Poster: EasyTrans: Enable Fast Iteration of Transport Protocol

Jie Zhang*, Chuan Ma*, Wei Wang†, Kai Zheng†, Yong Cui*
Tsinghua University, Huawei Technologies†*

***Abstract*—The main iteration goal of transport protocols is to optimize the performance of specific modules. In this poster, we propose a framework named EasyTrans, that enables fast iteration of transport protocol modules. With EasyTrans, developers can focus on the modules they want to iterate and no longer need to deal with other unnecessary parts of the transport protocol. Through different module calling modes, EasyTrans enables high performance even if the modules use algorithms that require sophisticated computation such as machine learning. We implement EasyTrans based on QUIC. Evaluation results show that the overhead of EasyTrans is slight.**

***Index Terms*—transport protocol, protocol design, extensibility**

I. Introduction

Traditional Internet transports provide two abstractions: reliable byte stream and best-effort datagram. As the representatives of these two abstractions, TCP and UDP have been the most widely used. However, recently, new protocols are emerging, inspiring by rapid deployments of continuingly evolving application. Among these new protocols, the most successful one is QUIC [1]. QUIC provides rich features and can update at the same frequency as applications, bringing new opportunities for the extensibility and performance optimization of the transport protocol. PQUIC [2], leveraging the features of QUIC, proposes a new extensible model, which enables clients and servers to dynamically exchange plugins that extend the protocol on a per-connection basis. However, PQUIC is plagued by the complexity of use and low performance. For example, leveraging PQUIC to add FEC involves modification from 51 code anchors and causes the achieved goodput reduced by half. Inspired by the following observations, we believe it is necessary and possible to design a framework that enables easier development and maintains good performance.

First, the optimization of performance is the most frequent iteration in the protocol, and it often involves specific modules such as congestion control, stream scheduling. The optimization of the protocol mechanism, such as the optimizing of the connection establishment process, although it is very important, is not frequent.

Second, in a single project, each researcher or developer often only needs to optimize one or several specific modules. However, due to the tightly coupled implementation of the protocols, they usually have to understand much more documents and codes, which is difficult and unnecessary. Some work is devoted to alleviating this pain. For example, CCP [3] decouples congestion control from protocols and allows developers to write algorithms in user-space with consistent and easy-to-use interfaces.

This work was supported in part by the National Key R&D Program of China (No.2018YFB1800300).

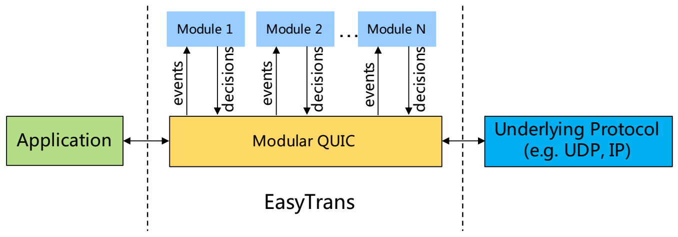

Fig. 1. EasyTrans Architecture.

Third, compared with flexibility, developers are generally more sensitive to performance of the transport protocols. Solutions that affect performance are unlikely to be adopted. Multi-process schemes such as CCP and complex single-thread schemes such as PQUIC often induce considerable performance problems.

In this poster, we propose a framework, *EasyTrans*, which enables modular development of transport protocol while inducing slightly overhead. EasyTrans is less flexible than extensible frameworks like PQUIC—users of EasyTrans cannot define new modules unless they directly modify the system implementation. However, EasTrans is much more easy-to-use because users can focus on the iteration of specific modules without caring about other parts.

II. Design

Figure 1 shows the architecture of the EasyTrans. EasyTrans is composed of *Modular QUIC* and *Modules*. The Modules are several protocol modules that require frequent iteration. They interact with Modular QUIC in runtime. To enable fast iteration of Modules, EasyTrans provides developers with complete interface for implementation and iteration of Modules. For each module, we first determine the events that are highly related to its performance, and then trigger a processing call when these events happen in Modular QUIC. Modules and Modular QUIC can be iterated separately.

