APS: Adaptive Packet Sizing for Efficient End-to-End Network Transmission

Feixue Han*†, Qing Li†, Jianer Zhou‡†, Hong Xu§, Yong Jiang*†
*Tsinghua Shenzhen International Graduate school, Shenzhen, China
†Peng Cheng Laboratory, Shenzhen, China
‡Southern University of Science and Technology, Shenzhen, China
§The Chinese University of Hong Kong, Hong Kong, China

***Abstract*—Much effort has been devoted to improving the performance of network transmission. Yet, the impact of packet size which is limited by the 1500-byte maximum transmission unit (MTU) has not received adequate attention. Through comprehensive experiments, we find that jumbo frames which are commonly used as an alternate do not always yield the best performance under different transmission situations.**

In this paper, we elaborate on the limitations of the regular and jumbo frames and analyze how packet sizes affect network performance. Based on these, we present Adaptively Packet Sizing (APS), a dynamic packet size adjustment method that can be easily integrated into existing window-based congestion control algorithms. APS utilizes a machine learning method to predict the optimal packet size, which can minimize flow completion time (FCT) according to the instantaneous network condition. Besides, a packet size based priority mechanism is proposed to further improve the performance. We implement APS in both simulation and testbed environments. APS reduces the FCT by up to 50% and gains better performance in scenarios with various loss rates.

I. Introduction

Much effort has been devoted to improving the performance of data transmission, ranging from flow scheduling [1–3], congestion control [4–9], load balancing at various layers [10–12], to network stack optimization and re-design [13, 14]. Though many aspects of network transmission have been investigated, packet size receives little attention as it is bounded by the maximum transmission unit (MTU) setting in the link layer. The MTU is set to 1500 bytes according to the Ethernet standards [15], which was stipulated in 1981 based on the network properties (e.g. scale, propagation delay, forwarding delay, etc.) at that time. Since then, computer networks have undergone tremendous changes and one of the most notable changes is the explosive growth of data. As a result, the number of packets with 1500B MTU has increased dramatically, posing a great challenge to network devices [16].

On the other hand, most networking devices support jumbo frames that are larger than the standard 1500B (typically up to 9000B) [17]. Prior work has demonstrated that they achieve better throughput and lower flow completion times (FCTs) [16, 18–20]. Although promising, jumbo frames have certain limitations. 1) A 9000-byte jumbo frame takes six times longer to serialize than a standard frame. The serialization latency, which refers to the time needed to place the data on the physical wire, delays the acknowledgment (ACK) for the jumbo frame and impedes the congestion window growth. This can directly affect mice flows' FCTs which typically last just a few RTTs. 2) The cost of re-transmitting jumbo frames and the jitter are both higher [18]. These issues and their effects have not been thoroughly analyzed thus far.

The limitations of both standard and jumbo frames motivate us to find the optimal packet size for the best performance. We simulate the transmission process of flows under various network configurations and different packet sizes. Using FCT as the metric, the experiment result shows that the optimal packet size varies as the flow size or propagation delay changes. This confirms our intuition and reveals that the packet size should adapt to the transmission scenario.

Therefore, we introduce Adaptive Packet Sizing (APS), where packet size is dynamically adjusted during the transmission process of each (TCP) flow. To obtain the optimal packet size, we ought to consider various factors, however, some factors such as packet loss and their impact on FCT is difficult to quantify. Therefore, APS leverages the machine learning method to find the relationship between packet size and FCT under certain network status, and predict the optimal packet size (ops). APS also combines the priority mechanism with packet sizing to approximate the Shortest Remaining Processing Time (SRPT) discipline [21] and ensure fairness. Thereby, APS achieves lower FCT, especially for mice flows. Note that the packet size modification is independent of the congestion control mechanism. We make three technical contributions to building APS.

- **Analytical characterization:** We elaborate on the limitations of both the standard and jumbo frames, and analyze how packet sizes affect latency, rate growth, and packet loss in detail.
- **Ops prediction and dynamic adjustment:** With training data generated from the measurement of the sample flows in both simulation and testbed environments, we screen out the features with high impact factors and train an ops prediction model. To adapt to the real-time network situations, we continuously monitor the network status and adjust the

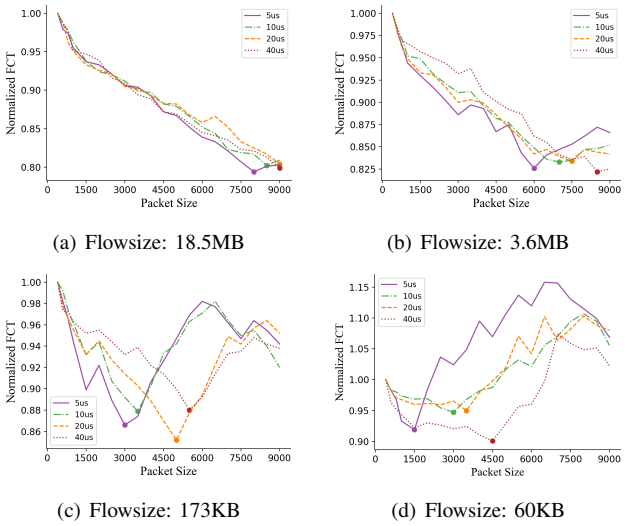
Corresponding Author: Qing Li, liq@pcl.ac.cn

978-1-6654-6824-4/22/$31.00 ©2022 IEEE

packet size every few RTTs.

- **Packet size based priority mechanism:** We specify the priority of each packet based on its size. Flows with few remaining bytes should be assigned with higher priorities according to the SRPT principle, and as our experiment result shows, they perform better with smaller packets. Hence, we assign higher priorities for smaller packets.

We implement APS in both the Linux kernel and ns3 simulation environments. In our experiments, APS is integrated with three CC algorithms: NewReno, Cubic, and DCTCP. We evaluate their performance under different network scenarios and workloads. The experiment result shows that compared to employing the standard 1500B packets, APS reduces the average FCT of overall flows and achieves a prominent optimization, especially for mice flows. When combined with the priority mechanism, APS yields a lower FCT by up to 50%. Besides, APS shows performance advantages even in scenarios with various loss rates.



