Pacing User-Space Applications

Luis Kleinheinz, Marcel Kempf*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: luis.kleinheinz@tum.de, marcel.kempf@tum.de

***Abstract*—Pacing improves network performance by evenly spacing packet transmissions, reducing delays and packet loss caused by bursty traffic. While Linux kernels support pacing, making it applicable for TCP, user-space protocols like QUIC fall short due to their lack of direct access to kernel-level mechanisms. This paper investigates using Time-Sensitive Networking (TSN) queueing disciplines (qdiscs) to provide effective pacing for user-space protocols. The goal is to bring the benefits of pacing to user-space protocols by discussing the technical details and evaluating which TSN qdiscs is best-suited for this task.**

***Index Terms*—pacing, user-space, tsn, qdiscs**

1. Introduction

In today's networking environment, managing traffic flow is important to ensure efficient and reliable data transmission. Congestion control algorithms can help with that by dynamically regulating the rate at which data packages are sent over the network. By doing so, we can maximize our throughput and ensure a fair distribution of network resources, ultimately providing reliable communication by reducing package loss and retransmissions.

However, congestion control algorithms can produce irregular and bursty traffic, where large volumes of data packets are sent in quick succession. This sudden traffic can lead to queueing delays and increased packet loss, weakening the reliability and efficiency that congestion control aims to achieve in the first place [1]. To address this issue, packet pacing has proven to be a suitable solution. This technique smoothens the traffic by evenly spacing packets over time, and with that reducing bursts and their associated negative impacts on network performance.

Modern Linux kernels have integrated packet pacing within TCP [2], using kernel-level controls to ensure a smoother and more predictable traffic flow. This integration has shown massive improvements in managing network congestion and maintaining steady data transmission rates. However, not all network protocols operate within the kernel space. User-space protocols operate outside the kernel, presenting unique challenges for implementing effective pacing.

The Quick UDP Internet Connections protocol, or QUIC for short, is one such user-space protocol [3]. Like TCP, it is a transport layer protocol designed to make internet connections faster and more reliable. However, QUIC offers some benefits over the well-established protocol, like reduced latency or multiplexing. This is accomplished by combining the transport and the cryptographic layers into a single protocol, and supporting multiple lightweight streams that allow for efficient multiplexing. Yet, its user-space operation means it can not benefit from the kernel-level pacing mechanisms available to TCP. Therefore, achieving effective packet pacing for user-space protocols like QUIC is a problem that has to be solved.

One promising solution is to utilize Time-Sensitive Networking (TSN) queueing disciplines, or qdiscs. TSN qdiscs are advanced network scheduling mechanisms that manage packet transmission timing with high precision [4]. As TSN qdiscs can regulate the intervals at which packets are sent, they could be used to implement mechanisms to bring the same pacing benefits to user-space protocols as those used by kernel-space protocols. How the functionalities of the qdiscs can be used for this is explained later on in this paper.

This paper investigates the feasibility of using TSN qdiscs for pacing user-space protocols, focusing on QUIC. We will explore the mechanics of various TSN qdiscs, compare their effectiveness in providing packet pacing, and evaluate their applicability to user-space protocols. Furthermore, we aim to find a solution as to how TSN qdiscs can be used by user-space protocols, ultimately improving the performance and reliability of modern network communications.

2. Introduction to Pacing

Before investigating whether using TSN qdiscs for pacing user-spaced protocols is feasible, this section will give a rough overview over pacing itself. The aim is to understand the concept behind the mechanism.

2.1. Pacing at kernel-level

The concept of pacing was brought up as a solution to the limitations of traditional congestion control algorithms, which often resulted in packet loss and queueing delays due to their bursty nature [1]. Research investigated the benefits of spacing out packet transmissions to achieve a more stable and efficient data flow [5]. This led to the development and implementation of practical pacing mechanisms in mainstream networking protocols.