All Modules and Modular QUIC runs in an user-space process, thus not induce context switch or inter-process communication. We carefully tuned the calling of modules for

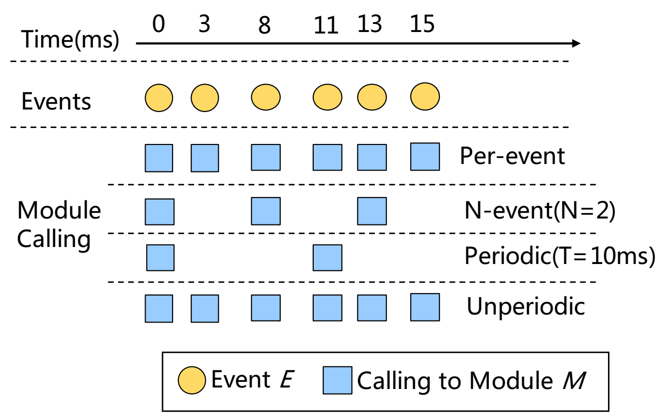

Fig. 2. An example of events processing modes.

performance. EasyTrans provides both blocking modes (*Per-event, N-event*) and non-blocking modes (*Periodic, Unperiodic*) to process the events. The Modules with a small amount of computation should use blocking modes thus run in the same thread of Modular QUIC. The Modules that require a lot of computation should use non-blocking modes to open their own threads, and benefiting from the multi-core. EasyTrans provide a *ChangeMode* method before every event process trigger. Developers can use this method to custom the call mode.

Figure 2 shows an example of these modes. During this period of time, there are 6 events E occur that need to be processed by module M, which occurred at the 0th ms, 3ms, 8ms, 11ms, 13ms, 15ms. M does not require much computation and can be completed immediately after every call. Examples of the four modes are as follows. (1) Per-event mode: Every time an event E occurs, M will be called immediately for processing. (2) N-event mode: When an event E occurs for the first time, M is called immediately. Each subsequent time an event E occurs, the Modular QUIC checks the number of events accumulated since the last call, and calls M for every Nth event. (3) Periodic mode: When E occurs for the first time, M is called immediately. Each subsequent time an E occurs, the Modular QUIC call the M if the time interval since the last call has exceed T. (4) Unperiodic mode: Every time an event E occurs, Modular QUIC will call the M while there is no running M. Unperiodic and Per-event are similar in this example because the M can be completed quickly enough.

III. Implementation and Preliminary Results

We develop a prototype of EasyTrans based on a RUST implementation [4] of QUIC. The prototype consists of the Modular QUIC and two Modules (congestion control Module and stream scheduling Module). We set the congestion control Module as Unperiodic mode defaultly. Evaluation shows it works well both with the ACK-clocked algorithms or learning-based algorithms. We set the stream scheduling Module as Per-event mode to enable scheduling before each packet sent.

We evaluate the prototype using Linux 5.11.0 on a machine with four 3.4 Ghz cores and 8 GB memory. Table I shows

TABLE I
Achieved loopback goodput of different Calling Modes

Modules	Calling Modes	Goodput(Gbps)
No module	-	1.895
Congestion control (a) and stream scheduling (b)	a: Unperiodic b: Per-event	1.799
a and b	a: Periodic (T=10ms) b: Per-event	1.790
a and b	a: Per-event b: Per-event	1.786

the achieved single-core goodput over a loopback interface. The goodput of EasyTrans reduced about 5% if the modules are invoked. When we changed the calling modes of the congestion control module, EasyTrans achieved almost identical goodput.

An earlier version of the prototype was used in the final stage of a competition to optimize deadline requirements of data delivery. Twenty teams entered the final. We provided them with a docker environment [5] for development and testing. Using the interface we provided, they all implemented their congestion control and scheduling algorithms without reading code of the Modular QUIC. When the algorithms are the same, there is no significant difference in the scores obtained by the implementation using EasyTrans and the implementation directly modified in the integrated system.

IV. Future Work

There are other parts that have significant impact on the performance of the transport protocol, such as ACK [6] and multi-path [7]. We will implement these as Modules to enable fast iteration of transport protocol. We also plan to carry out an extensive evaluation to explore the behavior of EasyTrans.

References

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 183–196.

[2] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing quic," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19, 2019.

[3] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan, "Restructuring endpoint congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, 2018, pp. 30–43.

[4] A. Ghedini. (2021) Savoury implementation of the quic transport protocol and http/3. [Online]. Available: https://github.com/cloudflare/quiche

[5] (2020) docker image of the second aitrans competition. [Online]. Available: https://hub.docker.com/r/aitrans/aitrans2

[6] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "Tack: Improving wireless transport performance by taming acknowledgments," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '20, 2020, pp. 15–30.

[7] Z. Zheng, Y. Ma, Y. Liu, F. Yang, Z. Li, Y. Zhang, J. Zhang, W. Shi, W. Chen, D. Li *et al.*, "Xlink: Qoe-driven multi-path quic transport in large-scale video services," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '21, 2021, pp. 418–432.