**前向纠错编码在新型网络协议QUIC中的应用综述**

林利祥1

（1.复旦大学 信息科学与工程学院，上海 200433）

摘要：TCP（Transmission Control Protocol）是如今使用最广泛的网络协议。由于其实现在系统内核，TCP的更新迭代十分缓慢，也难以满足日新月异的网络需求。于是谷歌公司在2013年提出QUIC（Quick UDP Internet Connections）协议。与TCP相比，QUIC在连接性能以及网络安全等方面都更有优势。为了提高实时传输数据的可靠性，避免丢包导致的超时重传，谷歌曾在QUIC中引入前向纠错编码（Forward Error Correction Coding）技术。大量研究表明，FEC可以在高丢包环境下改善实时数据传输的可靠性。本文首先介绍了QUIC协议的特性以及相关研究；之后介绍了前向纠错编码以及其在TCP、QUIC等协议中的相关应用研究；最后给出了在网络传输中使用前向纠错编码仍存在的挑战。

关键词：前向纠错编码；QUIC协议；TCP协议；数据传输；

**A Survey on the Applications of Forward Error Correction Coding in the New Network Protocol QUIC**

LIN Lixiang1

（1.School of Information Science and Engineering, Fudan University, Shanghai, 200433）

**Abstract:**Nowadays, Transmission Control Protocol (TCP) is the most widely used network protocol. Since TCP is implemented in system kernel, the process of TCP update is stalled, and it is difficult for TCP to meet the ever-changing network requirements. Therefore, Google proposed QUIC(Quick UDP Internet Connections) Protocol in 2013. Compared with TCP, QUIC has more advantages in connection performance and network security. In order to improve the reliability of real-time data transmission and avoid time-out retransmission, Google introduced Forward Error Correction Coding in QUIC. A large number of studies have shown that FEC can improve the reliability of real-time data transmission in the high-loss environment. This paper first introduces QUIC and related study. Then it demonstrates the forward error correction coding and its application in TCP and QUIC. Finally, we present the challenge of using forward error correction coding in network transmission.

**Keywords:** Forward error correction coding；QUIC；TCP；Data transmission

# 

# 引言

近三十年来，虽然TCP（Transmission Control Protocol）成为了互联网中广泛使用的一个协议，但是其更新迭代却一直较为困难。这是主要因为TCP在系统内核中进行部署，每次更新都需要重新部署操作系统。这些涉及到底层代码的大规模迭代本身就非常缓慢；其次，由于TCP已被作为一个长期可靠的协议使用，很多中间件，比如防火墙、NAT网关、整流器等设备的规则都已固化。修改TCP可能会让这些中间件无法准确识别数据包。最后，从编程的角度而言，内核级别的编程相比于应用层编程会更加复杂，更新周期也会更长。

如今网络环境的发展日新月异，面对高吞吐、低延时的用户需求，传统TCP在一些网络场景中的问题也逐渐暴露出来。例如TCP在弱网中建立连接的握手延迟较大、队头拥塞、连接迁移不平滑等等问题。为了解决这种复杂网络环境下的传输性能问题。我们做了两方面的调研：一方面研究了谷歌公司提出的全新的QUIC协议。另一方面针对专门应对复杂网络环境，提升数据传输效率，减小时延的前向纠错算法做了详细的调研。

本文首先介绍了QUIC协议的特性和相关研究。然后介绍了FEC的发展历史和现状，重点讲述了在复杂场景中（高铁，卫星网络，组播等）应用前向纠错编码的动机、建模、以及在QUIC协议中的应用。文章最后提出了QUIC协议结合前向纠错编码面临的种种挑战。

# QUIC协议概述

QUIC（Quick UDP Internet Connections）是谷歌公司在2013年提出的一种基于UDP的全新互联网安全传输协议，该版本也被称为gQUIC。*[索引]*谷歌公司期望它能够成为TLS/TCP的替代品，为大众提供更好的网络服务。相比于传统TCP协议，基于UDP底层协议的QUIC可以在应用层实现低延时传输。这使得QUIC不会被网络中间件认为是不安全的协议，也能更快地在今后部署更新。

