Inferring In-Network Queue Management from End Hosts in Real-Time Communications

Yaning Guo[1], Zili Meng[3], Bo Wang[1,2], Mingwei Xu[1,2]
[1]Tsinghua University [2]Zhongguancun Laboratory [3]Hong Kong University of Science and Technology

***Abstract*—Active queue management (AQM) algorithms, widely deployed in the internet, are designed to signal end hosts with network conditions in the format of packet losses. However, real-time communication (RTC) applications adopt delay-sensitive congestion control algorithms (CCAs), which are no longer responsive to losses or explicit notifications from AQMs. Moreover, packet losses introduced by different AQMs will further degrade the performance of RTC applications due to unexpected and unnecessary loss recovery. We are therefore motivated to understand the behaviors of AQMs and take necessary countermeasures for RTC applications proactively. For example, with the help of AQM inference, RTC applications will benefit by using loss recovery mechanisms that adapt to various kinds of AQMs to deal with packet losses. However, it is challenging to infer the AQM from end hosts since numerous AQMs have different configurations after decades of evolution. We analyze the temporal behaviors of loss series, extract the inherent invariant features of different AQMs, and categorize them into three types. Our simulation shows that AQM inference can classify AQMs with an accuracy of 96%. We also evaluate a use case on using the AQM inference to improve the loss recovery mechanism (forward error correction, FEC). Our FEC method based on AQM inference improves the recovery rate by at least 56%, and finally reduces the end-to-end delay by 13%.**

***Index Terms*—Active queue management (AQM), Congestion control algorithms (CCAs), Forward error correction (FEC)**

I. Introduction

In recent years, various RTC applications, such as video conferencing and cloud gaming, have become rather prevalent in people's daily life. Users can enjoy abundant real-time contents and the convenience of remote cooperation anywhere through RTC applications. RTC applications take low latency as a prominent advantage so latency optimization for these applications is of great significance and also receives widespread concern from both industry and academia. For example, in order to reduce the end-to-end delay, researchers introduce advanced CCAs [1–3].

However, a key observation in our paper is that the evolution of those recently advanced latency optimizations on the end hosts may not work well with mechanisms on in-network devices. Specifically, AQM algorithms, which have been widely deployed and are designed to control the bottleneck queue length as well as signal the end hosts with network conditions, might introduce side effects when they meet with the latency optimization mechanisms of RTC applications such as CCAs. Most AQMs notify the end hosts in the format of packet losses, while advanced CCAs adopted by RTC applications are no longer responsive to losses. Such a mismatch can actually further degrade the performance of RTC applications. AQM intentionally drops packets as feedback for end hosts, but the video streams in RTC applications cannot tolerate such packet losses – they will retransmit the lost packets to ensure the image quality. In this case, the end-to-end delay will be degraded. Instead of improving performance, AQMs introduce undesirable consequences.

In response, we are motivated to infer the AQM from end hosts. Our insight is that if end hosts are aware of the behavior of AQMs in the network, they can proactively mitigate the potential drawbacks of unexpected packet losses brought by AQMs and further improve end-host latency optimizations. End hosts use loss recovery mechanisms such as FEC methods to proactively compensate the packet losses with redundant packets. The number of redundant packets is chosen based on the future packet losses which are controlled by AQMs. Therefore, AQM inference can help end hosts achieve a more precise loss prediction and use the prediction to implement FEC methods better. For the essential issue of incompatibility between CCAs of RTC applications and AQMs, CCAs can adaptively modify according to the characteristics of AQMs. In this way, CCAs can achieve a rapid identification of various congestion signals varying between different AQMs and make appropriate adjustments of the sending rate. This fundamentally suppresses queuing and further packet losses.

However, it is challenging to know the exact AQM of the bottleneck queue. Numerous AQMs have been proposed in the last several decades. Moreover, diverse end-to-end traffic and AQM parameters can also interfere with our inference, as they can make the loss rate and loss trends indistinguishable over a period of time. When the servers notice a single packet loss (or losses in a short time interval), it is difficult to confidently categorize the AQMs. To address these challenges, we analyze the loss behaviors of commonly used AQMs and categorize them into three types. These three types can represent common kinds of packet loss trends and patterns. Each type has its particular features and typical loss behaviors. Instead of using single-point data that contains little information, we splice the single-point data into time series and infer through these time series that can reflect some characteristics. To eliminate the interference of external factors, we capture inherent features of AQMs for inference because of their invariance and determine the type by judging whether the time series exhibits unique features of a certain type of AQM.

