An Improvement Mechanism for Low Priority Traffic TCP Performance in Strict Priority Queueing

Mahdi Rahimi and Vahab Pournaghshband
Advanced Network and Security Research Laboratory
Computer Science Department
California State University, Northridge
Northridge, California, USA
mahdi.r.rahimi@ieee.org
vahab@csun.edu

***Abstract*—Strict Priority Queuing (SPQ) is a widely used queuing method for applying preferential service on edge networks. SPQ, in presence of persistent high priority traffic, is known to behave in a very hostile manner toward low priority flows, causing them to experience poor TCP performance. This often leads to substantial packet losses, which consequently results in aggressive reduction in the congestion window size at the sender. This severe performance degradation then leads to channel underutilization when the high priority queue is emptied. In this paper, we introduce *Amicable Strict Priority Queuing*, an approach to improve the TCP performance of low priority traffic, in the presence of interruptions caused by a surge of high priority traffic flows. Our proposed approach has two major advantages. First, it does not require any changes to the TCP stack of the end hosts. Second, through the simple augmentation introduced in this paper, no modification to the existing SPQ-enabled switches and routers is required, making it readily deployable in current systems. Experiments in a simulated environment confirm the validity of our proposed approach.**

Keywords-Strict Priority Queueing; Freeze TCP; Differentiated Services; Quality of Service

I. INTRODUCTION

In the digital economy, a significant amount of all business-critical processes involve data transfers. Large corporations, retailers, publishing companies and other enterprises that have large repositories of data, produce high volumes of data, have multiple locations, or have high volume of international transfers both require and benefit from reliable and stable file transfers. Reliable file transfers can have a significant effect on the productivity and the reduction of the costs of companies, as file transfer inefficiencies cause major costs. In reality, however, when demands are high, large data transfers happen frequently, and resources such as network bandwidth are limited, all of the users may not receive satisfactory service, and some file transfers may experience poor performance and disruption.

Additionally, network service providers provide their services to a vast majority of users. However, they may not treat all of their users the same. For example, ISPs may have some special customers and provide better service to them, or categorize customers into different groups and prioritize the traffic of some groups over others. Video-on-Demand providers may favor customers who pay more or have purchased a premium service. Therefore, other users may experience lower quality of service when the favored customers are being served. The same scenario happens for cloud and big data storage providers, as they provide preferential service toward some customers. However, this comes at the expense of other customers', which motivates the companies into deploying a solution that alleviates this problem.

When resources are limited or preferential service is being provided, prioritization of traffic seems inevitable. Strict Priority Queuing (SPQ) is used as a primary method of providing such services on edge networks. SPQ classifies network packets as either priority or regular traffic, and ensures that higher priority traffic will always be served before low priority. Priority packets and regular packets are filtered into separate FIFO queues. The priority queue must be completely empty before the regular queue will be served. The advantage of this method is that high priority packets are guaranteed delivery so long as their inflow does not exceed the transmission rate on the network. The potential disadvantage is that a high proportion of priority traffic will cause regular traffic to suffer extreme performance degradation [1]. Fig. 1 illustrates an example of strict priority queuing; packets from flow 2 cannot be sent until the priority queue is completely emptied of packets from flow 1.

SPQ has a hostile nature toward lower priority flows. When the high priority traffic is present, the regular or low

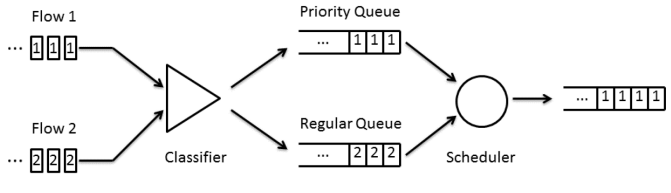

Fig. 1. A strict priority queue

priority traffic is not served. Therefore, in a regular TCP data transmission, the sender of the low priority traffic will not receive acknowledgement regarding the packets sent because the packets are accumulating in the low priority queue. In fact, they may never reach the receiver, or they may arrive so late that the sender may have already marked them as lost, retransmitted them, and adjusted its congestion window size to a lower value. Hence, when the retransmission timeout is expired, the sender feels congestion somewhere in the path and sets its congestion window size to the minimum possible amount (one maximum segment size according to Allman et al. [3]). Also, it stops transmitting new packets and retransmits the last sent packet in increasing intervals. Now, when the high priority traffic finishes, the low priority sender starts sending data but with a very low congestion window size. As the time passes, during the TCP slow-start, the sender gradually increases its window size until it reaches its previous amount. However, the sender loses a significant amount of channel utilization during this period. Fig. 2 shows this period. The vast amount of channel utilization that is lost is clear in the figure. When these periods of high priority traffic are abundant, the low priority traffic's predicament becomes even more dismal.

