment stored
7:              end for
8:          end if
9:      end for
10: end procedure
```

In our controller, a list of addresses of all disassociated nodes is maintained. It may be noted in Line-2 and Line-3 of Algorithm.2 that controller checks all the incoming segments against the list of addresses of disassociated nodes. Further, controller also maintains a buffer to temporarily store the segments destined for disassociated nodes. As may be noted in Line-4 and Line-5, only those segments which are not already stored are saved in buffer. This is done to avoid the storage of duplicate segments which can come in the case of long disconnection period. For each stored segment, a ZWA is issued from controller to the TCP sender.

In our initial phases of implementation, we had expected that a single ZWA is enough to force the sender to enter in persist mode (start sending ZWP) and freeze the associated retransmission timers. However, after examining the Wireshark traces we could not find

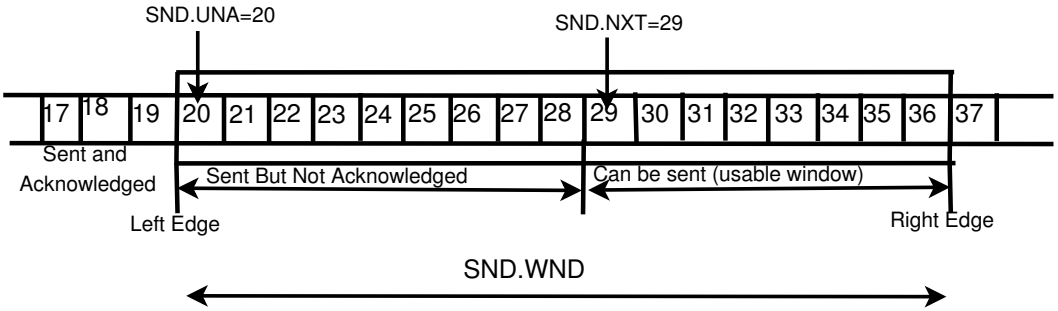


Figure 3: TCP Flow Control

ZWP segments coming out of TCP sender's network interface even after sending ZWA from the receiver. After careful analysis, we concluded that it is related to how different TCP implementations handles "window shrinking" which may result in negative usable window (equation.1).

For example, if suppose Figure.3 is taken as a snapshot of sender's window and a ZWA is received which acknowledges segment no. 25. At this instance, the values of *SND.UNA*, *SND.WND* and *SND.NXT* are 26, 0 and 29 respectively which makes $usable\_window = -3$. According to RFC793 and RFC1122, shrinking of window is discouraged. It is stated that TCP receiver should not shrink the window. However, TCP sender should be robust against any such window shrinking (which causes negative usable window).

We have tested our implementation for [15]. If *usable_window* is becoming negative TCP does not enter in persist mode. TCP sender enters in persist mode in response to that ZWA only which does not make *usable_window* negative. To counter this issue, we modified our controller module (which acts as TCP receiver) and started sending ZWA for every segment reaching at controller destined for disassociated node (Line-4,5,6 of Algorithm.2). After this modification we observed the following:

- For every ZWA received (which makes the usable window negative), *SND.UNA* is updated and left edge of the window shifts towards right. However, TCP sender does not enter in persist mode.
- Receipt of ZWA and subsequent updation of *SND.UNA* does not have any effect on congestion window size (which for normal acknowledgements gets incremented by 1 for every acknowledgement). As a result, right edge of the window remains fixed.
- Eventually, *SND.UNA* is equal to *SND.NXT* and *usable_window* becomes non-negative (*SND.WND* is 0 due to ZWA) according to equation.1. Consequent to this, TCP sender enters in persist mode, freezes its retransmission timers and start sending ZWP.

To summarize, our implementation is successful in freezing

TCP sender and avoided repeated RTOs. Also, all the segments destined for disassociated node are buffered at controller.