Our evaluation shows our AQM inference mechanism is effective and significantly improved the performance of the FEC method. The inference accuracy is higher than 96% and the relative error of loss prediction after AQM inference is less than 20%, which means that we can accurately obtain the type of AQM and the loss model. For FEC methods, our method achieves an average recovery rate of exceeding 85%, which is 56% higher than the baseline with a similar redundancy rate. Such an AQM inference can also help to improve the end-to-end delay by 13% compared to state-of-the-art mechanisms without knowledge of AQM inference.

II. Background and Motivation

A. *Mismatch between CCAs of RTC applications and AQMs*

RTC applications commonly utilize delay-sensitive CCAs. They apply different transport protocols thus different congestion feedback methods. For the TCP/QUIC protocol, Copa [2] and BBR [1] use ACK packets to run delay estimation and bandwidth measurement respectively to change their congestion window. GCC [3] and NADA [4] used with UDP determine the network states according to the delay, bandwidth, packet losses and other information provided in the feedback message sent by the clients. These algorithms have similar purposes: strictly control queuing delay and react fast enough to perceive the rise of delay.

However, some delay-sensitive CCAs are insensitive to packet losses. Copa has no specific design for packet losses and BBR does not react to the loss rate below 0.2. GCC has the logic for packet losses though, it's much simpler and less flexible compared to the delay-based logic and will only decrease the sending rate after the loss rate is greater than 0.1.

In contrast to these CCAs, in-network AQMs treat the packet loss as the congestion signal and also use it to reduce queuing. Some AQMs have been proposed in recent years and use different loss mechanisms. The packet loss probability of RED [5] and ARED [6] is positively correlated with the queue length. CoDel [7] aims to maintain the time when the queuing delay exceeds the target shorter than the threshold time. PIE [8] also focuses on queuing delay and designs a proportional-integral controller to update the loss probability.

This mismatch between CCAs and AQMs coupled with the lack of cooperation due to different deployment locations can bring adverse consequences. When the queue starts to pile up, some AQMs immediately drop packets to inform end hosts. However, the initial small amount of packet losses are usually ignored by the above delay-sensitive CCAs so end hosts will not change their sending rate until the queue keeps growing to a level that the delay-based congestion control is triggered. This is not a fatal drawback when the network is stable, but the available bandwidth of bottleneck can be suddenly reduced by 10× [9] while CCAs cannot respond in time in such a deteriorated network condition, the queue will build up rapidly because of the extreme disparity between the sending rate and the available bandwidth. As a result, AQMs will eventually create numerous packet losses and the loss rate will increase dramatically in a short period of time. Even after CCAs adjust their sending rate, the queue still takes some time to be drained, and packet losses will also continue to occur during this period. Retransmission caused by packet losses can greatly increase the delay and lead to blurry frames or jitters.

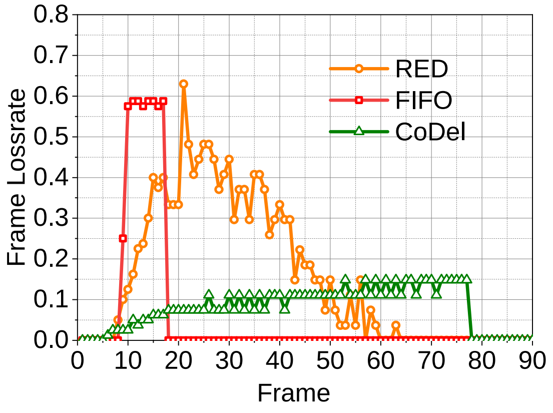


Fig. 1: Different loss trends influenced by AQMs

This phenomenon inspires us to make CCAs and AQMs cooperate with each other to compensate for the negative effects. Because AQMs are implemented in the network, the realistic approach should be carried out on the end hosts, that is, the design of end-host mechanisms is supposed to become AQM-aware and fully consider the influence of AQMs.