In addition, if the high priority traffic is large enough to saturate the link and long enough to continue rigorously for a period of time, no low priority packet will have a chance to pass SPQ. As a result, the low priority sender will not receive any acknowledgments and goes into a retransmission state. The sender, however, will not try forever. Instead, it will try to resend packets for a period of time and then finally it will give up, terminating the connection. This is the highest level adversity SPQ introduces to low priority flows. Braden [2] explains the procedure to handle excessive retransmissions of data segments. It introduces two thresholds *R1* and *R2* measuring the amount of retransmission that has occurred for the same segment. *R1* and *R2* might be measured in time units, or as a count of retransmissions. When the number of transmissions of the same segment reaches or exceeds threshold *R1*, TCP should pass negative advice to the IP layer, to trigger dead-gateway diagnosis. When the number of transmissions of the same segment reaches a threshold of *R2* greater than *R1*, TCP should close the connection. The value of *R1* should correspond to at least 3 retransmissions, at the current RTO. The value of *R2* should correspond to at least 100 seconds. For example, current implementations of Microsoft Windows uses a parameter called *TcpMaxDataRetransmissions* [4] which has a default value equal to 5. This parameter controls the number of times TCP retransmits an individual data segment before aborting the connection. Therefore, SPQ can cause termination of low priority connections.

In this paper, we propose a new approach to alleviate this problem that we call *Amicable Strict Priority Queuing (ASPQ)* to address these shortcomings. Our approach uses the built-in capabilities of TCP protocol and does not require any changes to either end hosts, or current existing SPQ middleboxes, such as commercial SPQ-enabled switches and routers.

We introduce a module named *ASPQ controller* which sits behind regular SPQ, and by harnessing the abilities of TCP protocol we provide a more reliable way to achieve the maximum possible amount of link utilization without dropping the low priority connections. We finally evaluate our approach through simulations.

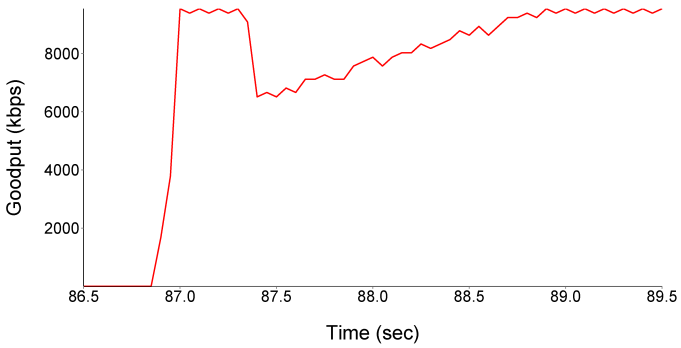

Fig. 2. Low priority traffic's slow-start state immediately after high priority queue is empty.

The remainder of the paper is organized as follows: Section II presents related work, followed by Section III, which discusses our methodology. Section IV offers our experimental results, and Section V concludes the paper.

II. RELATED WORK

Numerous works have addressed the shortcomings of SPQ [1,5,6]. Kang et al. [5] proposed an opportunistic priority queuing scheme for mobile broadband wireless systems, supporting both latency-sensitive real-time and best-effort non-real-time service. Qian et al. [1] built analytical models for traffic flows under strict priority queuing and weighted round robin and compare their service behavior. The authors report that SPQ is unfair since flows with low priority may be starved. On the other hand, WRR is capable of providing isolation to individual flows, and is insensitive to other flows' traffic patterns. Ferrari et al. [6] analyzed the priority queuing mechanism to verify its effectiveness when applied for the support of Expedited Forwarding-based services in the differentiated services environment. They adopted an experimental measurement-based methodology to outline its properties and end-to-end performance when supported in real transmission devices.