Pacing operates by controlling the intervals at which packets are transmitted, resulting in smoother traffic bursts and improving overall network performance. The process begins with packet queuing, where incoming packets are

 doi: 10.2313/NET-2024-09-1_13

initially queued in the network stack or within the application layer, awaiting transmission. Pacing mechanisms then use rate-limiting algorithms to calculate the sending rate based on bandwidth, congestion signals, or application requirements. This mechanism brings many advantages, like the reduction of sudden bursts of traffic and, therefore, lower queueing delays and reduced latency. In general, pacing enables the network to work more stable and consistently, leading to better bandwidth use and higher overall throughput [5].

Pacing support for the Linux kernel was officially introduced with the addition of TCP Small Queues (TSQ) in the kernel version 3.6 in 2012 [6] and the Fair Queue (fq) pacing scheduler in version 3.12 in 2013 [7]. TSQ aims to reduce bufferbloat by limiting the amount of data that could be queued in the network stack. The fq scheduler further enhances the pacing capabilities by implementing fair queuing with per-flow pacing. With these integrations of pacing in the kernel, protocols like TCP could provide for more consistent and predictable network performance. Having these benefits is important for any protocol that operates on the network stack today; therefore, using them for user-spaced protocols like QUIC is desirable. The following section will explore why this can not be done easily and what challenges we are facing.

2.2. Pacing at user-space applications

As pacing offers great advantages and solves various problems of congestion control algorithms, pacing user-space applications would be beneficial to profit from all these aspects as well. For this, we have to adapt to the specific challenges posed by protocols such as QUIC, which operate independently of direct kernel control and cannot use the traditional way of pacing implemented in the Linux kernel. QUIC has gained popularity due to its significant improvements over TCP. While TCP operates over IP and relies on a separate TLS (Transport Layer Security) layer for encryption, QUIC combines both transport and security functionalities into a single protocol. This integration reduces the overhead of multiple handshakes when connections are established and with that enhances security by ensuring encryption is applied at the beginning of communication by default. Moreover, QUIC directly incorporates various additional features, making it a more suitable choice in many scenarios than the traditional TCP. One example is connection migration, which allows sessions to seamlessly switch between network interfaces or IP addresses without interrupting ongoing transmissions, making it especially useful in mobile environments. Another feature is built-in packet pacing in QUIC within its application, which provides a consistent data transmission rate. [3]

However, this pacing, of course, differs from our kernel-level pacing, as it only operates within its own protocol stack instead of directly influencing how packets are scheduled. These technical advancements make QUIC well-suited for modern applications, especially for mobile and multi-homed environments.

Despite all these advantages, QUIC still faces some problems when it comes to implementing packet pacing in its user-space environment. Unlike kernel-level protocols like TCP, which benefit from being integrated with the operating system's network stack and hardware, user-space protocols like QUIC operate at a higher layer, relying on application-specific libraries and interfaces. Therefore, using kernel-level optimizations like the pacing scheduler is not possible, and another solution has to be found to mitigate this problem.

Because of that, it is important to research different methods to mimic kernel-level pacing mechanisms within user-space applications. One interesting method we will talk about is TSN qdiscs, which enable user-space applications to use Time-Sensitive Networking.

3. TSN Qdiscs as solution

3.1. Introduction to TSN qdiscs

Time-Sensitive Networking (TSN) is a set of IEEE standards developed to provide reliable real-time communication over networks by introducing advanced traffic management techniques to support applications like industrial automation and live audio-video transmission that demand precise timing and low-latency data transmission. The Linux kernel has implemented different TSN queueing disciplines (qdiscs) to introduce various mechanisms for managing packet transmission in the kernel space. For pacing our user-space applications, we will focus on these qdiscs provided by the Linux kernel as well, as there are far too many different qdiscs to evaluate all of them.

Linux currently supports three different qdiscs related to TSN. [8]

Credit-Based Shaper (CBS): CBS, which is defined in the IEEE 802.1Qav standard [9], assigns credits to different traffic classes and manages when packets can be transmitted based on the accumulated credits. By dynamically adjusting the transmission rate according to available credits, this qdisc can manage the use of available resources and allocate the packets accordingly.