B. *Significance of AQM inference*

We describe the important role of AQM inference in two aspects in this subsection. On one hand, end hosts choose FEC methods to combat packet losses, we can use AQM inference results for FEC methods. On the other hand, through AQM inference, we can also adjust the design mechanism of CCAs to mitigate the aforementioned mismatch.

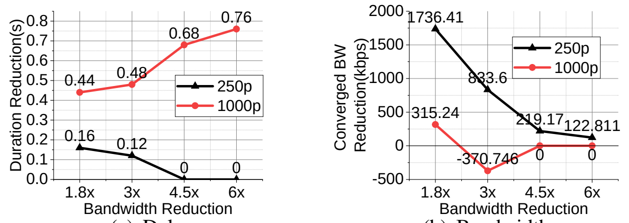
1) ***FEC:*** The performance of FEC methods depends on the relationship between the proportion of redundant packets and the real loss rate. If this proportion is higher than the real loss rate, end hosts have to waste extra bandwidth and increase operation cost while if it's smaller, retransmission is still required after one round of transmission which increases the latency. This indicates that advanced FEC methods require accurate loss prediction.

Meanwhile, packet losses which are controlled by AQMs, cannot be formalized with a single regularity. There are many AQMs in practice and the loss logics have their own particularity. Different AQMs can calculate completely different loss rate for the same queue state, not to mention some AQMs focus on the queue's historical states, even if queues experience the same congestion process, the overall loss trends may display dissimilar ways. Fig. 1 shows that under the same network condition and end-to-end traffic, different AQMs can create multiple loss trends.

The overall trend of packet losses is closely related to AQMs but none of current FEC methods consider AQMs in their prediction, obviously this will reduce their accuracy and generality. FEC methods can be divided into two types. Some methods [10] [11] employ simple time series models which just relate the real-time loss rate to the previous network state. When the loss rate increases due to congestion, they may still

AQM\CCA	Delay-based	Loss-based
FIFO(1000p)	2.8s ✓	5.84s
CoDel	2.32s	1.72s ✓

TABLE I: Delay duration under different combinations of AQMs and CCAs. CCAs here are delay-based logic and the modified loss-based logic of GCC that begins to reduce the sending rate as soon as end hosts are conscious of packet losses. In order to control variables, these two CCAs reduce the sending rate at the same speed after noticing the network congestion. Delay Duration is the time when the end-to-end delay is larger than 200ms.



(a) Delay (b) Bandwidth

Fig. 2: Delay and throughput performance under different CCA and AQM parameters. k in kx of the x-axis represents the bandwidth reduction ratio.(a)The time by which delay duration is shortened after using more sensitive parameters. Delay duration is the time when end-to-end delay is larger than 200ms. (b)The amount by which the converged bandwidth is reduced using more sensitive parameters.

provide a low loss rate while after the network becomes stable again, they may carry on radical prediction. Other schemes [12] [13] [14] use machine learning to predict the loss rate. The limitation is that packet losses can display various evolutionary trends but the model can't learn all of them. Moreover, packet losses under different bottleneck queues dissatisfy a single model therefore such model training is not targeted.

Besides, The specific loss pattern determined by AQM also influences the selection of FEC parameters. If the packet is lost based on the probability, the redundancy rate should be appropriately higher to ensure the successful arrival of the frame. However, if packets are lost at a certain time interval, the number of redundant packets can be a little conservative. It is necessary to focus on the implementation differences brought by different AQMs in the FEC method.

*2) **CCA**:* At present, CCAs on the end hosts independently judge the network condition based on the client's feedback without cooperation with in-network AQM mechanisms, which leads to the neglect and inappropriate processing of congestion signals. For example, [15] concludes that the synergy effect of delay-sensitive CCAs and some AQMs is not satisfying. We compare the duration of delay increase with different combinations of AQMs and CCAs in the face of congestion in TABLE I. For CoDel, compared with delay signals exceeding the thresholds set by CCAs, loss signals are sent to the end hosts earlier so loss-based CCAs can respond faster to the congestion and get better latency performance. In contrast, the duration is shorter for FIFO combined with delay-based CCAs because packets are not dropped until the queue is sufficiently inflated. For different types of AQMs, the optimal choice of CCAs is different.