*[索引] Roskind J. Experimenting with QUIC[J]. The Chromium Blog, 2013.*

如今，QUIC已被部署在谷歌的Chrome浏览器以及Youtube流媒体服务上。据统计*[索引] [索引]* 数千个部署有QUIC的服务器已经处理了十亿级的网页请求。30%的谷歌流量和7%的全球流量也都使用QUIC协议。得益于其应用层实现，QUIC的版本迭代速度也有着极大提升。*[索引]*这将有助于QUIC以后大规模的部署更新。

*[索引]Zhong Z, Hamchaoui I, Khatoun R, et al. Performance evaluation of cqic and tcp bbr in mobile network[C]//2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). IEEE, 2018: 1-5.*

*【Cited from 南大 2018】*

*[索引][1] J. Ruth, I. Poese, C. Dietzel, and O. Hohlfeld, “A First Look at QUIC ¨in the Wild,” in International Conference on Passive and Active Network*

*Measurement. Springer, 2018, pp. 255–268.*

*[索引]Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. 2018. A First Look at QUIC in the Wild. In International Conference on Passive and Active Network Measurement. Springer, 255–268.*

客观来说，QUIC融合了TCP，TLS与HTTP/2的特性。QUIC协议虽然基于UDP实现，但是它和TCP一样有着拥塞控制，流量控制等功能，并且在握手上实现了TLS的保密功能。与TCP相比，QUIC能使用更少的往返通信时间（Round-Trip-Time，RTT）建立安全的会话。可以说QUIC整合了TCP协议的可靠性以及UDP协议的传输效率。

如今，QUIC协议已经成为了HTTP/3*[索引]*的标准。为了更好地推广QUIC协议，谷歌公司已经于2015年将QUIC的标准化交给了IETF。由IETF标准化的QUIC协议一般被成为iQUIC。谷歌公司自己的应用中依旧使用的是gQUIC版本，但是每次的更新在标准上会参照iQUIC的标准。

*[索引] Bishop M. Hypertext transfer protocol version 3 (HTTP/3)[J]. Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-24, 2019.*

## QUIC协议特性

QUIC协议相比于传统TCP协议有了几个具有代表性的特性，包括0-RTT连接，多路复用，解决重传模糊，连接无缝迁移等特性。

### 1.1.1 0-RTT连接特性

对于TCP+TLS协议而言，建立可靠连接需要3个RTT。TCP为了保证数据包不丢失并且有序到达，需要对收发两端进行同步确认。首先，客户端向服务器发送SYN信号来请求同步。服务器端收到后，发送ACK确认信号和SYN同步信号到客户端。客户端收到信号，再发送ACK向服务器确认。随后还要进行TLS1.3的握手，交换密钥及证书。在对数据即时性要求较高的情况下，TCP这样的握手的成本就会相对较高。

而对于QUIC而言，初次建立连接时，如果客户端没有缓存服务器端的相应配置信息，则需要花费1个RTT来请求相关信息。如果已缓存了服务器端的相关配置信息则第二次连接可以依靠缓存的数据直接生成密钥，并立即进行数据传输。这样便能显著减少连接建立的时间，提高数据的传输效率。

### 1.1.2 多路复用解决队头拥塞

多路复用技术最早在HTTP2*[索引]*协议中被提出，这也是HTTP2最经典的设计之一。多路复用使得多个HTTP请求在一条TCP连接上同时传输。HTTP/2多路复用概念的提出解决了HTTP层面的队头拥塞*[索引1，2，3]*问题。

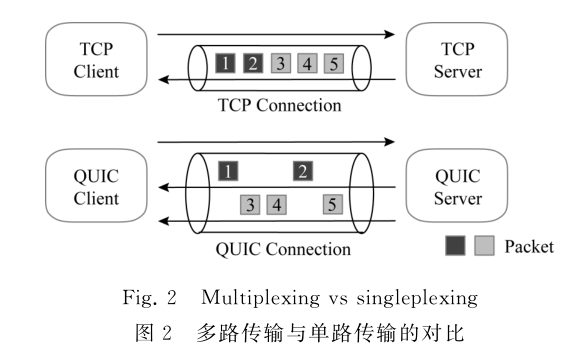
*[索引] Belshe M, Peon R, Thomson M. Hypertext transfer protocol version 2 (http/2)[J]. 2015.*