When the mobile node again re-associates with any other AP, controller adds fresh flows in infrastructure switches to maintain its reachability (Algorithm.1, Line 6-8). Simultaneously, it sends all the buffered segments towards mobile node which were stored during disassociation period. For this, we have made a module in the controller which imitates the TCP sender. In this way, the TCP connection is revived in transparent and seamless manner.

Limitations of ZWA based Scheme : The ZWA scheme described above performs really well as may be noted in Section 4, where implementation results are discussed. However, in few situations this scheme performs poorly to the extent that TCP connection has to be reset. The Fig. 6(b) depicted in results section shows the TCP connection reset in such situations. After careful analysis of time stamped Wireshark traces we figured out the reasons for this poor performance. As depicted in Figure.1, suppose a mobile node acting as TCP receiver moves from AP1 to AP2. Figure.4 depicts the timing sequence of various events related to disassociation and association of the mobile node. At time T0, mobile node is disassociated and this notification is sent to controller at time T1. Further, at time T2, all the flows related to disassociated node are deleted. Only after time T2, the segments destined for disassociated node reach at controller via *packet_in*. However, before time T2, the segments destined for disassociated node are routed using existing flows. This creates a possibility of arrival of segments at AP1 between time T0 and T2 when mobile node is disconnected. The segments arriving between time T0 and T2 are dropped by AP1.

The success of ZWA scheme is heavily dependent on arrival and non-arrival of segments at AP1 between time T0 and T2. Arrival of even a single segment (and its subsequent drop) can result in failure. In ZWA scheme all the segments arriving after T2 are sent to controller which imitates as TCP receiver and acknowledges (using ZWA) the receipt of these segments. Due to cumulative acknowledgement phenomenon followed by TCP, the TCP sender gets rest assured that all the segments upto the acknowledged segment are received correctly. This result in sliding of left edge of the sender's window and flushing of segments (which are acknowledged) from its buffer.

Clearly, all the segments arriving at AP1 between time T0 and T2 are dropped but this cannot be conveyed back to the TCP sender. Later on, when mobile node gets re-associated (between time T3 to T5) to AP2, all the buffered segments which arrived after time T2 are pushed from controller to mobile node. However, the mobile node (actual TCP receiver) can never get back the segments which are dropped between time T0 and T2. To summarize, the synchrony between TCP sender and actual TCP receiver gets broken in this situation which results in connection reset.

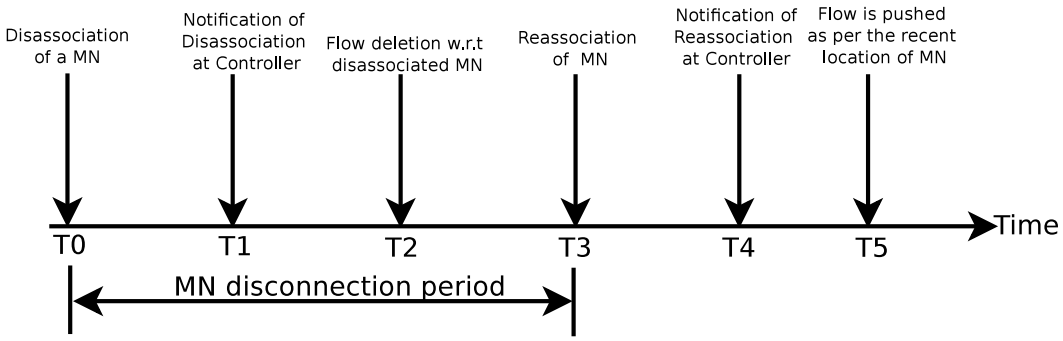


Figure 4: Timeline of events related to disassociation/association

Tackling limitations ZWA based scheme: The limitation of ZWA scheme described above are tackled by statistics module of controller. We make our TCP implementation adaptive according to the condition (Arrival and non-arrival of segments at AP-1 between T0 and T2). The behaviour of the controller module is described as follows:

- If any segment arrives between time T0 and T2, then Algorithm.2 is disabled (buffering of segments is not done and ZWA is not issued). In this case, segment transmission (and retransmission) is performed according to original TCP running at end hosts. Overall performance degrades but it never leads to TCP connection reset.