The parameters of CCAs should also be adjusted adaptively with the parameters of AQMs. Fig. 2 shows the performance presented by FIFO of different queue sizes combined with different sensitivity parameters. The results show that for FIFO of small queue size, more sensitive parameters have either negligible or little help in shortening the duration of delay increase but can drastically decrease the converged bandwidth. On the contrary, more sensitive parameters are necessary for FIFO of large queue size because all reduced packets for transmission can play a role in preventing queue accumulation and the reduction of convergence bandwidth is relatively small or even negative because the overall congestion time becomes significantly shorter. This indicates that the internal parameters of CCAs can also be adjusted in combination with in-network AQMs to achieve a better trade-off of different network indicators.

III. Challenge

Chanllenge & Solution 1: There are many types of AQMs in practice. We carry out a generalized classification for common AQM types based on their packet loss behaviors. We find the most suitable type for each loss case and then establish the loss model within the type.

Chanllenge & Solution 2: We have to infer the type of AQMs under the disturbance of multiple factors and two types of AQMs may also display similar loss evolution in a certain period of time. AQMs are designed with a complete self-consistent logic for queue accumulation and mitigation, which is manifested with coherent loss behaviors. They can be summarized into some expressible characteristics. These characteristics are independent of other factors and faithfully belong to a certain type of AQM. The purpose of our method is to capture invariant loss characteristics for inference. Meanwhile, characteristics are difficult to be reflected by a few points of data in a short period of time so we extract useful features from the time series consisting of all queue state and loss information in a loss event.

IV. Design

The overall design framework is as Fig. 3, We splice the single-point data containing queue and loss information from the previous loss event into a time series and extract features from the series to infer the AQM type of the bottleneck queue and establish the corresponding loss model. Generally speaking, the bottleneck in the network does not change for a period of time, neither does the regularity of packet losses at the bottleneck. Therefore, when the queue starts piling up again and end hosts perceive the congestion after one RTT, end hosts can predict the future loss rate for FEC methods based on the previously obtained loss model. After the current loss event ends, the statistics about packet losses are processed again and used as the basis for the next loss event.

A. Queue and Loss indicators

To extract features, end hosts first need to estimate the single-point data of the bottleneck queue and loss information.

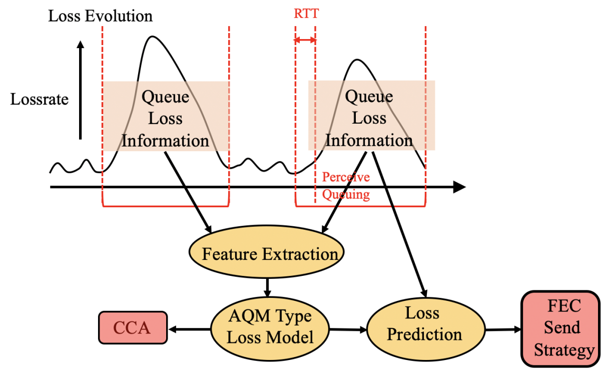


Fig. 3: Overall Framework

End hosts use the feedback of per-packet arrival time and loss information from clients to estimate the queue state. End hosts estimate each packet's queuing delay `QDelay` by Eq. 1, `minOwd` refers to the minimum one-way delay in the past period of time.

$$QDelay = ArrTime - SendTime - minOwd \quad (1)$$

Besides `QDelay`, `QueueLength` is also required for feature extraction. Although end hosts are aware of the traffic of flows sent by themselves, they also need to estimate the traffic of competing flows. End hosts continuously calculate the instantaneous rate at which clients receive packets (`ReceiveRate`) based on per-packet arrival time and add it to a `Rate` list. For each packet, end hosts estimate the `ReceiveRate` for a short time window around the packet's arrival time and update `Rate` list. This `Rate` list retains the values of `ReceiveRate` for a relatively long period of time before and after this packet arrives at the client, then the max(`ReceiveRate`) on the `Rate` list is considered to be the `DequeueRate` during the period the packet is about to leave the queue. End hosts can obtain the traffic of the competing flow based on the difference between `DequeueRate` and `ReceiveRate` computed for this packet. Combining the competing traffic with their own traffic can provide an estimate of the overall traffic and `QueueLength`.