(a) Flowsize: 18.5MB (b) Flowsize: 3.6MB

(c) Flowsize: 173KB (d) Flowsize: 60KB

Fig. 1. FCTs with different packet sizes under different per-hop latencies

II. Motivation and Background

To observe the influence of the packet size on FCT under different network environments, we run four flows in ns3 simulator. The topology is a simple dumbbell topology and the link bandwidth is 10Gbps. Different packet sizes and propagation latency configurations are adopted in different transmissions. Fig. 1 shows the FCTs of the flows, which are normalized to the value of 400B in each group, and the ops is marked with dots.

In the following, we will present the applicable scenarios for the standard frames and the jumbo frames according to the experiment results, and analyze their limitations separately.

A. Limitations of the standard frames

1) ***Poor performance on elephant flows****:* In Fig. 1(a), the FCT shows a decreasing trend as the packet size grows. When the per-hop latency is 20/40us, the minimal FCT appears at 9000B, with about 20% FCT optimization. Fig. 1(a) and Fig. 1(b) both show that for an elephant flow, transmitting data with larger packets can evidently reduce the FCT compared to the standard ones. This proves that the standard packets cannot guarantee the best performance for all the flow sizes.

2) ***Packet number explosion****:* With the advancement of Ethernet technology, the data transfer rate has been upgrading from 10Mbps in 1985 to 400Gbps in 2017. However, the maximum length of an IP packet from the network layer sent over Ethernet remains untouched as 1500 bytes. This indicates that an IP packet with more than 1500 bytes is fragmented into several fragments, which leads to explosive growth in the number of data packets and thus damages the network performance: **1) CPU overhead:** To receive the packets, there are hard and soft CPU interruptions introduced in the kernel. The hard interruption happens when the NIC receives the packets, and the soft interruption happens when the kernel deals with the packets. Those interruptions bring tremendous CPU overhead. **2) Power consumption:** Power consumption has been one of the main concerns for operators and is expected to become the main limiting factor for scaling the current network architectures [22]. Routers and switches are major contributors to network energy consumption. Published measurement-based studies [23–25] to date have shown that the power consumption of a switch increases fairly linearly from idle power (i.e., under zero load) to maximum power. (i.e., under full load). These prove that the reduction of the packet number can save CPU capacity and energy. Therefore, supporting the transmission of larger packets in the link is necessary.

B. Limitations of the jumbo frames

1) ***Limited performance on mice flows****:* From Figure. 1(d), we can see that with a flow size of 60KB and a per-hop latency of 5us, using 1500B packets can reduce the average FCT by about 15% over 9000B packets. In the range from 1500 bytes to 7300 bytes, the FCT continues to increase as the packet size grows, which indicates that not all the network scenarios and flows require jumbo packets.

2) ***Limited analysis for jumbo frames****:* Many researchers have made efforts in evaluating the performance of jumbo frames under different network configurations. Shaneel Narayan et al. [26] and Abhijit Das et al. [19] evaluated the performance of jumbo frames in both IPv4 and IPv6 networks. Their results both show that jumbo frames yield a throughput increase. Murray et al. [16] shows that jumbo frames have better resistance to packet loss and can gain faster TCP growth. JFEPM [20] aggregates the standard Ethernet frames as jumbo frames at the Top of Rack (TOR) switches and shows that the average FCT decreases as the packet size increases.

These works all elaborate the superiority of jumbo frames. However, as shown in Fig. 1(d), not all the scenarios match their conclusion, because their experiments have certain limitations: 1) The different initial cwnd (larger initial cwnd for a 9000B MTU connection according to RFC 5681 [27]) is a non-negligible factor that contributes to the outstanding

performance of jumbo frames. 2) Their analysis is based on the average FCT without distinguishing between the elephant and mice flows. 3) Their experiments usually run in high-speed, lossless networks, however, the experiment result may become different in links with low bandwidth and high error rates.

C. The optimal packet size changes

The above two subsections show that both the standard and jumbo frames have their own limitations, and the *ops* can be any value within the valid range. Comparing the four sub-graphs in Fig. 1, we can make a preliminary supposition on the changing pattern of *ops*. Under the same network configuration, the *ops* increases as the flow size rises, and when the flow size is determined, the best packet size tends to increase as the propagation delay grows. The intrinsic reasons for this phenomenon will be analyzed in Section III.

Supposing that we have already known how to select the *ops*, just determining the *ops* at the start of the transmission is insufficient. Because networks are complicated for their numerous applications and dynamic structures. Some of them have unstable links and connections (e.g. wireless sensor networks). They may frequently switch between the stable and unstable states for a limited number of consecutive packets [28]. This directly results in frequent changes in the RTT and loss rate of the network. Therefore, it's of great significance to constantly detect the network and make timely adjustments.

III. Why packet sizes influence performance

A. Influence on data transmission

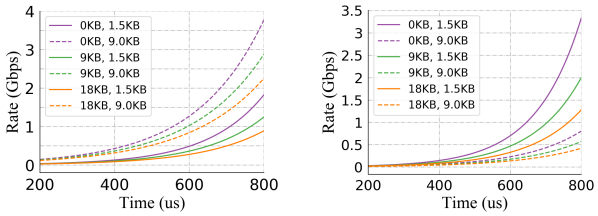
When a flow runs at a stable rate, regardless of the packet loss, the impact of packet sizes is reflected in two aspects.

TABLE I
COMPARE OF DIFFERENT FRAME SIZES

Packet	Serialization Delay (us)				Store-forward Delay (us)				Efficiency	
Size	100M	1G	10G	100G	100M	1G	10G	100G	IPv4	IPv6
1500	120	12	1.2	0.12	5.779	4.335	1.956	1.050	94.3%	93.6%
4500	360	36	3.6	0.36	6.018	4.661	1.972	1.054	98.1%	97.8%
9000	720	72	7.2	0.72	5.965	4.678	1.899	1.051	99%	98.9%

*1) **Transmission efficiency**:* Compared to the standard frames, jumbo frames promote transmission efficiency (denoted as E), since they carry more user data. Table I shows that the effective payload of a 1500B frame only accounts for 94.3%, but 99% in a 9000B frame under IPv4. The prevalence of IPv6 further expands this gap, because the transmission overhead of the IPv6 packet header is exacerbated and the transferable payload is reduced. This simple analysis on overhead shows that 9000B frames can achieve an approximate 5%-6% throughput promotion than 1500B frames.