- Conversely, if no segment arrives between time T0 and T2, then Algorithm.2 is enabled. In this case, good performance is achieved without any problem.

In actual hardware based SDN switches and APs, disassociation scan and its notification to controller (between time T0 and T1 in Figure.4) can be done within 10 ms [17] [21]. Further, after disassociation notification, the related flows can be deleted ((between time T1 and T2 in Figure.4) within 10 ms [17, 21]. The total time spent between T0 and T2 may vary from 10 ms to 15 ms depending upon the response from controller. Hence, chances of packet arrival at the old AP between this duration are comparatively low. Hence, the percentage of the time when original TCP takes over is very less which may lead to good overall performance.

The major challenge is to find out whether any segment has arrived at old AP between time T0 and T2. All popular SDN controllers including Floodlight [2] have provisions to collect the switch statistics. The OpenFlow enabled switches are capable to notify the flow related information to the controller. The controller uses Read-State messages to collect statistics from the switch. These statistics are collected periodically in the form of port utilization and flow utilization. These statistics describe how many packets have been forwarded using a certain port or flow.

The statistics collection module of controller is modified for utilization of our purpose. We have written a module which can categorically specify whether any packet has been forwarded using a certain flow during certain time period (in our case between T0 and T2). For this purpose we collected the flow statistics at two timestamps. Once, at the time of when AP received a disassociation request, The AP sends a disassociation notification along with the flow statistics related to disassociated MN. The second statistics collection request is sent along with the flow deletion command issued by controller. However, When a flow entry is removed, the switch generally verify the flow entry using OFPFF_SEND_FLOW_REM flag [22]. If this flag is set, the switch must send a flow removal message to the controller. Each flow removal message contains complete description of the flow entry, the reason for removal like expiry or delete, the flow entry duration at the time of removal, and the flow statistics at time of removal. We have utilized this functionality of OpenFlow switch for collecting the flow statistics, firstly on disassociation of MN and secondly on flow removal in response to disassociated MN. Both the statistics responses are compared to decide the enabling or disabling of the Algorithm. 2.

The steps for flow statistics collection are described as follows

1. In Step-1 the disassociation event is sent from the mobile station and the OpenFlow enabled Access Points receives the disassociation event.
2. In Step 2 the statistics along with the disassociation notification is sent with the time stamp. The time stamp here represents the time of disassociation of a specific MN. At this time we set the OFPFF_SEND_FLOW_REM flag at the disassociated point.
3. In Step-3, a flow delete command with respect to disassociated MN is sent along with the flow statistics request to the access point which generated the disassociation notification. Other en route switches receive only the flow delete request from the controller.
4. Step-4 represents the access point sending its flow statistics upon flow deletion because the OFPFF_SEND_FLOW_REM flag is set which forces the disassociated AP to send the flow statistics along with time stamp to the controller.
5. Now, Controller compares the old flow statistics with new flow statistics received due to deletion of flow.
6. If the loss (difference of bytes) is found then original TCP (Bare TCP) is utilized.
7. Else if there is no loss (similar, bytes transmission recorded at both timestamp) of segment then Controller is permitted to freeze the sender by utilizing ZWA based approach.

3 IMPLEMENTATION DETAILS AND RESULTS

We have implemented our proposed methodology using Mininet-WiFi [12] and Floodlight Controller [29]. Mininet-WiFi provides excellent support for IEEE 802.11 based wireless access points where OpenFlow enabled switches are extended with wireless interfaces to act as access points. Further, it provides support for various mobility models such as random waypoint. These mobility models can be utilized to emulate the mobility patterns of mobile nodes moving from one access point to another. The emulated topology is similar to the one depicted in Figure.5.