Freeze-TCP [7] is an end-to-end TCP enhancement mechanism for mobile environments. By exploiting the built-in capabilities of TCP protocol, such as zero window advertisement, it introduces an approach to address the problems that TCP protocol has raised in unstable mobile environments, allowing mobile clients to achieve full network capacity utilization. In this scheme, the receiver sends zero window advertisements to the sender when it senses an impending handoff. This freezes the sender, preventing retransmission timeout or packet loss. Since Freeze-TCP has been introduced, there have been numerous approaches that adapt the core idea into solving various problems in the field of networks [8,9]. Zaghal et al. [8] offer a scheme that performs loss-free rapid handoff

in mobile networks and switches IP address in the TCP/IP stack on both end-points. The authors adapted Freeze-TCP solution in their scheme as a way to avoid packet loss during handoff and to improve TCP performance. Pournaghshband et al. [9] introduce an approach to edge network control, called network dissuation, which makes particular uses of a network application intolerable, while providing acceptable services for approved network uses. In order to achieve this goal, they adopted a modified version of Freeze-TCP into their proposed "dissuade router". In their TCP model, if a particular stream uses up its assigned quota, the dissuade router attempts to persuade the sender to freeze its TCP congestion control parameters by sending zero window advertisements. As a result, the stream enters a silence period, where all further packets are dropped.

III. METHODOLOGY

As explained in Section I, a regular SPQ middlebox imposes significant performance degradation on low priority senders, as TCP senses congestion somewhere in the path, minimizing its congestion window size. As a result, the sender resumes the transmission with a very small congestion window size. This is a standard practice of TCP protocol because it wants to make sure that the sender only sends the amount of data that the link can tolerate. Hence, it forces the sender to start cautiously. Here, however, this TCP's slow-start mechanism unnecessarily holds the sender back since the high priority traffic has already left and the link is fully available for the low priority, but it naively starts slowly, losing substantial amount of the available bandwidth.

Our approach to address the shortcomings of SPQ is inspired by Freeze-TCP [7], an end-to-end TCP enhancement mechanism for mobile environments. In such environments, temporary disconnections (due to signal fading or other link errors or due to the movements of mobile nodes) can cause packet loss, which in turn triggers TCP's slow start mechanism. Freeze-TCP uses the zero window advertisement and the persist mode capacity of TCP protocol to mitigate this problem. In Freeze-TCP, if a mobile receiver can sense an impending disconnection, it will advertise a zero window advertisement on behalf of the receiver to force the sender into the persist mode and prevent it from reducing its congestion window. As a result, Freeze-TCP is able to improve the network channel utilization.

A. TCP Persist Mode

In regular TCP communication where a sender transmits data to a receiver, the consuming process on the receiver side may be slower than the sender. Therefore, the receiver's buffer starts filling up, and the receiver advertises progressively smaller and smaller window sizes. Eventually the receiver's buffer becomes full. Therefore, it sends a Zero Window Advertisement (ZWA) packet to the sender, stating that it cannot accept more packets. Upon receiving a ZWA packet, the sender stops sending data and becomes silent, entering a state called persist mode. Although the sender has stopped sending packets, similar to what the low priority sender does as explained before, there is a fundamental difference between them. Here, the sender keeps all of its current status including its congestion window size, ready to resume at full speed upon a signal from the receiver.

During the period of persist mode, the sender sends small packets called Zero Window Probes (ZWPs) to the receiver periodically, probing whether or not the receiver is ready to accept packets again. This probing is necessary since it precludes the creation of a deadlock. When the receiver has digested all of the received data and is ready to accept new data again, it sends the acknowledgement of the last packet it has already received, but this time, with a window size greater than zero. This shows the sender that the receiver is ready again. However, this acknowledgement may be lost somewhere in the path, as TCP only guarantees the receipt of original data, but not acknowledgements. Therefore, if the acknowledgement is lost, the receiver will be waiting for the sender, and the sender will be waiting for the receiver. As a result, the sender probes the receiver periodically to prevent a deadlock.

When the sender enters out of the persist mode and resumes transmitting data, it starts at full speed since it has kept all its previous status including its congestion window size. Another important characteristic of the persist mode is that, unlike a low priority sender facing SPQ, here, the sender does not terminate the connection. Instead, it continues to send ZWPs periodically for an indefinite amount of time, as long as the receiver acknowledges those ZWPs with new ZWAs.