*2) **Store-forward delay**:* As Table I shows, with the rapid growth of network speed, the serialization delay of a packet reduces to microseconds, and the gap of the serialization delay among different packet sizes is significantly narrowed. We measure the store-forward delay in iDetTrans510 switch interfaces with different rates. The result shows that the store-forward delay decreases sharply with the increase of port bandwidth, however, is hardly affected by the packet size. Considering that IPv6 is more computationally expensive, cutting down the packet number can significantly reduce the total processing delay.



(a) Different initial cwnd (b) The same initial cwnd

Fig. 2. The delivery rate changes with time. The first value in the legend refers to the queue length at the host and the second value is the packet size.

B. Influence on rate growth

We take the Reno algorithm as an example to demonstrate the effect of packet size on rate growth. In Reno, the $cwnd$ grows in two phases: the slow start phase and congestion avoidance phase.

In the slow start phase, the swnd is limited by the cwnd. Considering that the cwnd grows exponentially over time, with the RTT of a single packet T_1 and initial $cwnd$ I_c, the amount of data that the host can send within time t is:

$$D_t = I_c\{2^{\frac{t}{T_1}} - 1\} \tag{1}$$

The instantaneous sending rate is the ratio of the data sent in an infinitely short time to the passed time, hence we have:

$$\begin{aligned} v_1 &= E \lim_{h\to 0} \frac{D_t - D_{t-h}}{h} \\ &= I_c E \lim_{h\to 0} \frac{2^{\frac{h+t}{T_1}} - 2^{\frac{t}{T_1}}}{h2^{\frac{h}{T_1}}} \\ &= ln2 I_c E\{\frac{2^{\frac{t}{T_1}}}{T_1}\} \end{aligned} \tag{2}$$

According to Equation 2, the sending rate is proportional to E but decreases as T_1 grows. To present the impact of packet size on transmission rate more intuitively and prove that the I_c affects the performance, we plot the sending rate over time during the slow start phase with diverse packet sizes and NIC queue lengths. In Figure 2(a), refer to Google's standard, the I_c is set to 10 segments. Under this circumstance, the 9000B packets outperform 1500B packets because of the larger I_c. However, from a fair perspective, as Figure 2(b) shows, when adopting the same I_c (9000 bytes), the rate grows faster with a packet size of 1500B.

Then, during congestion avoidance, the $cwnd$ increases by a single segment in each RTT, that is, the sending rate grows linearly and the rate grows faster with the 9000B packets:

$$v_2 = v_1(t_1) + L_p E\{\frac{t - t_1}{T_1}\} \tag{3}$$

Although the analysis is based on the Reno algorithm, the impact of E and T can also provide a reference to other window-based CC algorithms.

C. Influence on packet loss and throughput

Packet loss due to drop-tail queuing and Random Early Detection (RED) [29] can cause great degradation in network performance [30]. With different packet sizes, the packet loss probability and the cost of re-transmission varies. The following analysis is based on the same bit error rate.

1) ***Loss rate****:* For all SACK-based TCPs, multiple losses in one RTT are treated as a single congestion signal. Therefore, dropping a 9000B packet or six consecutive 1500B packets has the same impact.The difference is that jumbo packets can hardly utilize the limited remaining space in the queue. For example, if there are only 8K bytes left in a drop-tail queue, a 9000B packet will be dropped, however, several 1500B packets can still be accommodated. Similarly, under the RED mechanism, the large packet is more likely to exceed the min-threshold and the max-threshold. Besides, the polynomial that was selected for the CRC32 algorithm has been tuned for frame sizes up to 1500B, which makes it less effective for larger frames and results in a greater latency before the error is detected. When an error is detected, re-transmitting jumbo packets will waste more occupied resources and re-transmit more bytes.

2) ***Throughput****:* Let p denote the number of congestion signals per ACK. The throughput is bounded by [30]:

$$TP < \{\frac{MSS}{RTT}\}\frac{1}{\sqrt{p}} \tag{4}$$

MSS means the packet size minus the length of TCP/IP header. This equation shows that the ratio of MSS and RTT will limit the upper bound of throughput. The relationship between the RTT and packet size is analyzed in Section III-A. A flow enters the congestion avoidance phase after packet loss, hence larger packets provide faster window growth.

IV. Design of APS

The key design of APS is to select the ops according to the network and flow status. However, due to the complicated network environment, predefined or empirical functions can hardly represent the relationships between transmission parameters and the ops. Therefore, we believe data-driven methods are promising to predict ops precisely. In addition to the learning-based ops prediction approach, this section also discusses the dynamic adjustment and priority mechanisms, which make APS complete and effective.

A. Selecting scope of packet sizes

The MTU on the data link layer is determined in the first three-way handshake stage when establishing a TCP connection. To minimize the modifications on the TCP stack, we regulate the packet size on the transport layer instead of the MTU. Nowadays, almost all the NICs at end-hosts and Gigabit switches (e.g. Cisco) support 9000-byte jumbo frames (not including the Ethernet header and CRC). Therefore, we set the MTU to the largest value (9000-byte) and keep it unchanged. During the transmission process of a single flow, we adjust the packet size within the range of the settled MTU.

Although all the packet sizes within 9k bytes are available, through our experiment, we find several sizes outperform other choices. We test the throughput of a long FTP flow using different MTUs with two identical computers connected via a Gigabit switch. We set the MTUs in the servers and the switch from 400-byte to 9000-byte in steps of 100-byte in each transfer separately. The result shows that the peak of throughput appears when the MTU is set to an integral multiple of 1500-byte, that is, the set of (1.5k, 3k, 4.5k, 6k, 7.5k, 9k), denoted as set ζ. This results from that the network stack, operating system, and driver behavior have been tuned to expect 1500-byte packets over several years. Therefore, the ops is selected from this set, and it can cover a certain range of network scenarios (e.g. one-hop latency from 5us to 20us).