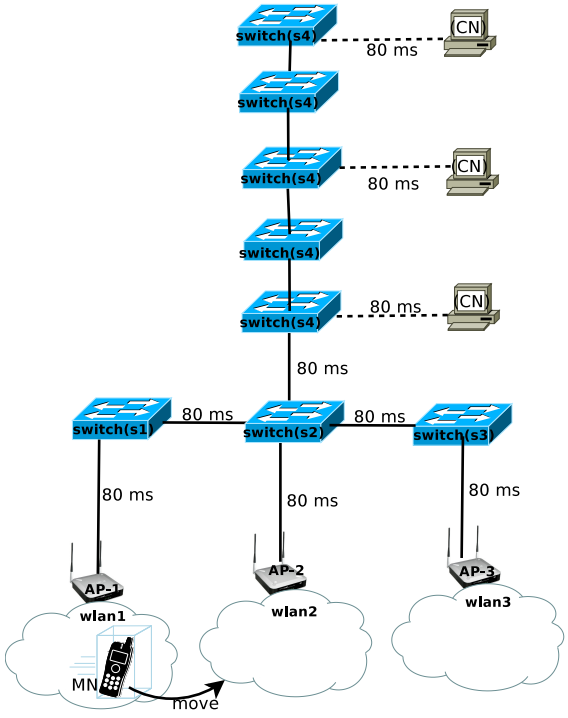


Figure 5: Emulated Topology

This emulated topology consists of 8 OpenFlow switches (S-1 to S-8). Three openFlow enabled wireless access points AP-1, AP-2 and AP-3 are attached to switch S1, S2 and S3 respectively. Further, the wireless channels 1, 6 and 11 have been assigned to these access points to minimize co-channel interference. The link delay of each link is varied from 1 to 80 ms. There is a reason behind setting the link delays to such large values. Large link delay and less window size is necessary to visualize better performance of SDN assisted ZWP based TCP scheme. We have implemented disassociation and re-association notification mechanism over Mininet environment which takes substantial time (around 100 ms) due to software APs and switches. To understand the performance of ZWA scheme and its associated pitfalls we have to set link delays accordingly. We have observed that SDN assisted ZWP based scheme perform more better with high link delay and less window. Further, as can be noticed in Figure.5, the point of attachment of fixed node(CN) can be changed to analyse the performance of proposed approach with respect to different hop counts between a fixed node (CN) and the mobile node(MN).

We have to do few more changes to clearly analyse the impact of our proposed approach. We figured out that the default TCP implementation in Linux kernel follows TCP Cubic [15] with SACK option [11] enabled. TCP Cubic sets initial congestion window size to 10 by default. Also, the window growth function is also aggressive in comparison to standard TCP implementations. We changed congestion control mechanism and window growth function according to standard TCP implementation (TCP Reno) and disabled the SACK option. Further, we set initial congestion window size to 1 as specified in TCP Reno.

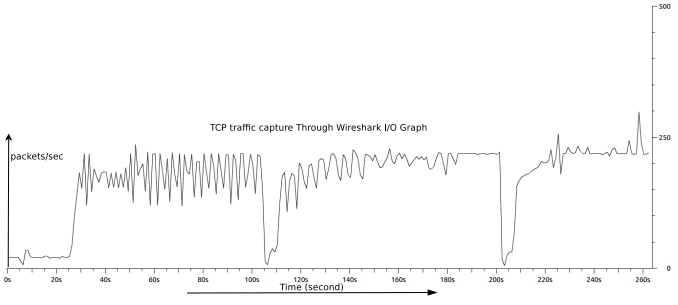
For performance evaluation we have used iperf [20] which is commonly used for network performance evaluation. The parameters such as throughput are analysed directly using iperf. Further, the changes in size of TCP sender's window are also recorded using shell script. We have also used Wireshark [9] to study the behaviour of TCP sender and receiver during handoff. We analysed all the segments captured in Wireless trace which are transferred between mobile node (MN) and correspondent node (CN). To capture the wireshark trace at the wireless station we have made an alias of the existing wireless interface (sta1-wlan0) and then enabled it in the monitor mode.