*[索引1] ScharfM K S. HeadGofGlineblockinginTCPandSCTP: Analysisand measurements[J]. ProcoftheIEEE Globe CommunicationsConf. Piscataway, NJ: IEEE, 2006, 15.*

*[索引2] Qian F, Gopalakrishnan V, Halepovic E, et al. Tm3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking[C]//Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. 2015: 1-13.*

*[索引3] Mi X, Qian F, Wang X. Smig: Stream migration extension for http/2[C]//Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. 2016: 121-128.*

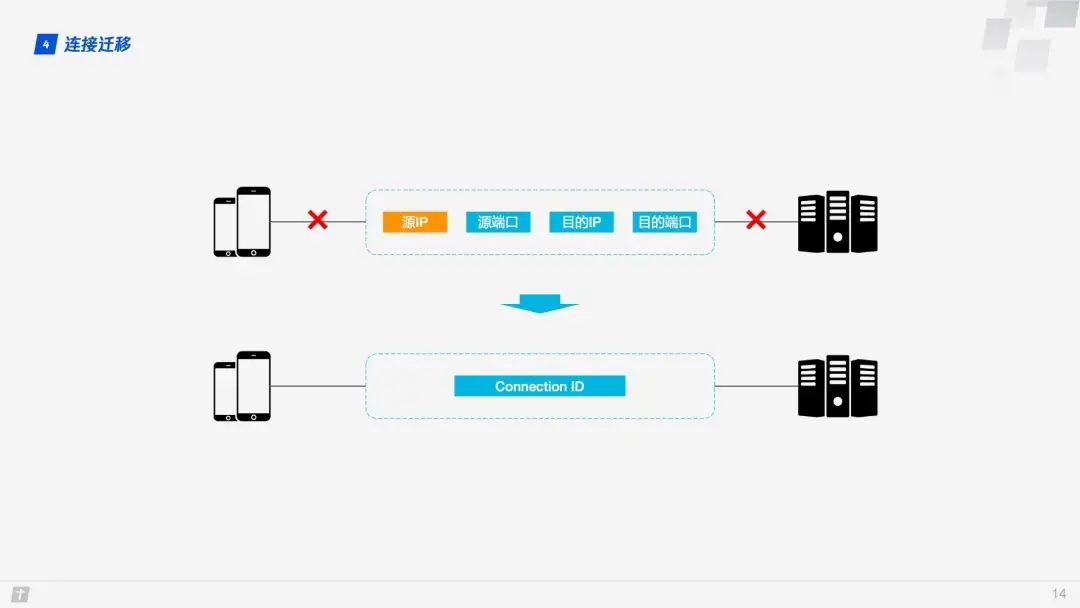
QUIC的设计继承了HTTP/2的优点，引入了流（Stream）的概念。在同一条连接（Connection）上可以并发地建立多个流，每个流相互独立，不存在约束与依赖关系。即使一个流的数据包丢失，后续流的数据包也能被应用层直接读取并处理。在一条连接中，不同的流之间共享连接信息。所有的流共同参与拥塞控制、丢包检测以及数据加密。



1.1.3 解决重传模糊问题

传统TCP的拥塞控制算法在QUIC中都有相应的实现，如TCP默认的拥塞控制算法Cubic也是QUIC的默认算法。另外如Reno、BBR等拥塞控制算法也可以在QUIC中实现。但是QUIC对TCP的重传模糊问题进行了改进。对于TCP而言，每一个数据包都有一个独立的序列号（Sequence Number）。发送端通过观察接收端是否发来Ack消息，来确认该数据包是否成功到达。若响应超时或接收端请求重传数据包，则该数据包的Sequence Number依旧和原来相同。由于网络链路无法保证数据包按顺序到达，接收端也就无法辨别到达的数据包是首次发送还是重传发送。而在QUIC协议中， Packet Number被用来代替TCP中的Sequence Number，但协议保证了每一个Packet Number都严格递增，不会重复。如果当前数据包丢失，重传数据包的Packet Number就会在当前发送数据包的Packet Number之上加一。这便在接收端解决了重传模糊的问题，使接收端对RTT也可以有更精确的估计。