Enhancements for Scheduled Traffic (TAPRIO): TAPRIO implements features introduced by the IEEE 802.1Qbv standard [10]. It introduces precise packet transmission scheduling, so that network administrators can define transmission schedules for different traffic classes, including priority access for important data. This deterministic approach to packet scheduling requires more manual scheduling by the administrators in contrast to the other two alternatives, but has less operational overhead. With multiple priority levels and strict scheduling policies, TAPRIO provides an efficient transmission of time-sensitive data streams in our network.

Earliest TxTime First (ETF): This qdisc is not part of the IEEE TSN standards. However, it is a specialized queueing discipline in the Linux kernel designed to prioritize packet transmission based on transmission time (TxTime) values. The problem is that only certain network interface cards, like the Intel Ethernet Controller I210, support this feature by letting packets be tagged with specific TxTimes that specify when they should be sent. ETF then proceeds to transmit the packets according

 doi: 10.2313/NET-2024-09-1_13

to their TxTime, enabling real-time applications to maintain their synchronization and timing. [11]

All in all, time-sensitive networking and the queueing disciplines are great tools in the Linux system that offer various possibilities to manage packet transmission with precision timing and prioritization. If they are applicable in user-space applications, they can be used to implement a pacing-like mechanism for applications like QUIC. If this is successful, we can mitigate bursty traffic and minimize latency, effectively gaining the same benefits as from pacing itself. Therefore, by evaluating their applicability to user-space environments in the following sections, we can potentially find a way to get all the kernel-level benefits we already have to a user-space-level as well, ultimately supporting a similar mechanism to pacing for protocols like QUIC.

3.2. Evaluating methods of using TSN qdiscs in user-space

Usually, TSN (Time-Sensitive Networking) queuing disciplines (qdiscs) are implemented at the kernel level [12] [13] [11] and interact with the network stack to manage packets. Operating at the kernel level provides qdiscs with the advantage of relatively low overhead and the ability to interact closely with the operating system. However, to use TSN qdiscs with user-space protocols like QUIC, modifications are needed to adapt the kernel-level qdiscs for compatibility with these protocols.

3.2.1. Implementing qdiscs in user-space. Implementing qdiscs directly in the user space to be used by our user space applications allows for greater flexibility and customization. Certain measures can be taken to adapt the implementation to fit the user space and our application. As our implementation is independent of the kernel-level implementation, it is isolated from changes and updates of the kernel implementation. This brings advantages like more control and more stability to our implementation. However, new findings and improved implementation have to be maintained by ourselves. Another downside is that user-space implementations generally introduce more overhead than kernel-level operations, potentially affecting the performance. Also, using kernel-level functionality in user-space can be complex and may be time-consuming to develop. As it is our own implementation, extensive testing and performance checks are necessary. However, as the kernel implementation is rather well-tested and optimized, it is probable that our implementation will not quite match the quality of the one at the kernel-level.

3.2.2. Using the kernel-level implementation. In contrast to that, when trying to rely on the kernel-level implementation, communication between the kernel-level and our user-space application is necessary. A common approach involves creating an interface or API to expose the kernel-level qdiscs to our application. With this method, we can take advantage of the efficiency of kernel processes, providing minimal performance overhead. Kernel-level implementations are generally more robust and stable due to extensive testing and integration with the operating system. Therefore, we benefit from these aspects by directly using the well-tested implementation. Also, developing an API to use these qdiscs in user-space applications is naturally less effort and can be done faster than developing the qdiscs on the user-space itself. Therefore, it is also less prone to development errors and offers a more robust method of using qdiscs in our application. Of course, some overhead will exist due to the communication with the kernel level. However, this is usually quite minor when working with an API on the same machine.

3.2.3. Conclusion of using qdiscs in user-space. Due to the improved quality of the implementation, the reduced development effort, and the potential performance advantages, using the kernel-level implementation by introducing an API or other interface for communicating with the kernel implementation from our user-space application seems like the better method for using TSN qdiscs in user-space applications. As the Linux kernel currently offers three different types of implemented qdiscs, the question of which of these qdiscs is best suited for our task arises. This will be evaluated in the following section.