B. Learning-based optimal packet size selection

1) ***Features****:* We consider nine features that are readily available or can be easily obtained, including avg_rtt, $base\_rtt$, lt_rtts, $loss$, dli_rate, $cwnd$, and re_bytes. The avg_rtt embodies the general network state over a long period. The lt_rtts means the latest three $rtts$, which can reflect the severity and changing trend of network congestion. In the first prediction, we fill these four values with the RTT measured during the second handshake. We update the delivery rate (dli_rate) with the ratio of data delivered and the time elapsed within a prediction interval [4], and the line rate is regarded as its initial value. We take each reduction of cwnd as a packet loss and record the total packet loss times. The ratio of lost packets is regarded as the $loss$. The remaining bytes (re_bytes) are the flow size minus the sent bytes and the $cwnd$ represents the real-time size of cwnd. The feature set is $9 \times N$, where N is the total data number.

2) ***Data collection****:* We collect the running results of millions of flows to train the prediction model. The distribution of the flow size is according to the CDF of the empirical WebSearch workload [31]. We run five long-lived FTP flows as the background traffic and select the packet size from the set ζ. The collected features and the FCT are recorded as an entry after each flow completion. In each network scenario, we can collect six entries for each flow size (corresponding to the six sizes in set s). We add the entry with the shortest FCT into our training dataset. Note that the FCT is used for data filtering, but not as a training feature. We adopt different data sets for different CC algorithms. The ratio of training-set, validation-set, and test-set is 7:1:2.

3) ***Model****:* A wide range of supervised classification techniques can be used for the ops selection. We expect to build a multi (six) classification model and adopt the ops as the label. We select XGBoost as it is a massively parallel boosting tree model with the advantages of high accuracy, fast convergence, and fast inference. It introduces the regularized loss function to avoid over-fitting. The weight of each new tree can be scaled

down by a given constant, which reduces the influence of a single tree on the prediction result.

We choose softmax as the objective function and train several XGBoost models with different sizes (i.e. the number of trees). During the training process, we monitor the importance of the features to the score. We find that the impact factor of min_rtt is 0, which means the feature is ineffective. Therefore, only the rest eight features are applied to train the final model.

C. Dynamic adjustment

From the perspective of network status, a growing number of applications are based on the partition/aggregation workflow pattern. In this pattern, flows usually arrive in a burst manner, which can cause severe congestion at the bottleneck and sharp fluctuation on RTTs. For the flows which cannot complete in a single packet, especially the ones with long life cycles, such fluctuation may happen several times during the whole transmission process. Considering each flow itself, along with the delivery of data, the remaining bytes gradually decrease and the cwnd changes according to the CC algorithm. These factors all can lead to the update of ops. For some specific flows, if we only adjust the packet size at each flow's arrival, the adjustment may achieve no optimization and even cause degradation to the FCT.

*1) **Workflow**:* Instead of adopting the pre-determined static packet size, we dynamically adjust the sending size at set intervals. At each flow's arrival, we first collect the required information for prediction and deliver the information to the packet size selection module. While waiting for the prediction result, the data is transmitted in 1500-byte packets. After getting the prediction result, we update the packet size with the prediction result and keep this size until receiving the next prediction value. This indicates that the prediction process and the transmission process are asynchronous.

*2) **Prediction frequency**:* In the workflow, it is crucial to determine an appropriate adjustment (prediction) interval. On the one hand, the computation expenses should be considered. Frequent predictions can bring much burden on the computing resources, and assuming that the network oscillates intensively in a short period, it is difficult to bring much performance improvement even if the adjustments are made in time. On the other hand, the network status can be hardly reflected in the packet size adjustment if the prediction interval is too long. Besides, flows on diverse paths have different RTTs, thus, they should adopt different adjustment intervals. For example, suppose there are two flows, F_a and F_b, starting from the same sender. The RTTs for F_a and F_b are about 2ms and 50ms separately. Under this scenario, an interval of 50ms is too long for F_a and too short for F_b.

Aiming at the above considerations, we associate the adjustment interval with the average RTT, which also acts as a feature for prediction. We maintain the average RTT and the last adjustment time for each flow. The average RTT is updated at each ACK's arrival and the adjustment time is updated when receiving the prediction result. When the sender receives an ACK, it checks whether $\alpha RTTs$ have passed since the last update (α is a hyper-parameter, setting the α to 10 is comparatively appropriate according to our experiment results). If so, the prediction module will be invoked.

D. Priority mechanism

APS performs well in reducing the average FCT under light traffic load. However, as the link load increases, the packets may get queued at the switches. Considering that a larger packet occupies more transmission time on the link than a smaller packet, it causes higher queuing time for the following packets, especially in low-speed networks. Therefore, we consider adopting multi queues with relative priorities and assigning the different size packets with the corresponding priority.

*1) **Assign priorities for packets**:* As shown in our previous analysis, under the same network environment, mice flows tend to choose smaller packet sizes in set ζ. This indicates that the ops of a single flow decreases as the remaining bytes decline. According to the principle of SRPT, flows with fewer remaining bytes should be allocated with higher priorities. The state-of-the-art SRPT-based designs priority of flows according to the priority label on the packet headers or a set of predefined thresholds. Considering the special characteristics of APS, we can emulate the SRPT principle more simply.

We use six priority queues to correspond to the six packet sizes in set ζ. The packets with less than 1500 bytes are placed into the highest-priority queue and the packets with more than 7500 bytes are placed into the lowest one, that is, the smaller the packet, the higher the priority. This design can guarantee that the mice flows have the highest priority and the priority of a flow gradually improves during the transmission process, which provides a good approximation to SRPT.

*2) **Determine the sending probability**:* There are K (6 in our experiments) priority queues Q_i ($0 \leq i < K$) where Q_0 has the highest priority. We assign each priority queue with a sending probability of p_i. Let θ_i be the percentage of bytes in Q_i. We use a discrete function $F(k)$ as the cumulative density function to denote the probability that the lowest priority of a flow is $\leq Q_k$. Let α_k denote the probability that the lowest priority of a flow is k. We have $\alpha_k = F(k) - F(k-1)$ ($K \geq 1$). We use T_j to represent the average time spent in Q_j, if flow x_j experiences the delays in different priorities down to the j-th priority, the average FCT is:

$$T(x_j) = \sum_{i=0}^{j} T_j \tag{5}$$

We aim to choose an optimal set of probabilities p_j to minimize the average FCT. The problem can be described as:

$$\begin{aligned} \min_{p_m} \quad & \Gamma = \sum_{m=0}^{K} (\alpha_m \sum_{i=0}^{m} T_i) = \sum_{m=0}^{K} (T_m \sum_{i=m}^{K} \alpha_i) \\ \text{s.t.} \quad & \sum_{m=0}^{K} p_m = 1, p_i > 0, i = 0, ..., K \end{aligned} \tag{6}$$