1.2.4 无缝连接迁移特性



传统TCP的连接建立时依赖四元组（源IP，源端口，目的IP，目的端口）来标识一条连接。这便导致如果用户使用蜂巢网络在基站之间进行切换，IP地址肯定也会发生变化，TCP就需要中断再重新建立连接。这一特性使得TCP在高速移动的环境下（如高铁环境）的性能表现非常不稳定。而QUIC协议的连接并不依赖IP和端口的四元组。通过一个64位随机数生成的Connection ID，QUIC可以唯一标识一条连接。即使在数据传输过程中IP和端口都发生了改变，只要客户端和服务器端都记录着相同的Connection ID，那么上层的业务逻辑就能继续保持连接。

## QUIC协议的研究现状

本文讨论的QUIC相关研究主要分为以下几个方面：QUIC协议的拓展，QUIC协议性能的测量以及QUIC协议其他方面的研究，如安全性分析等。

1.2.1 QUIC协议的拓展

在多径TCP（multipath TCP，MPTCP）的启示下，2017年De Coninck等人*[索引]*率先设计了多路径QUIC (multipath-QUIC，MPQUIC)作为QUIC的扩展。项目的基础为开源代码quic-go。他们对每个流下的每个包的公共包头添加了路径ID（Path ID），用以描述该数据包在传输时对应的不同物理网络路径。他们在Mininet仿真环境下的实验表明：多径传输在大部分情况下能改善QUIC性能。此外，对于传输大文件而言，无论是在高延迟/低延迟还是有丢包/无丢包的情况下，MPQUIC相比于MPTCP都有更好的性能表现。对于传输小文件而言，多径传输的表现并不好。这是因为使用多径传输需要多一次握手，这让传输时间有所延长。但如果初始连接在一条状况良好的链路上，多径的增益或许还能有所体现。

*[索引]De Coninck Q, Bonaventure O. Multipath quic: Design and evaluation[C]//Proceedings of the 13th international conference on emerging networking experiments and technologies. 2017: 160-166.*

2018年Viernickel等人*[索引]*对MPQUIC进行了真实环境和Mininet仿真环境下的部署和性能测试。实际测试时，两条网络链路分别是WIFI和LTE。LTE的数据接收是通过一台Nexus 5移动设备连接笔记本电脑的USB接口实现的。他们用不同大小的文件和网页对MPQUIC、MPTCP、QUIC以及TCP进行了比较测试。结果显示MPQUIC性能皆优于QUIC，TCP，MPTCP。

*[索引]Viernickel T, Froemmgen A, Rizk A, et al. Multipath QUIC: A deployable multipath transport protocol[C]//2018 IEEE International Conference on Communications (ICC). IEEE, 2018: 1-7.*

2019年De Coninck Q等人*[索引]*针对QUIC协议的可拓展性展开研究，提出Pluginized QUIC (PQUIC)框架，该框架允许QUIC客户端和服务器在每个连接的基础上，动态地交换协议的拓展插件。如多径连接或前向纠错等拓展。他们的系统实现了对QUIC拓展插件的动态加载，而不用在连接建立，握手时确定插件的拓展。这种安全且可扩展的技术，使协议插件能够通过QUIC连接按需交换。PQUIC的实现基础为以C语言为基础的picoquic*[小索引1]*，（这是IETF QUIC最完善的QUIC协议的实现版本之一）。此外，他们在quic-go版本也有实现*[小索引2]。*PQUIC系统实现的动态插入QUIC拓展插件包括：多路径QUIC拓展*[小索引3]*，不可靠的数据报扩展*[小索引4]*和一些前向纠错拓展。测试结果表明：作者他们在可接受的开销下，实现了在QUIC协议实时运行环境下（Protocol Runtime Environment），拓展插件动态插入的效果。