B. Amicable Strict Priority Queuing

With these features in mind, we present an approach to a reliable and efficient version of Strict Priority Queuing: Amicable Strict Priority Queuing (ASPQ). By harnessing the special characteristics of the persist mode, we introduce a middlebox named ASPQ controller which sits behind a regular SPQ-enabled device (Fig. 4). An ASPQ controller observes the traffic from both directions, and is aware of the existence of any high priority or low priority traffic at any time. It has an internal table that keeps a list of all of the existing high and low priority traffic. In the absence of high priority traffic, low priority can have all of the bandwidth. However, upon the emergence of the first high priority flow, the ASPQ controller enters into a special state in which it takes the low priority senders into persist mode. By taking the low priority senders into persist mode, we are able to keep their congestion window intact, enabling them to resume at full speed. Furthermore, since the senders start probing ZWPs, they will not terminate the connection. Since it is necessary to respond to this probing, the ASPQ controller acknowledges those ZWPs on behalf of the receivers.

In order to put the low priority senders into persist mode, we must advertise zero window to them. In other words, they must receive at least one ZWA packet to go into persist mode. However, this ZWA packet must have special characteristics and cannot be any acknowledgment packet already sent by the receiver. Braden [2] states that TCP clients go into persist

mode when the window is shrunk to zero. However, in order to shrink the window to zero, the receiver must send a new ACK. In order to achieve this goal, we use in-flight ACKs to create ZWAs. The ASPQ controller captures the in-flight ACKs, converts them into ZWAs by setting their window size to zero, recalculates checksum, and then lets them pass.

The ASPQ controller always keeps a copy of the last ACK of each flow. When the last flow of existing high priority traffic finishes, the ASPQ controller immediately signals the low priority senders by sending the last ACK of their corresponding flow. Since the window size of the ACK is not zero, the senders see the ACK as a window update packet (unlike freezing the senders, unfreezing them does not require the ACK to be new). Hence, they stop probing, leave the persist mode, and start transmitting data at full speed.

As a result, Amicable SPQ brings the best possible performance out of low priority senders and maximizes channel utilization. Nevertheless, it does not impose any changes to the TCP stack of senders and receivers, or internal workings of a regular SPQ-enabled device, making it readily deployable in current systems.

IV. Experiments

We evaluated the performance of our approach against regular SPQ using Network Simulator 3 (ns-3) [10]. In our previous work [11] we implemented a strict priority queuing module for ns-3, which was not available before. We used this module alongside a new module for ASPQ controller in our experiments. We ran several experiments for two different scenarios to compare the TCP performance of low priority traffic in presence of Amicable SPQ to when only a regular SPQ is in place.

A. Experimental Setup

Our experiments included two senders and two receivers. One of the senders was chosen as the high priority sender and the other one was chosen as the low priority sender. Likewise, one of the receivers was reserved for high priority traffic and the other one was reserved for low priority traffic. Regular SPQ was installed on the rightmost egress port of its corresponding router, and the link connected to that was the bottleneck. The bandwidths of all of the links were equal to 50 Mbps except for the bottleneck which was 10 Mbps. The links' delays were different in different scenarios. For the first set of experiments they were set to 5ms. The size of the high priority and low priority queues were set to 240 packets which is a typical default queue size found in Cisco switches. Both of the high priority and low priority senders sent 100 MB of data, with the packets of 1000 bytes. With the start of the experiment, the low priority sender starts transmitting data and saturates the link. In order to interrupt the low priority traffic, after 2 seconds the high priority sender begins transmitting data and it continues the transmission until it is finished. Then the low priority sender resumes its transmission and goes on until the end of the experiment. Each experiment was run twice, once for ASPQ and once for regular SPQ. The topology of the