3.3. Evaluating the best suited qdisc for pacing

As we figured that using the already implemented TSN qdiscs of the Linux kernel seems to be the most appropriate solution for our purpose, we will only evaluate the three qdiscs supported by the kernel. It is definitely notable that many more qdiscs are standardized by IEEE. However, they would need to be manually implemented in the user space. This approach has its downsides, and the other qdiscs do not provide massive advantages compared to the existing implementations. Therefore, we will focus on the existing qdiscs in the Linux kernel and evaluate the best option for our task [8].

3.3.1. Credit-Based Shaper (CBS). As previously mentioned, CBS (Credit-Based Shaper) operates by allocating credits to different traffic classes and dynamically adjusting transmission rates based on the availability of these credits. [9] This dynamic adjustment is also one of the biggest advantages of CBS as a qdisc, as it offers a high flexibility for managing packet transmission. The credit-based mechanism regulates the timing of packet transmissions, which results in smoother bursts of traffic which is what we want to achieve with our pacing emulation. By controlling the rate at which packets are sent, CBS can provide a consistent transmission rate and optimize our network performance. This adjustment is especially beneficial for user-space applications like QUIC that require an adaptive rate, depending on the required usecase.

However, using CBS in user-space applications also has some problems. When implementing a pacing-like solution that uses CBS, we will naturally introduce some overhead that could potentially damage our performance. This overhead comes from the additional computing power needed for managing the credit system and the calculations of the transmission rate based on the remaining credits. This amount of overhead may not be that prevalent with other types of qdiscs, as they do not

have a credit management mechanism.

In conclusion, CBS has a great flexibility for managing transmission rates and controlling packet timing, making it suitable for implementing pacing-like mechanisms in protocols like QUIC. Its dynamic credit-based adjustment ensures smoother traffic flows, which reduces latency spikes and jitter in high-traffic scenarios when compared to the other TSN qdiscs. There is some overhead that comes with using CBS, but that may be worth it for complex systems that can benefit off the flexibility that the credit system offers.

3.3.2. Enhancements for Scheduled Traffic (TAPRIO). Enhancements for Scheduled Traffic, or TAPRIO for short, enables network administrators to define different traffic classes for packages and create transmission schedules for those [10]. TAPRIO also offers multiple priority levels for critical data and, with that, provides consistent and time-sensitive data transmission, which is really beneficial for user-space applications like QUIC due to a low latency and precise timing. This approach of packet scheduling makes this qdisc a relatively precise and efficient option with little overhead, which is another major advantage of this qdisc. Especially compared to CBS, TAPRIO mitigates the heavy computational overhead needed in CBS by not using a credit-based management system. With those mechanisms, this qdisc can effectively emulated pacing in user-spaced protocols.

However, TAPRIO's deterministic nature also introduces some challenges. Manually configuring the schedules appropriately means a high effort from the network administrator side. Especially in more complex applications, this may be prone to more errors. Especially a dynamic user-space application like QUIC requires frequent adjustments to the transmission schedule based on real-time network conditions. Additionally, the synchronization between the application and schedules introduces overhead and also potential security risks if configured poorly. For example, misconfigured TAPRIO schedules could allow attackers to exploit predictable packet transmission times by using timing attacks.

TAPRIO's biggest downside, however, is that the manual configuration might not offer the same level of flexibility as dynamically adjusted qdiscs like CBS. While TAPRIO is best suited in environments where precise and stable scheduling is important, it is not as adaptable to changing network conditions that require real-time adjustments to transmission rates. For user-spaced applications, flexibility should be a priority to offer a suitable solution for most applications.

In conclusion, even though TAPRIO offers robust and precise packet scheduling, its restricting manual configuration is very limiting for our purpose. For supporting a pacing-like mechanism for user-space applications it is important to provide flexibility and minimize the configuration overhead on the administrator side. As TAPRIO does not offer that flexibility and introduces a large configuration overhead for the transmission scheduling, it does not offer a more suitable solution than CBS does, even though it would mitigate the computational overhead provided by CBS.