We also define the indicator `DropSpeed` and `lossrate` respectively representing the number of packets lost per unit of time and the percentage of lost packets in all packets. These two indicators are calculated for a group of packets (frames) that are sent as a burst [16]. We obtain a series of single-point data of the queue and loss information. They are then connected in the order of the sending time to form a time series for the following AQM inference.

B. Feature extraction and AQM inference

Features are displayed for three aspects: *Trend*, *End* and *Pattern*. Our method accomplishes a **packet-level** feature extraction to better understand the loss mechanism through fine-grained analysis. Following is a detailed description of the features of these three aspects.

- **Trend:** It records whether the indicators such as `lossrate` and `DropSpeed` show generally rising or falling trends or just fluctuate in a small range within a given period of time. It also refers to when the queue state (`QDelay` and `QueueLength`) has a certain trend the corresponding trend of loss information (`lossrate` and `DropSpeed`).
- **End:** Comparison of the value of indicators at the end of the queuing or the loss event to the peak value during the entire period. It also records the trend of these indicators near the end.
- **Pattern:** ①The degree of tail packet losses within a group. ②The degree of continuous packet losses. ③Time interval characteristics of packet losses.

Pattern	In groups with packet losses, packet losses basically occur at the tail of the group and the corresponding `QueueLength` fluctuates within a small range.
End	The `QueueLength` at the end of the loss event still has a certain size, may even be nearly close to the maximum value for the whole period and gradually decreases to a balanced state.

TABLE II: Spike Features

Trend	When the `QDelay` begins to gradually decrease, the `DropSpeed` continues to rise (the rising speed may be very small and appears little changed) until the end of loss (when `QDelay` still has certain value).
Pattern	In groups with relatively low `lossrate`,continuous losses occur less frequently. Packets are lost with a function of the packet loss time interval and this interval is stable during the entire period.

TABLE III: Cliff Features

AQMs are classified into three types: Spike, Cliff and Hill. We then extract features and infer the AQM type. Through the degree to which the behaviors of time sequences match the features of a type of AQM, we determine whether the bottleneck queue belongs to that type of AQM.

Spike: This type of AQM has a very simple loss logic. If the number of currently queued packets exceeds the queue size limitation, the new packet will be dropped, otherwise, it can enter the queue. Inference for this type is based on the features of *Pattern* and *End* as in TABLE II.

Cliff: This type of AQMs is different from the ones that generally give the loss probability. When the queue exceeds the threshold, it gradually reduces the time interval between each packet loss to alleviate queue accumulation. The result of this design is that even if the queue starts to become shorter, the speed of packet losses still gradually increases until the queue reaches the threshold requirement. Inference for this type is based on *Trend* and *Pattern* as in TABLE III.

Hill: The `lossrate` either corresponds to the real-time queue state or the cumulative state of the queue evolution (`CumulQ`). Inference for this type focuses on *Trend* to establish a relationship between `lossrate` and `CumulQ`.

We use the EWMA method to calculate the `CumulQ` through Eq. 2. `qw` indicates how much the change of `lossrate` lags behind the change of `QueueLength`. Different candidates of `qw` should make the `CumulQ` show clear

Spike	Cliff	Hill	Total
96.25%	96.0%	97.48%	96.89%

TABLE IV: Inference accuracy of different types of AQMs

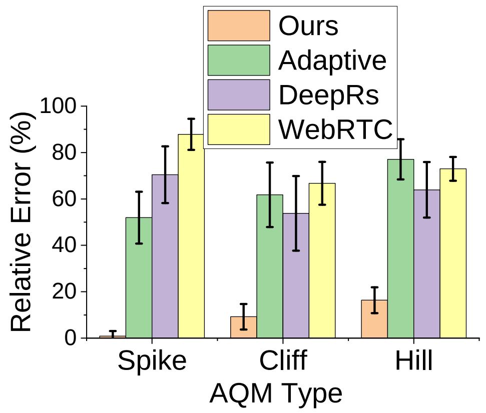


Fig. 4: Loss prediction

differences and the candidates' collection can cover various levels of lagging degrees.

$$CumulQ = (1 - qw) * CumulQ + qw * QueueLength \quad (2)$$

We estimate the `CumulQ` and realize Linear fit for `lossrate` and `CumulQ`. We define the `CorresDegree` to choose `qw`. `CorresDegree` is computed according to the difference of coefficients from the linear fit for the period when `lossrate` increases and the entire period, which represents the degree of conformity between `qw` and the actual loss performance. The fitting error of linear fitting should also be considered. The most appropriate `qw` is selected following the highest `CorresDegree`, together with the coefficients of the corresponding fitting result to establish the loss model.