Assuming the M/M/1 priority queues, if the packets arrive with a rate of λ and the queue can consume μ 1.5k-byte packets per time unit, T_j can be expressed as:

$$T_j = \frac{1}{p_i\mu - \theta_i\lambda} \tag{7}$$

Since $\sum_{i=m}^{K} \alpha_i = \sum_{i=m}^{K} F(\alpha_m) - F(\alpha_{m-1})$, we can re-express the equation 6 as:

$$\begin{aligned} \min_{p_m} \quad & \Gamma = \sum_{m=0}^{K} \frac{1}{p_m\mu - \theta_m\lambda}(1 - F(\alpha_{m-1})) \\ \text{s.t.} \quad & \sum_{m=0}^{K} p_m = 1, 1 \geq p_i > 0, i = 0, ..., K \end{aligned} \tag{8}$$

Since Γ is represented by a multivariate function and there are many constraints, we use the sequential least-square programming optimization algorithm in Scipy to get the optimal priority set. Note that p_i is the probability for queue i to send a 1.5KB packet, we need to convert it into the probability of sending a packet in queue i. Therefore, the probabilities we finally adopt is:

$$\overline{p}_m = \frac{p_m}{\sum_{i=0}^{K} \frac{p_i}{i+1}} \tag{9}$$

*3) **Multi-queue ECN marking mechanism**:* ECN is originally designed with a single queue in mind and is not developed for multi-queue scenarios. Therefore, we design an ECN marking mechanism for the priority queues adopted in APS. Under the condition of multi queues and various packet sizes, there are two intuitions.

The first intuition is that the proportion of the ECN marked packets should not be affected by the packet size. Assuming that there are two servers (with the same bandwidth) sending 1500-byte and 3000-byte packets separately and the packets compete fairly, the ratio of the 1500-byte and 3000-byte packets on the link is 2:1. When these packets get queued in the switches and the queue length exceeds the threshold, the marked ratio is also 2:1, that is, the marking proportion is not disturbed by the packet size when the packets are congested in the same link.

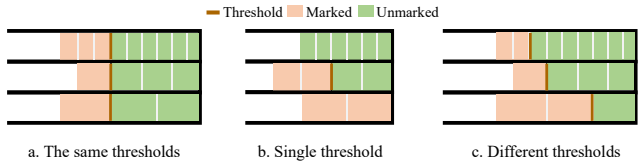


Fig. 3. Different threshold settings for ECN marking

The second one is the timely delivery of the ECN marking should not be affected by the different transmission probabilities. As Fig. 3–a shows, if we mark the packets according to the total length in the queues, the marked packets are always the ones with the lowest priorities, which is unfair to the large packets because they already have lower sending probabilities. If we assign each queue with the same threshold, as shown in Fig. 3–b, the ECN signals in the low-priority queues may not be delivered in time due to the lower sending probabilities, resulting in packet accumulation and even packet loss. Therefore, we allocate distinct thresholds for the queues following the principle of the lower the priority, the lower the threshold (as Fig. 3–c shows). This can effectively limit the queue length in the low-priority queues. Assuming the total capacity of the queues is V and α represents the proportion of the unmarked packets, the threshold in Q_i is set to $V \times \alpha \times p_i$.

V. Experiment Setup and Implementation

A. Experiment setup

*1) **Traffic loads**:* We simulate empirical workloads based on observed distributions in production datacenters. In particular, we utilize two flow size distributions from the WebSearch cluster [31] and Facebook's Hadoop [32] cluster separately. Mice flows account for a large proportion in the WebSearch workload and the workload in the Hadoop production clusters exhibits heavy-tailed characteristics.

The flows' arrival times follow the Poisson process and the source-destination hosts of each flow are randomly chosen uniformly. We keep the utilization of the links at 50-80%. ECMP is the default multi-path routing scheme.

*2) **Window-based CC selection**:* To observe whether APS can well adapt to different algorithms, we integrate APS with three window-based CC algorithms. Considering that our previous analysis is based on Reno, we choose NewReno as one of the adopted CC algorithms. Besides, to verify that APS can well adapt to different cwnd adjustment mechanisms, we also pick Cubic and DCTCP. Cubic is the default algorithm in the current Linux kernel, which shapes the cwnd in line with a cubic function, and DCTCP regulates the cwnd according to the percentage of ECN-marked packets. We collect the data of the flows running under these three algorithms.

*3) **Performance metrics**:*

Classification evaluation: We use the ratio between the number of correctly classified samples and the overall number of samples as the prediction performance metric. This measure is called $accuracy$, and it also works when labels are more than two (multi-class case). Since it is critical to make predictions within a time budget, we also pay attention to the inference latency.

Network performance: We mainly focus on the optimization of the average FCT, which is critical in improving the experience of users. We also try to verify whether the changing of the ops during a flow's transmission meets our expectations. What's more, we pay attention to the effectiveness of APS in resisting packet loss.

*4) **Experiment parameters**:* We have trained several XGBoost models with different sizes and find that the best configuration in our settings is using 50 trees with a max depth of 6. According to our experiment results, setting the α between 6 and 10 is comparatively appropriate. In our implementation, we invoke the prediction workflow every 10 avg_rtts. The initial cwnd of TCP in Linux is 10 packets by defualt. Predicting the ops at the beginning of flows is hard

to be accurate due to the limited information, so we set the initial cwnd of all the flows to 9K bytes uniformly.

B. Testbed implementation

To evaluate the performance enhancement of APS, we have implemented a prototype of APS and built a small testbed, consisting of four servers and two switches. Three servers act as the senders (one of them is used for generating the background traffic) and the other one as the receiver. Each server runs Ubuntu 16.04 with Linux kernel 4.4.0 and has the following hardware specifications: CPU: Intel(R) Xeon(R) Gold 5218, RAM: 32GB. The switches support 9018-byte jumbo frames, however, they do not support priority queues. Therefore, the experiments related to priority scheduling are carried out in the simulation environment. All the servers and switches are equipped with 10Gbps NICs. By default, advanced NIC offload mechanisms are enabled to reduce the CPU overhead. The base RTT of our testbed is around 150us and the MTUs of all the devices are set to 9k. In our implementation, the main modification of APS on the TCP stack can be divided into the following two parts.