3.3.3. Earliest TxTime First (ETF). The Earliest TxTime First (ETF) qdisc is quite different from the other two supported qdiscs. First, it is not part of the IEEE TSN standard, as mentioned, making it less standardized. This qdisc is designed to prioritize packet transmission based on a unique value named transmission time, or TxTime. By allowing packets to be tagged with specific TxTimes, ETF can specify the exact moment they should be sent. This feature is particularly supported by certain network interface cards, such as the Intel Ethernet Controller I210, to enable the accurate timing of packet transmissions. [11]

ETF offers some advantages over CBS and TAPRIO; however, they are really situational. This qdisc can ensure that packets are transmitted at precise intervals, maintaining synchronization and, with that, minimizing latency and jitter. While this sounds like a great advantage, it comes with the cost of severe dependency on hardware support. Not all network interface cards support TxTime tagging, making ETF a specialized and niche tool for managing certain applications like high-frequency trading in finances. Even if the hardware aspect was to be overcome, the fixed TxTime still lacks the flexibility we seek in our qdisc, making it rather similar to TAPRIO. All in all, for an all-purpose tool and a solution for pacing in the user-space, ETF does not provide a suitable solution and is probably not worth investigating further.

4. Conclusion

This paper analysed the feasibility of using Time-Sensitive Networking (TSN) queueing disciplines (qdiscs) as a way to emulate effective pacing for user-space protocols like QUIC. Firstly, we concluded that building upon the already implemented qdiscs in the Linux kernel via an interface or API that is yet to be implemented seems like the best approach. Then, after evaluating the three main TSN qdiscs supported by the Linux kernel—Credit-Based Shaper (CBS), Enhancements for Scheduled Traffic (TAPRIO), and Earliest TxTime First (ETF)—we conclude that CBS offers high flexibility with its dynamic credit-based mechanism, which is a desirable feat for implementing pacing for our applications. ETF falls short due to its hardware constraints, and TAPRIO introduces a high configuration and maintenance overhead that should be avoided in the user-space.

Therefore, CBS offers the most promising option for implementing pacing in user-space protocols. However, the integration of any TSN qdisc in user-space applications like QUIC would still require consideration of overhead and compatibility issues. Also, it is notable that there might be other qdiscs described by IEEE that are better suited than the preexistent ones in the Linux kernel. Those would need to be implemented manually, of course, providing a whole range of different challenges but also advantages.

In further research, the feasibility and implementation of an API for communicating between the application and the kernel-level qdisc should be evaluated to fully realize the benefits of pacing in user-space environments.

References

[1] C. Nandhini and G. P. Gupta, "Exploration and evaluation of congestion control algorithms for data center networks," *SN Computer Science*, vol. 4, no. 5, p. 509, 2023. [Online]. Available: https://doi.org/10.1007/s42979-023-02016-4

[2] M. Ghobadi and Y. Ganjali, "Tcp pacing in data center networks," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 2013, pp. 25–32.

[3] E. J. Iyengar and E. M. Thomson, "Quic: A udp-based multiplexed and secure transport," Internet Engineering Task Force, Proposed Standard RFC 9000, May 2021, accessed: 2024-06-16. [Online]. Available: https://www.hjp.at/doc/rfc/rfc9000.html

[4] N. Finn, "Introduction to time-sensitive networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018.

[5] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of tcp pacing," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, 2000, pp. 1157–1165 vol.3.

[6] E. Dumazet, "Tcp small queues," *LWN.net*, July 2012, accessed: 2024-06-16. [Online]. Available: https://lwn.net/Articles/506237/

[7] ——, "Fq: Fair queue traffic policing," https://www.man7.org/linux/man-pages/man8/tc-fq.8.html, iproute2 project, September 2015, accessed: 2024-06-16. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-fq.8.html

[8] "Configuring tsn qdiscs — tsn documentation project for linux* 0.1 documentation," 2018, readthedocs.io. [Online]. Available: https://tsn.readthedocs.io/qdiscs.html

[9] "Ieee standard for local and metropolitan area networks– virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. 1–72, 2010.

[10] "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[11] Man7.org, *tc-etf(8) - Linux manual page*, 2018, accessed: 2024-06-16. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-etf.8.html

[12] ——, *tc-cbs(8) - Linux manual page*, 2018, accessed: 2024-06-16. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-cbs.8.html

[13] ——, *tc-taprio(8) - Linux manual page*, 2017, accessed: 2024-06-16. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-taprio.8.html