If there is no obvious correlation between the `lossrate` and `CumulQ`, and considering that to alleviate queuing, AQMs continuously increase the loss rate when there exists queuing, the `lossrate` may add a value at a regular time interval, the value is the function of `QueueState`. We can calculate the corresponding coefficients according to the total change of the `lossrate` over a period of time and the value of the `QueueState` at each update within this period.

C. Loss Model and loss prediction

During the process of AQM inference, end hosts establish the loss model according to the loss mechanism of each type and calculate key parameters. For the new loss event, end hosts estimate the `DequeueRate` as described in IV-A to predict the future queue state. The loss rate can then be expressed: $lossrate = f(queuestate)$, f represents the loss model. End hosts fine-tune the result of the loss prediction based on the type of AQM to obtain the proportion of injected redundant packets.

V. Evaluation

A. Experimental Setup

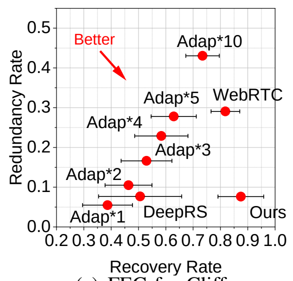
Traces: We evaluate our method with NS-3 simulation and establish a link with 50ms RTT and 30Mbps initial bandwidth of the bottleneck queue which is deployed with different AQMs. End hosts initially send packets at a sending rate slightly less than 30Mbps and they use two time intervals (5ms and 40ms) of sending a group of packets (frame) as a burst. Then we use two ways to change the available bandwidth. The first way is that the available bandwidth is reduced from 2x to 10x. End hosts then take different response time to adjust the sending rate to the new bandwidth. The second way is that the available bandwidth changes every 200ms, and the bandwidth distribution satisfies a Gaussian distribution with mean values of 10Mbps, 15Mbps, 20Mbps and 25Mbps, and a standard deviation of 5Mbps. We totally generate 418 traces including 9 AQMs of different types or parameters. We leave real-world experiments as our future work.

Metrics: We evaluate the accuracy of AQM inference for all traces. We also combine traces (correctly infer the type) belonging to the same AQM, the former trace establishes the loss model for the latter trace. We examine the effectiveness of the obtained loss model based on relative error measurement of loss prediction for the latter trace. We further evaluate the performance of the FEC method for traces of 6 different AQMs. These AQMs are from the second and third types because they are suitable for the FEC method (The redundant packets are not more than data packets). We measure the recovery rate, redundancy rate and also application-layer metrics, the end-to-end delay. If the number of injected redundant packets is greater than or equal to the actual number of lost packets, this frame (a group of packets sent in a burst) can be recovered. The redundancy rate is the ratio of the number of redundant packets to that of data packets.

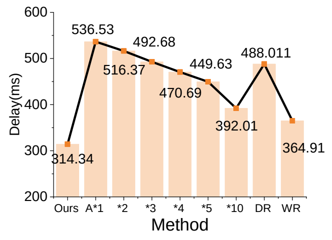
Baselines: We use Adaptive-FEC and Learning-FEC as baselines. For adaptive-FEC, we take the lossrate for the latest one second to multiply a coefficient as the prediction result of future lossrate, the coefficients vary from 1 to 10 and they are labeled Adap*x and A*x in Fig 4,5, where x is the coefficient. We also compare the FEC method in WebRTC [17], which uses a lookup table to map past lossrate to a certain redundancy rate. In Fig 4,5, it's labeled WebRTC or WR. Learning-FEC uses the the LSTM model in DeepRs [13] to predict, training traces are generated with AQMs of different types and parameters. It's labeled DeepRS or DR in Fig 4,5.