3.1 Results and Discussion

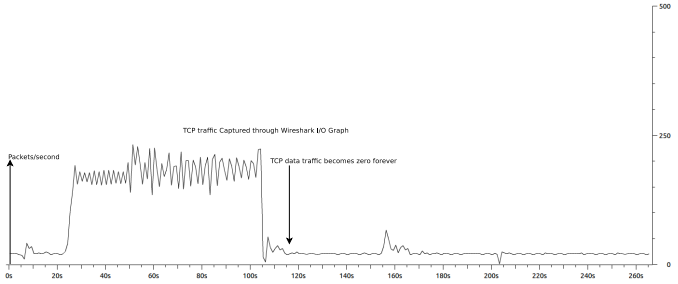
The performance of our proposed approach is evaluated in terms of TCP data transfer rate (throughput), idle period after disconnection, and number of retransmissions during disconnection. These values are recorded with respect to different disconnection duration. Further, different sets of results are recorded for varying hop counts between fixed node CN (in role of TCP sender) and mobile node MN (in role of TCP receiver).

Figure.6 depicts the traffic captured for bare TCP (TCP without any modification), and TCP using ZWA and TCP . In these results, TCP sender(CN) receiver(MN) are 2 hop away and link delays are set to 80 ms. Further, the disconnection period of mobile node is set as 1-2 seconds. In Figure.6(a), traffic captured for bare TCP is depicted. These results are recorded using default TCP (TCP Reno) and SDN controller is used only for mobility detection and route establishment as described in Section 3.1. It may be observed that significantly large periods of interruption (3-5 seconds) in data transfer are present. This is due to the successive retransmission attempts by TCP sender during disconnection period. These successive timeouts leads to long idle period for TCP sender even after re-association and route establishment.

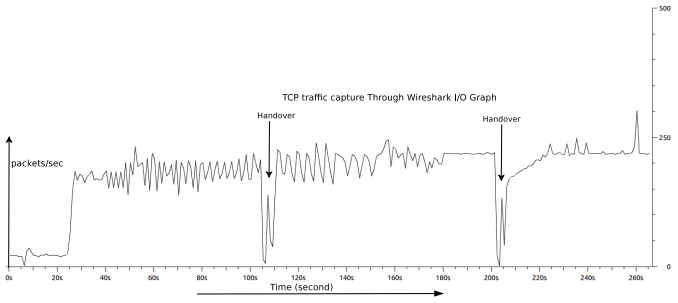
Figure.6(c) depicts the traffic captured for SDN assisted ZWA based scheme. In few runs of emulations, it is observed that TCP connection is not able to revive itself. after disassociation depicted using 6(b). As already discussed in Section 3.2.1, if any segment arrives between time T0 and T2 (when mobile node is disconnected, Figure.4), then there is no other option but to reset the connection. It may be observed that this scheme performs lot better than bare TCP. Data transfer is interrupted only for the time when mobile node is disconnected (1-2 seconds). As soon as mobile node re-associates itself, data transfer starts. Effectively, the interruption is only for the time period when mobile node is not connected.



(a) Bare TCP

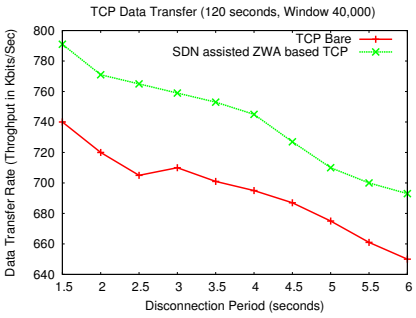


(b) Example where the scheme doesn't kick in Using ZWA

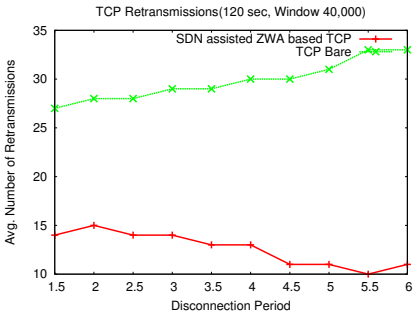


(c) TCP Using ZWA

Figure 6: Overall performance comparison.