The first part is the acquisition of the ops and the modification of the packet size in use, which is implemented in the kernel space. We use a hot-pluggable kernel module to control the enabling of APS, which provides great convenience for the on and off switching of APS. Since the TCP stack obtains the value through the communication between the kernel modules, this module also acts as the postman of the ops. After obtaining the ops, we compare the ops with the current available MSS (denoted as mss_cache in the kernel) and let *mss_cache=min(ops, mss_cache)*, which is operated at the point where the transmission layer passes the packets to the IP layer. Two reasons motivate us to modify the mss_cache. First, mss_cache does not interfere with the concrete CC design, thus different CC algorithms can be shifted easily while retaining compatibility with APS. Second, to alleviate the CPU overhead, most of the operating systems enable GSO/TSO by default. The essence of GSO/TSO is to postpone the fragmentation as much as possible until the data is transmitted to the NIC. If the data block is less than 64KB, it will be sent directly to the driver queue of the NIC. Then the NIC performs the segmentation according to the preset mss_cache in the skb of each flow.

The second part is the ops prediction logic, which is an application in userspace. When the stack judges that αRTTs have passed since the last prediction, it collects the information required for the prediction (most recorded in the tcp_sock) and invokes the prediction application through the $call\_usermodehelper$ function. This function regards the collected information as the parameters and starts the prediction application directly in the kernel.

GSO is based on software and TSO is implemented by hardware. The segmentation is first postponed in the network stack by GSO technology. If the NIC supports TSO, the fragmentation is performed in the NIC. Otherwise, the packet will be segmented just before it is pushed into the NIC (before calling the $xmit()$ function).

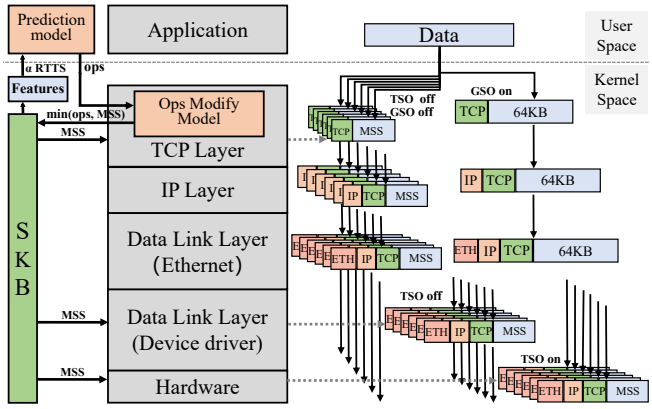


Fig. 4. Implementation of APS

Too much CPU resource will be consumed if the arrival of each flow triggers the ops prediction. Therefore, when a flow initiates a prediction request, we temporarily store the flow ID and the features. We take $\alpha \times base\_rtts$ as a time slice and integrate the N pieces of information collected in each time slice into a $9 \times N$ matrix as the input of the prediction model. After each prediction, we distribute the results to the flows according to their ID. In this way, we execute the prediction process only once in each time slice and greatly reduce the computational overhead.

Although the XGBoost model has the advantage of fast inference, the prediction time is still unacceptable for latency-sensitive applications. To minimize the impact on FCT, the prediction application starts and runs asynchronously with the kernel stack. That is, after calling the $call\_usermodehelper$ function, the TCP stack does not track the progress of the application anymore, and the packet size is modified when the stack is informed with a new prediction value.

C. Simulation:

In datacenter, the volume of traffic which go out to WAN is about 15% [32]. Despite the small fraction of WAN traffic, its impact on datacenter traffic is significant when both types of traffic are bottlenecked at the same switch [33]. To better understand the performance of APS when datacenter communicates with WAN, we connect a simple WAN topology with a datacenter topology. The loops in WAN and datacenter have different RTTs, thus they may use diverse packet sizes.

The datacenter topology is a small leaf-spine network of 8 × 8 to simulate the real architecture of data centers. This topology contains 8 leaf switches, 8 spine switches, and 128 hosts. We use 10Gbps point-to-point Ethernet links across our entire datacenter network. Each leaf switch has 16 downlinks to the hosts and 8 uplinks to the spine switches, forming a 2:1 over-subscription network and there is a WAN switch connecting to four hosts through 1Gbps links, which is also connected to a border switch of datacenter. Three types of workloads are generated in our simulation experiments: 1) Datacenter/WAN mixture of inter- and intra-datacenter traffic with the ratio of 1:5. 2) The simple WAN traffic. 3) Purely

datacenter traffic that reflects the most common cases in previous researches. Note that DCTCP is only implemented in the Datacenter environment.

We configure the intra-datacenter propagation and switching delay to 2us and 1us individually. Thus, the minimum RTT between two servers in different pods of a datacenter is about 30us, and in WAN the base RTT is about 12ms. The initial cwnd for all the flows is set to 9K bytes. The sending probabilities of the priority queues are (0.6872, 0.1678, 0.0639, 0.03796, 0.02688, 0.01622). We set V to 225KB and α to 0.65. Then the ECN marking thresholds are (42, 11, 4, 3, 2, 1) packets in each queue.

In our simulation experiments, to find a suitable adjustment interval, we assess the performance of several α values (from 3 to 15). To monitor the change of the network status during the transmission process, despite the FCT, we also track the queuing length in the NICs, the dynamic RTTs, and the change of the ops.

VI. Evaluation

A. Prediction accuracy and time budget

Larger models often yield higher accuracy at the cost of more memory and computation, and consequently, more crucially, higher latency for inference. Therefore, there is a trade-off between the model size and the prediction overhead. We find that using 50 trees with a max depth of 6 can give fast yet accurate results. The accuracy of the models trained with the data collected in the simulation and the testbed environment are shown in Table II.

TABLE II
ACCURACY OF THE XGBOOST CLASSIFICATION MODEL

Env \ Dataset	NewReno	Cubic	DCTCP
Simulation	94	91	93
Testbed	89	84	–

The mean latency of XGBoost is 3.135us on the CPU and 1.896us on a 2080Ti GPU, which means the inference latency can be driven down to a fraction of a prediction cycle. We evaluate the computing resource overhead of the prediction model in the use of CPU and GPU. When the model is invoked every $1.5ms$, the CPU occupation is about 1.6% and the resource consumption on GPU is about 1/40 of the CPU.