B. AQM inference and loss prediction

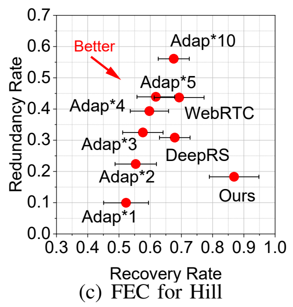
AQM inference accuracy is shown in TABLE IV, three types of AQM can achieve an average accuracy of 96.89% and the inference accuracy of each type of AQM is more than 95% which demonstrates the effectiveness of our inference method. Fig. 4 shows the mean and standard deviation of the relative error of the loss prediction and the average relative error of our method is less than 20% under the three types of AQMs, which is several times lower than the adaptive and the learning-based prediction. The small standard deviation of the prediction accuracy indicates that our method is less affected by the varied network environment.



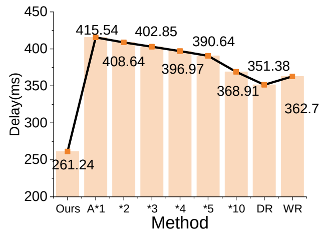
(a) FEC for Cliff



(b) Delay for Cliff



(c) FEC for Hill



(d) Delay for Hill

Fig. 5: Performance of FEC method

C. FEC method

In Fig. 5, We show the mean and standard deviation of the recovery rate as well as the mean of the redundancy rate of FEC methods for the second and third types of AQMs. Our method performs better than all baselines. Our method can achieve the average recovery rate larger than 85% which is greater than other methods and maintain advantages under various network conditions. Moreover, our method does not sacrifice the redundancy rate for the recovery rate, we can achieve a $56.9\% \sim 72.76\%$ increase in the recovery rate with a similar redundancy rate as other baselines. This is because our method can dynamically predict packet losses according to the characteristics of each type of AQM instead of aggressively adding redundant packets. Our method accomplishes the balance between recovery rate and redundancy rate, and performs better in these two metrics. Because our method avoids retransmission, it also has a significant improvement in end-to-end latency performance, which can achieve a reduction of $13.86\% \sim 25.65\%$ compared to the baseline with the smallest delay.

VI. Discussion

A. Implementation

Our method is implemented on the end hosts without the support of in-network devices. All required information used for inference can be obtained directly from the transmission protocol. This design can be practically deployed on the end hosts as an independent module and we anticipate that even if the applications and CCAs used by end hosts change, the execution of the module of AQM inference will still unlikely be affected. AQM inference is only executed once after each loss event, and will not frequently consume computing resources.

B. The type of packet Losses

Packet losses discussed in this paper are because of the queuing of packets at the bottleneck and in-network devices drop the packets according to the loss mechanisms of AQMs. In fact, there are still other types of losses such as packet losses due to low signal to interference plus noise ratio (SINR) of wireless channels but this kind of packet losses can be distinguished from packet losses caused by AQMs. Because this kind of wireless losses is not accompanied by an increasing latency while our AQM inference is executed only when end hosts perceive packet losses with increasing latency.

VII. Related Work

The latency of RTC applications has been intensively investigated for decades. Among them, delay-sensitive CCAs and AQMs have been introduced in II. As for mismatches between these two measures, a variety of solutions have also been proposed to solve the problem.

Forward Error Correction. Except for our AQM-aware FEC method, there are numerous other FEC methods. No matter whether use traditional heuristic algorithms [10, 11, 18] or machine learning algorithms [12–14], the decision inputs of these algorithms are similar: they adapt the redundancy rate by inferring the network conditions. However, few of them are aware of the packet losses by the AQMs, especially the patterns of different AQMs. Our solution is capable of protecting packets from the packet losses due to AQMs while achieving a low redundancy rate as well.

End host-network coordination. Meanwhile, there are also some end host-network coordination research efforts in exchanging the in-network conditions with end hosts. Explicit congestion control (ECN) mechanisms enable the end hosts and network devices to cooperate with each other and achieve cross-layer optimization to solve the above problems. For example, in Kickass [19] and ABC [20], routers perceive key indicators such as link capacity, queuing delay, etc., and send the information back to the end hosts in various forms to adjust the congestion window. This kind of method requires the support of network devices and end hosts at the same time. Thus, the actual deployment overhead hinders further implementation. Our method bridges the gap between the end hosts and in-network AQMs without changing network devices.