(a) Bandwidth Utilization



(b) Number of Retransmissions

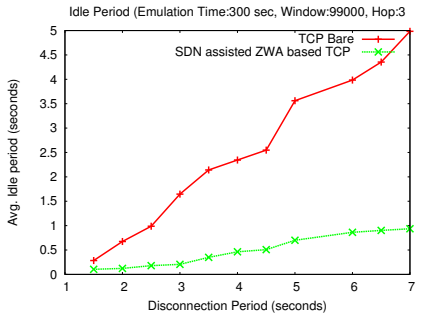
Figure 7: Bandwidth utilization and number of retransmissions

Figure.7 depicts the results collected for comparing bare TCP, SDN based TCP with respect to bandwidth utilization and number of retransmissions. For these experiments, TCP sender (CN) receiver (MN) are 2 hops away and link delays are set to 80 ms. Further, maximum sender's window size is set as 40,000 bytes. These settings are made to minimize the occurrence of the situation where ZWA based scheme fails. The Fig. 6(b) shows that TCP connection reset in such conditions. Bandwidth utilization and number of retransmissions are plotted against varying disconnection periods (1.5 to 6 seconds).

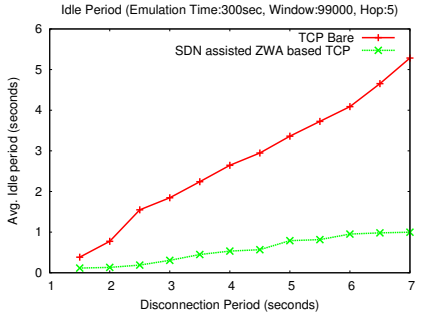
In Figure.7(a), it may be observed that bare TCP performs poorly in comparison to SDN assisted ZWP based TCP. Also, bandwidth utilization for all the approaches decreases with increased disconnection times. These results are expected because in bare TCP retransmission timeouts decrease the sender's congestion window which leads to non-optimal bandwidth utilization. On the other hand, in SDN assisted ZWP based TCP mobility induced segment losses cannot trigger congestion control mechanisms leading to reduction of sender's window size.

In Figure.7(b), it may be noted that average number of retransmissions are minimum in SDN assisted ZWP based TCP. This is due to the ZWA segments sent by controller which stalls the sender. As expected, in bare TCP, the average number of retransmissions is high. In the case of SDN assisted ZWP based TCP, the average number of retransmissions is less than bare-TCP because when the buffered segments are acknowledged and redirected to receiver from controller. Later on a receiver issues an acknowledgement which enables the sender to restart the transmission ahead of the left edge of sender's window. On the other hand, in bare-TCP the transmission starts from the left edge of the sender's window after the receiver is re-associated.

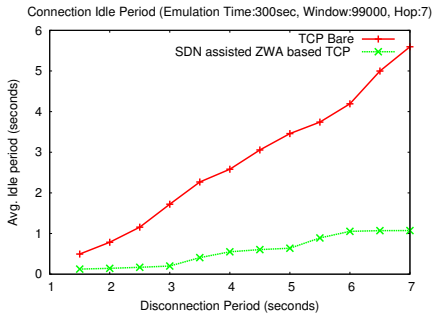
Figure. 8 presents the comparison of SDN based ZWA and bare-TCP with respect to time duration in which TCP sender remains idle even after re-association of mobile node to AP. In these results the idle period is plotted against the disconnection period. Figure 8(a), Figure. 8(b) and 8(c) depicts the results where TCP sender (CN) and receiver (MN) are 3, 5 and 7 hops away respectively. It may be noted that under all hop counts, SDN assisted ZWP based TCP performs very good in comparison to bare-TCP. Further, especially for long disconnection periods SDN assisted ZWP based TCP outperforms bare-TCP with respect to senders idle period.