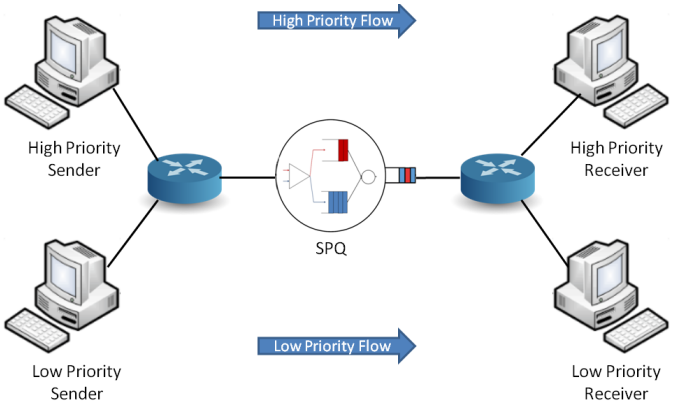

Fig. 3. The SPQ experiments topology

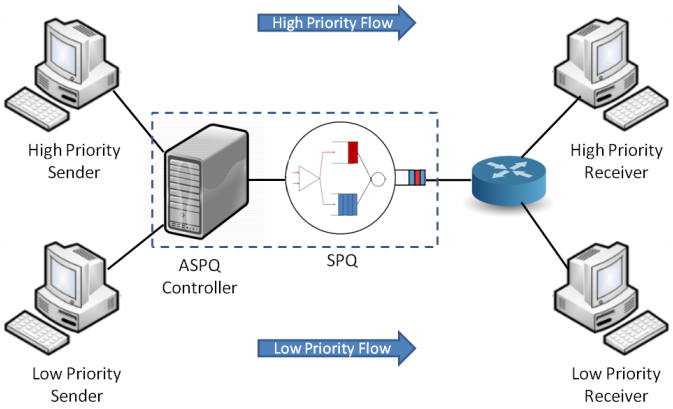

Fig. 4. The ASPQ experiments topology

regular SPQ experiments was the same as ASPQ's, except that ASPQ controller was replaced by a regular router. Fig. 3 and Fig. 4 illustrate the topologies of SPQ and ASPQ experiments respectively.

B. Results

First, we ran two experiments, one for ASPQ and one for SPQ. We were particularly interested in the period where high priority connection finishes transmitting data causing the resumption of the low priority traffic. Fig. 5 shows the results of both of the experiments for that period.

As it is clear in the figure, the low priority sender in Amicable SPQ leaves the persist mode promptly and reaches approximately the maximum available bandwidth utilization. SPQ, however, saturates the link for a short period of time, and after that starts slowly and struggles to reach the maximum link utilization again. The short successful period of SPQ is due to a flow of accumulated packets in the SPQ's low priority queue, which reaches the receiver first. After that, the actual packets from the sender arrive.

In the second scenario, we ran a set of experiments with different RTTs. The link delays of all of the links were set to 2ms, except for the link between the low priority sender and the ASPQ controller. Different sets of values ranging from

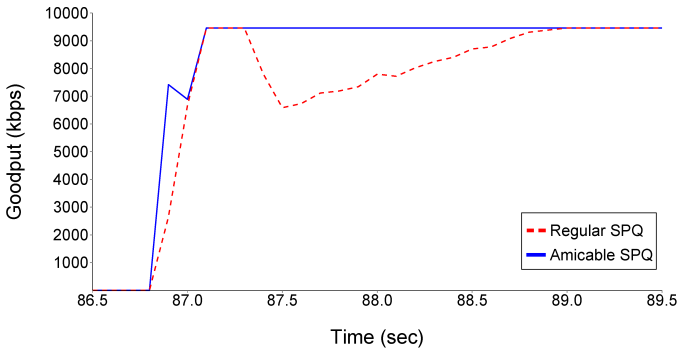

Fig. 5. Comparison of the performance of ASPQ against SPQ

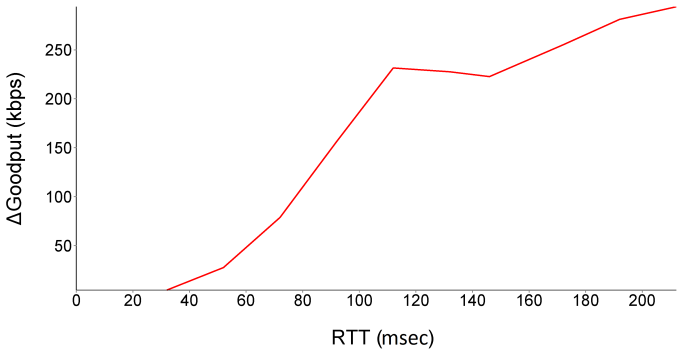

Fig. 6. Measurement of ΔGoodput against different RTTs

TABLE I
DETAILS OF MEASUREMENT OF ΔGOODPUT AGAINST DIFFERENT RTTS

RTT (ms)	ASPQ Avg. Goodput (kbps)	SPQ Avg. Goodput (kbps)	ΔGoodput (kbps)
30	4720.96	4716.42	4.54
50	4714.18	4686.63	27.55
70	4706.94	4628.15	78.79
90	4699.23	4542.95	156.28
110	4596.41	4364.91	231.5
130	4214.85	3987.26	227.59
150	3982.6	3759.93	222.67
170	3613.62	3358.5	255.12
190	3373.72	3092.56	281.16
210	3163.19	2869.17	294.02