VIII. Conclusion

We propose a server-side inference method for in-network AQMs. Through AQM inference, end hosts can obtain a more precise estimation of packet loss that occurs in the network, which helps end hosts better implement CCAs and FEC methods. For various latency-sensitive mechanisms in RTC applications, this inference method allows them to better adapt to the loss mechanism in the network, compensate for the adverse consequences brought by the packet losses and maintain stable performance.

IX. Acknowledgements

This work is sponsored by the National Natural Science Foundation of China (No. 62221003 and 62372261). Mingwei Xu is the corresponding author of the paper.

References

[1] N. Cardwell, Y. Cheng, C. S. Gunn *et al.*, "Bbr: congestion-based congestion control," *Communications of the ACM*, pp. 58–66, 2017.

[2] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. USENIX NSDI*, 2018, pp. 329–342.

[3] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Congestion control for web real-time communication," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2629–2642, 2017.

[4] X. Zhu and R. Pan, "Nada: A unified congestion control scheme for low-latency interactive video," in *2013 20th International Packet Video Workshop*. IEEE, 2013, pp. 1–8.

[5] 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.

[6] S. Floyd, R. Gummadi, S. Shenker *et al.*, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," 2001.

[7] T. Hoeiland-Joergensen, P. McKenney, D. Taht *et al.*, "The flow queue codel packet scheduler and active queue management algorithm," Tech. Rep., 2018.

[8] R. Pan, P. Natarajan, C. Piglione *et al.*, "Pie: A lightweight control scheme to address the bufferbloat problem," in *Proc. IEEE HPSR 2013*. IEEE, 2013, pp. 148–155.

[9] Z. Meng, Y. Guo, C. Sun *et al.*, "Achieving consistent low latency for wireless real-time communications with the shortest control loop," in *Proc. ACM SIGCOMM*, 2022, pp. 193–206.

[10] C. Padhye, K. J. Christensen, and W. Moreno, "A new adaptive fec loss control algorithm for voice over ip applications," in *Proc. IEEE IPCCC*, 2000, pp. 307–313.

[11] A. F. Atiya, S. G. Yoo, K. T. Chong, and H. Kim, "Packet loss rate prediction using the sparse basis prediction model," *IEEE transactions on neural networks*, vol. 18, no. 3, pp. 950–954, 2007.

[12] H. Hu, S. Cheng, X. Zhang, and Z. Guo, "Lightfec: Network adaptive fec with a lightweight deep-learning approach," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 3592–3600.

[13] S. Cheng, H. Hu, X. Zhang, and Z. Guo, "Deeprs: Deep-learning based network-adaptive fec for real-time video communications," in *Proc. IEEE ISCAS 2020*. IEEE, 2020, pp. 1–5.

[14] K. Chen, H. Wang, S. Fang *et al.*, "Rl-afec: adaptive forward error correction for real-time video communication based on reinforcement learning," in *Proc. ACM MMSys*, 2022, pp. 96–108.

[15] G. Carlucci, L. De Cicco, and S. Mascolo, "Controlling queuing delays for real-time communication: the interplay of e2e and aqm algorithms," *ACM SIGCOMM CCR*, vol. 46, no. 3, pp. 1–7, 2018.

[16] A. Zhou, H. Zhang, G. Su *et al.*, "Learning to coordinate video codec with transport protocol for mobile video telephony," in *Proc. ACM MobiCom*, 2019, pp. 1–16.

[17] S. Holmer, M. Shemer, and M. Paniconi, "Handling packet loss in webrtc," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 1860–1864.

[18] Z. Meng, X. Kong, J. Chen *et al.*, "Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming," in *Proc. USENIX NSDI*, 2024.

[19] M. Flores, A. Wenzel, and A. Kuzmanovic, "Enabling router-assisted congestion control on the internet," in *Proc. IEEE ICNP*, 2016.

[20] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "Abc: A simple explicit congestion controller for wireless networks," in *Proc. USENIX NSDI*, 2020, pp. 353–372.