(a) Connection Idle Period (Hop count 3)



(b) Connection Idle Period (Hop count 5)



(c) Connection Idle Period (Hop count 7)

Figure 8: Comparison with respect to connection Idle Period

4 RELATED WORK

After careful literature survey we could find only few efforts [3, 7, 13, 16, 18, 19] which have been made for tuning the TCP behaviour using SDN concepts. Except [7], all other efforts are devoted towards datacenter networks. In [7], attempts have been made to enhance the performance of TCP during change of connection endpoints.

In [13], an architecture of Open-TCP is suggested which utilizes the global network view and traffic characteristics available at SDN controller to dynamically change the TCP congestion control policies. The congestion control policies are periodically sent to end host which can change the TCP variant using a kernel level module.

In [18, 19] and [3], mechanisms for SDN based incast congestion control in datacenter are suggested. In [19], controller can select a long-lived flow to reduce sending rate. This is achieved by altering advertise window size of TCP acknowledgement segment when OpenFlow switch sends a congestion signal. In [18], the queue-length of congested switch port is sent as a congestion signal to controller to trigger TCP congestion control. Similarly, in [3] a system namely SDN-based Incast Congestion Control (SICC) is developed that can handle possible contention on the buffer space before the surge of incast traffic.

In [16], an architecture of Omniscient-TCP is presented which calculates suitable TCP retransmission timers, as well as the initial and maximum congestion window size that matches the route Bandwidth Delay Product between end-hosts. In [7], SDN mechanisms are used for efficient handing over the role of one of the two TCP endpoints to a third endpoint that was initially not a part of the communication. This scheme is proposed keeping in mind the Content Delivery Networks (CDN) where connection endpoints may change to select most appropriate server to serve specific content.

All these efforts have made significant contributions towards SDN enabled TCP. However, none of these efforts are made for wireless networks. To the best of our knowledge, ours is the first work which attempts to tune the TCP behaviour using SDN for wireless networks with node mobility. Further, most of these efforts require changes in TCP running at end points whereas in our approach no such change is needed.

5 CONCLUSION

We have proposed the design and implementation of SDN assisted TCP for IP based wireless networks. Our approach presented for tuning the TCP behaviour in wireless environments. In our approach, we have utilized the concept of zero window advertisement (ZWA) for freezing the sender during handover. The state of sender after handover remains preserved. The implementation results establish that our current implementation of ZWA approach well in all conditions.

Though, in some cases (for larger window and link with lower latency), TCP behaviour remains similar to the standard TCP. Hence, performance improvement in such cases is limited. We are currently exploring other existing features of TCP such as handling of spurious timeout discussed in (RFC 5682 and RFC 6582), to further improve our approach.

REFERENCES

[1] [n. d.]. Cisco Visual Networking Index: Forecast and Methodology, 2016âĂŞ2021. [White Paper]. *Available at https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf*.

[2] [n. d.]. Floodlight Project. [Online]. *Available at https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/*.

[3] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu. 2017. SICC: SDN-based incast congestion control for data centers. In *2017 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC.2017.7996826

[4] A. Bakre and B. R. Badrinath. 1995. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS '95)*. IEEE Computer Society, Washington, DC, USA, 136–. http://dl.acm.org/citation.cfm?id=876885.880054

[5] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. 1996. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *SIGCOMM Comput. Commun. Rev.* 26, 4 (Aug. 1996), 256–269. https://doi.org/10.1145/248157.248179

[6] Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz. 1995. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *Wirel. Netw.* 1, 4 (Dec. 1995), 469–481. https://doi.org/10.1007/BF01985757

[7] Andrej Binder, Tomas Boros, and Ivan Kotuliak. 2015. A SDN Based Method of TCP Connection Handover. In *Information and Communication Technology*, Ismail Khalil, Erich Neuhold, A Min Tjoa, Li Da Xu, and Ilsun You (Eds.). Springer International Publishing, Cham, 13–19.