B. FCT optimization

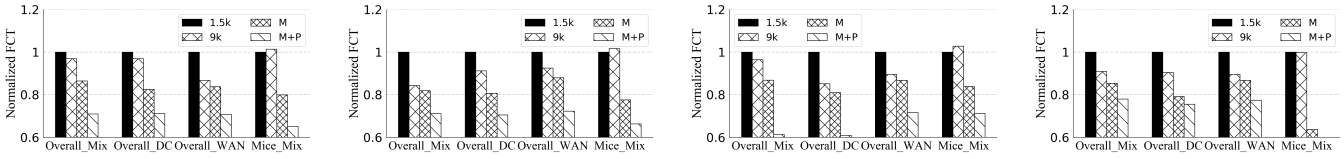
In the following analysis, we denote the experiment of transmitting data using 9K-byte and 1.5K-byte packets as *Jumbo* and *Standard* respectively. The APS and A&P (M and M+P in Fig. 5 and Fig. 6) stand for the simple APS mechanism and APS with priorities separately. Fig. 5 and Fig. 6 show the simulation performance of these four policies under two different workloads (WebSearch and Hadoop) and three CC algorithms (NewReno, Cubic, and DCTCP). We call the statistical data of these four methods in the same scenario as a group. There are four groups of data in each sub-figure and the data in each group is normalized to the *standard*. The labels in the x-axis show the types of flow and workload. *Overall* means the average FCT of all the flows and *Mice* is the average FCT of the mice flows smaller than 100KB. The *Mix*, *DC*, and *WAN* represent whether the traffic is a DC/WAN mixture, purely datacenter traffic, or purely WAN traffic. We collect the FCTs of over 30K flows.

In general, A&P always achieves the best performance under different workloads and algorithms and APS always outperforms the *Jumbo* and the *Standard*. Despite the chosen algorithm, APS and A&P have a relatively stable performance improvement. These indicate that APS can well adapt to various CC adjustment solutions. The *Standard* may behave better or worse than the *Jumbo* as the network scenario changes. This further confirms the conclusion that the ops alters as the network fluctuates and the necessity of dynamic adjustment. Besides, we can observe that A&P provides more evident optimization for mice flows (compared to all the flows), which reveals that prioritizing packets according to their size can provide a good approximation to SRPT.

In more detail, APS achieves up to 30% lower FCT for all the flows compared to the *Standard*, which is realized under the DCTCP algorithm and 60% fabric load. In particular, the benefits are more apparent in the average latencies of mice flows ([0, 100KB)) for these two workloads. Under 40% fabric load, APS can reduce the FCT of the mice flows by up to 40%. In the datacenter environment, APS has a prominent behavior as the FCT optimization is usually between $20\% - 30\%$, however, this optimization reduces to about 15% in WAN. When there is no priority mechanism, the FCT optimization mainly comes from the promotion of transmission efficiency and the reduction of packet header processing time.

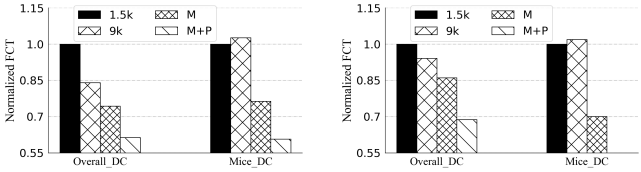
The ops is not only affected by the flow size but the network status, so does the packet priority in A&P. Since the packet size are obtained during the processing of the packet header, no additional comparison and calculation is required in the process of packet enqueuing. A&P further expands the optimization for both *Overall* and *Mice* flows. Compared to the *Standard*, A&P improves the average FCT by 30% to 50% under different scenarios. The maximum optimization is obtained under the Cubic algorithm and Hadoop workload. Observing the average FCT for the mice flows, we can find that the optimization is more obvious in the Hadoop workload because the elephant flows take a larger part in the Hadoop workload and the priority mechanism provides a good approximation of SRPT. Besides, Cubic and DCTCP outperform NewReno, and DCTCP has a better behavior in datacenter networks.

In the prototype implementation, since the switches do not support priority queues, we only tested the performance of *Standard*, *Jumbo*, and APS. We conduct our experiment with the default TCP SACK algorithm in Linux. The result shows that APS has a performance improvement of 19% and 15% compared with *Standard* and *Jumbo* respectively, which is not as good as the simulation performance. According to our



(a) NewReno_WebSearch (b) NewReno_Hadoop (c) Cubic_WebSearch (d) Cubic_Hadoop

Fig. 5. FCTs under different workloads and algorithms.



(a) DCTCP_WebSearch (b) DCTCP_Hadoop

Fig. 6. FCTs under DCTCP algorithms and datacenter environment.

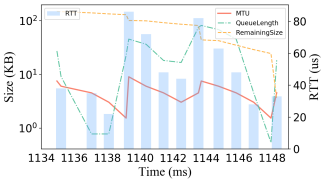


Fig. 7. Ops varies in a flow life cycle.

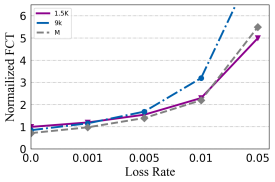


Fig. 8. Loss rate influences the FCT.

analysis, this is due to the difference in implementation of the network stack and the time overhead of the communication between user and kernel space.

C. The ops changes in a flow life cycle

Fig. 7 shows the change of the network and the flow status over a flow's life cycle. Considering that there are a number of flows running in the network simultaneously, to facilitate our analysis, we choose an elephant flow as the representative one.

As the orange line shows, the remaining size of the flow gradually reduces during the sending process. The change of the queue length at the sender server is shown in the green line, which grows rapidly when a great number of flows arrive. The red line represents the change of the *ops* (only selected from the predefined set ζ). What's more, the blue bar is the latest RTT when updating the size. The base RTT is around 20us. It embodies not only the fabric delay but also the host delay, which can be observed from the effect of the queue length on RTTs.

As shown in the figure, from $1135us$ to $1139us$, with the decreasing flow size and the low queue occupation, the sending size gradually becomes smaller. At $1139us$, the length of the queue and the RTT sharply grows, which means flows arrive at the host and congestion may occur at the network. This leads to an increase in the sending size. After a period of transmission, packets in queue are consumed and the congestion is relieved, so the sending size is reduced again until $1144us$. Note that at $1148us$, although the RTT increases again, the sending size decreases because there are only a few bytes left.