10ms to 100ms were used for the link. The average goodput for the ASPQ low priority sender traffic and the SPQ low priority sender traffic was measured separately for each experiment and their difference was recorded as ΔGoodput. By measuring this value we investigate the effectiveness of ASPQ compared to SPQ. In addition, we attempt to analyze the effects of RTT on the performance of ASPQ and SPQ. Fig. 6 and Table I shows the results of our experiments.

We observed that the amount of ΔGoodput always remained positive as ASPQ performed better comparing to regular SPQ. Therefore, the results confirm the effectiveness of ASPQ. In addition, we observed that in larger RTTs, ASPQ performance was not highly affected, but the performance of regular SPQ deteriorated significantly. The reason is that in TCP's slow-start mechanism, the sender begins transmission of data with minimum congestion window size, and each time an ACK is received from the receiver, TCP increases the window size. As in larger RTTs the amount of time required for the ACKs to reach the sender is higher, it takes more time for the senders to pass the slow start state. Therefore, with the increase of RTT the quality of their performance deteriorates.

V. CONCLUDING REMARKS

In this paper we discussed the shortcomings of Strict Priority Queuing and presented a new approach to improve the TCP performance of low priority traffic. Furthermore, we investigated the effectiveness of our approach through a series of experiments via Network Simulator 3. The results confirmed that Amicable Strict Priority Queuing provided a better performance for low priority traffic.

As a future work, we intend to investigate how ASPQ acts in real networks by conducting a series of live experiments. The ASPQ controller will be implemented using Click Modular Software Router, a Cisco Catalyst 3750 switch will be used as a SPQ enabled middlebox, and PlanetLab nodes [12] will be used as the senders and receivers. In addition, we intend to implement our approach via Software Defined Networking (SDN). As SDN is offering a more central view of networks and is making networks smarter, we find it suitable for further research and experimentation.

REFERENCES

[1] Y. Qian, Z. Lu, and Q. Dou, "Qos scheduling for nocs: Strict priority queueing versus weighted round robin," in IEEE International Conference on Computer Design, 2010.
[2] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122. [Online]. Available: https://tools.ietf.org/html/rfc1122/ [Retrieved: November, 2015]
[3] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581. [Online]. Available: https://tools.ietf.org/html/rfc2581/ [Retrieved: November, 2015]
[4] "How to modify the TCP/IP maximum retransmission time-out," Microsoft Support Center. [Online]. Available: https://support.microsoft.com/en-us/kb/170359/
[5] C.G Kang, T.W. Kim, and J.H. Kim, "Adaptive delay threshold-based priority queueing scheme with opportunistic packet scheduling for integrated service in mobile broadband wireless access systems," IEEE Communications Letters, vol. 12, no. 4, 2008, pp. 241-243.
[6] T. Ferrari, G. Pau, and C. Raffaelli, "Measurement Based Analysis of Delay in Priority Queuing," Proceeding of IEEE Global Telecommunications Conference Globecom, 2001.
[7] T. Goff, J. Moronski, D. Phatak, and V. Gupta, "Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments," INFOCOM'00, 2000, pp. 1537-1545.
[8] R. Y. Zaghal, S. Davu, and J. I. Khan, "An interactive transparent protocol for connection oriented mobility performance analysis with voice traffic," in Proc. of 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, 2005, pp. 219-228.
[9] V. Pournaghshband, L. Kleinrock, P. L. Reiher, and A. Afanasyev, "Controlling applications by managing network characteristics," in IEEE International Conference on Communications (ICC), 2012.
[10] "The ns-3 Network Simulator," Project Homepage. [Online]. Available: http://www.nsnam.org [Retrieved: November, 2015]
[11] R. Chang, M. Rahimi, and V. Pournaghshband, "Differentiated Service Queuing Disciplines in NS-3," in The Seventh IARIA International Conference on Advances in System Simulation, 2015.
[12] "PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services", [Online]. Available: http://planet-lab.org/ [Retrieved: November, 2015]