[8] Kevin Brown and Suresh Singh. 1997. M-TCP: TCP for Mobile Cellular Networks. *SIGCOMM Comput. Commun. Rev.* 27, 5 (Oct. 1997), 19–43. https://doi.org/10.1145/269790.269794

[9] Gerald Combs. [n. d.]. Wireshark. [Online]. *Available at https://www.wireshark.org/*.

[10] IEEE Computer Society LAN MAN Standards Committee. 1997. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11-1997* (1997). http://ci.nii.ac.jp/naid/10011885684/en/

[11] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. 1996. TCP Selective Acknowledgment Options. RFC 2018. https://doi.org/10.17487/RFC2018

[12] Ramon Fontes, Samira Afzal, Samuel Brito, Mateus Santos, and Christian Esteve Rothenberg. 2015. Mininet-WiFi: Emulating Software-Defined Wireless Networks. In *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*. Barcelona, Spain.

[13] Monia Ghobadi, Soheil Hassas Yeganeh, and Yashar Ganjali. 2012. Rethinking End-to-end Congestion Control in Software-defined Networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets-XI)*. ACM, New York, NY, USA, 61–66. https://doi.org/10.1145/2390231.2390242

[14] Tom Goff, James Moronski, Dhananjay S Phatak, and Vipul Gupta. 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*, Vol. 3. IEEE, 1537–1545.

[15] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[16] S. Jouet, C. Perkins, and D. Pezaros. 2016. OTCP: SDN-managed congestion control for data center networks. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 171–179. https://doi.org/10.1109/NOMS.2016.7502810

[17] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan Van Reijendam, Paul Weissmann, and Nick Mckeown. 2014. Maturing of OpenFlow and Software-defined Networking Through Deployments. *Comput. Netw.* 61 (March 2014), 151–175. https://doi.org/10.1016/j.bjp.2013.10.011

[18] Y. Lu, X. Fan, and L. Qian. 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)*. 69–74. https://doi.org/10.1109/CBD.2017.20

[19] Yifei Lu and Shuhong Zhu. 2015. SDN-based TCP congestion control in data center networks. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. 1–7. https://doi.org/10.1109/PCCC.2015.7410275

[20] Energy Science network. [n. d.]. Iperf. [Online]. *Available at https://iperf.fr/*.

[21] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti. 2014. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys Tutorials* 16, 3 (Third 2014), 1617–1634. https://doi.org/10.1109/SURV.2014.012214.00180

[22] 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).

[23] Christina Parsa and J.J. GarciaâĂŘLunaâĂŘAceves. 2004. Improving TCP performance over wireless networks at the link layer. *Mobile Networks and Applications* 5, 1 (01 Apr 2004), 57–71. https://doi.org/10.1023/A:1019131822786

[24] Charles E. Perkins. 1998. Mobile Networking Through Mobile IP. *IEEE Internet Computing* 2, 1 (Jan. 1998), 58–69. https://doi.org/10.1109/4236.656077

[25] Open Networking Foundation promoted. [n. d.]. Mininet. [Online]. *Available at http://mininet.org/*.

[26] Pasi Sarolahti and Alexey Kuznetsov. 2002. Congestion Control in Linux TCP.. In *USENIX Annual Technical Conference, FREENIX Track*. 49–62.

[27] Krishna Vijay Singh and Mayank Pandey. 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)*. 1–6. https://doi.org/10.1109/ANTS.2016.7947808

[28] Prasun Sinha, Thyagarajan Nandagopal, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. 2002. WTCP: A Reliable Transport Protocol for Wireless Wide-area Networks. *Wirel. Netw.* 8, 2/3 (March 2002), 301–316. https://doi.org/10.1023/A:1013702428498

[29] Big Switch Networks sponsored community project. [n. d.]. Floodlight. [Online]. *Available at http://www.projectfloodlight.org/*.

[30] W. Richard Stevens. 1993. *TCP/IP Illustrated (Vol. 1): The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[31] Thomas Stockhammer. 2011. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 133–144.