D. Performance with packet loss

To evaluate the effectiveness of APS under packet loss network environment, we show the performance of the *Standard*, *Jumbo*, and APS under gradually increased loss rates. Fig. 8 shows the variation of FCTs under different loss rates in a 10Gbps datacenter network. The values have been normalized to FCT in *Standard* with no packet loss. According to the figure, high loss rates have a huge impact on performance since the FCT grows sharply as the loss rate increases, due to the duration required to recover lost packets and rebuild the TCP window.

With the same initial *cwnd*, *Jumbo* is more affected by packet drops and has the worst performance from the loss rate of 0.5%. The FCT is even 8-9x higher than the initial value at a loss rate of 5%. Although the loss rate is treated as a feature in the prediction model, APS still outperforms *Standard* until the loss rate reaches 5%. While large packets occupy a certain proportion in APS so that the cost of packet retransmission is still relatively high, resulting in a worse performance.

VII. Conclusion

The data transfer rate of the Internet has increased many folds. However, the packet size has been unchanged for many years as it is limited by the 1500B MTU. In this paper, we prove that 1500B MTU has become a limitation to data transmission while adopting Jumbo frames does not always achieve the best performance. Therefore, we propose that the packet size should adapt to the real-time environment. By leveraging the *ops* prediction model, we show that dynamic packet size adjustment has an evident improvement on the network performance. This method greatly optimizes the FCT of the flows, especially for the mice ones. We envision that this work will motivate further investigation on the influence of packet sizes.

Acknowledgment

This work is supported by Guangdong Province Key Area R&D Program under grant No. 2018B010113001, National Natural Science Foundation of China under grant No. 61972189, the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989, and the Research Grants Council of Hong Kong (11209520) and CUHK (4055138, 4937007, 4937008, 5501329, 5501517).

REFERENCES

[1] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *USENIX NSDI*, 2015, pp. 455–468.

[2] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized" zero-queue" datacenter network," in *ACM SIGCOMM*, 2014, pp. 307–318.

[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti *et al.*, "pFabric: minimal near-optimal datacenter transport," in *ACM SIGCOMM*, 2013, pp. 435–446.

[4] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *COMMUN ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye *et al.*, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010, pp. 63–74.

[6] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *USENIX NSDI*, 2015, pp. 395–408.

[7] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *ACM SIGCOMM*, 2018, pp. 221–235.

[8] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *ACM SIGCOMM*, 2017, pp. 239–252.

[9] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *ACM SIGCOMM*, 2015, pp. 1–12.

[10] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE TPDS*, vol. 12, no. 10, pp. 1094–1104, 2001.

[11] Y. Lu, Q. Xie, G. Kliot, A. Geller *et al.*, "Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, 2011.

[12] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *USENIX NSDI*, 2017, pp. 407–420.

[13] C. Guo, H. Wu, Z. Deng, G. Soni *et al.*, "RDMA over Commodity Ethernet at Scale," in *ACM SIGCOMM*, 2016, p. 202–215.

[14] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mtcp: a highly scalable user-level TCP stack for multicore systems," in *USENIX NSDI*, 2014, pp. 489–502.

[15] J. Postel, "Internet protocol—DARPA internet program protocol specification, rfc 791," *The Internet Protocol*, 1981.

[16] D. Murray, T. Koziniec, K. Lee, and M. Dixon, "Large MTUs and internet performance," in *IEEE HPSR*, 2012, pp. 82–87.

[17] E. Alliance and B. Kohl, "Ethernet jumbo frames," 2009.

[18] S. Narayan and P. R. Lutui, "Network Performance Evaluation of Jumbo Frames on a Network," in *IEEE ICETET*, 2013, pp. 69–72.

[19] A. Das and S. Debbarma, "Performance of Jumbo Sized Data on Jumbo Frame and Ethernet Frame Using UDP over IPv4/IPv6," in *IEEE ADCONS*, 2013, pp. 204–207.

[20] K. Sharma and V. Badarla, "Curtailing latency in data center network by adopting Jumbo Frames," in *IEEE ANTS*, 2016, pp. 1–6.

[21] L. E. Schrage and L. W. Miller, "The queue M/G/1 with the shortest remaining processing time discipline," *Operations Research*, vol. 14, no. 4, pp. 670–684, 1966.

[22] S. Aleksić, "Analysis of Power Consumption in Future High-Capacity Network Nodes," *J. Opt. Commun. Netw.*, vol. 1, no. 3, pp. 245–258, 2009.

[23] J. Chabarek, J. Sommers, P. Barford, C. Estan *et al.*, "Power awareness in network design and routing," in *IEEE INFOCOM*, 2008, pp. 457–465.

[24] H. Hlavacs, G. Da Costa, and J.-M. Pierson, "Energy consumption of residential and professional switches," in *IEEE CSE*, 2009, pp. 240–246.

[25] W. Van Heddeghem, F. Idzikowski, E. Le Rouzic, J. Y. Mazeas *et al.*, "Evaluation of power rating of core network equipment in practical deployments," in *IEEE GreenCom*, 2012, pp. 126–132.

[26] S. Narayan and P. R. Lutui, "Impact on network performance of jumbo-frames on IPv4/IPv6 network infrastructure: An empirical test-bed analysis," in *IEEE IMSAA*, 2010, pp. 1–4.

[27] M. Allman and V. Paxson, "E. Blanton," TCP Congestion Control," RFC 5681, September, Tech. Rep., 2009.

[28] M. H. Alizai, O. Landsiedel, J. A. B. Link, S. Götz, and K. Wehrle, "Bursty traffic over bursty links," in *ACM SenSys*, 2009, pp. 71–84.

[29] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[30] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," in *ACM SIGCOMM*, 1997, pp. 67–82.

[31] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula *et al.*, "VL2: A scalable and flexible data center network," in *ACM SIGCOMM*, 2009, pp. 51–62.

[32] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM*, 2015, pp. 123–137.

[33] A. Saeed, V. Gupta, P. Goyal, M. Sharif *et al.*, "Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates," in *ACM SIGCOMM*, 2020, pp. 735–749.