*[索引]De Coninck Q, Michel F, Piraux M, et al. Pluginizing quic[M]//Proceedings of the ACM Special Interest Group on Data Communication. 2019: 59-74.*

*[小索引1] Christian Huitema. 2018. picoquic. Source code. https://github.com/ private-octopus/picoquic.【这个索引格式对吗】*

*[小索引2] De Coninck Q, Bonaventure O. The Case for Protocol Plugins[R]. Technical Report. https://hdl. handle. net/2078.1/216493, 2019.*

*[小索引3]De Coninck Q, Bonaventure O. Multipath quic: Design and evaluation[C]//Proceedings of the 13th international conference on emerging networking experiments and technologies. 2017: 160-166.*

*【重复重复！】*

*[小索引4]Pauly T, Kinnear E, Schinazi D. An Unreliable Datagram Extension to QUIC[J]. Internet Engineering Task Force.(September 2018). draft-pauly-quicdatagram-00, 2018.*

1.3.2 QUIC协议的安全性分析

【补】对于QUIC协议的安全性研究与分析也有很多，例如：*[索引]...[索引]*

*【cited from QUIC中文综述】*

*(****2,31,4,49****,50,32,31,48,51)*

*[索引]Fischlin M, Günther F. Multi-stage key exchange and the case of Google's QUIC protocol[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014: 1193-1204.*

*[索引]Lychev R, Jero S, Boldyreva A, et al. How secure and quick is QUIC? Provable security and performance analyses[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 214-231.*

*[索引]Jager T, Schwenk J, Somorovsky J. On the security of TLS 1.3 and QUIC against weaknesses in PKCS# 1 v1. 5 encryption[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 1185-1196.*

*[索引]Aviram N, Schinzel S, Somorovsky J, et al. {DROWN}: Breaking {TLS} Using SSLv2[C]//25th {USENIX} Security Symposium ({USENIX} Security 16). 2016: 689-706.*

*...*

1.3.3 QUIC协议的性能测试

2017年Zhang等人*[索引]*首次系统性地测试了QUIC在卫星网络下的性能。文章通过使用tc Netem控制输出队列，模拟卫星网络环境，并将QUIC、HTTP/2和HTTPS在仿真环境下的性能进行测试比较。实验结果表明，QUIC有助于减少卫星网络下网页页面的加载时间。QUIC在传播延迟较长和丢包率较高的环境下，性能要优于HTTPS和HTTP/2。

*[索引]Zhang H, Wang T, Tu Y, et al. How quick is quic in satellite networks[C]//International Conference in Communications, Signal Processing, and Systems. Springer, Singapore, 2017: 387-394.*

随后在2018年，南京大学Wang等人*[索引]*首次在卫星网络环境下，测量QUIC协议的BBR拥塞控制算法，并与Cubic拥塞控制算法的性能进行对比。他们利用tc Netem控制数据队列，模拟卫星网络环境。实验结果表明：（1）在有效吞吐量（Goodput Performance）方面，当丢包率在卫星网络下常见的1%-10%之间时，BBR比Cubic的有效吞吐量更高。（2）在公平性方面，BBR的多用户（2个）平均有效吞吐量更高，故公平性更好。即卫星网络下BBR拥塞控制算法比Cubic算法性能更好。

*[索引]Wang Y, Zhao K, Li W, et al. Performance evaluation of QUIC with BBR in satellite internet[C]//2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE). IEEE, 2018: 195-199.*

2017年Kakhki等人*[索引]*对多版本的QUIC协议在桌面端、移动端以及有线、无线等环境下进行测量与比较，并与TCP协议进行并发请求等测试。主要结论为：（1）在桌面端（desktop）环境下，QUIC几乎在每个测试场景中都优于TCP+HTTPS。这是因为QUIC有着0-RTT连接、快速恢复丢包等特性。（2）在无序数据包的情况下，QUIC容易将这种乱序“理解”为丢包，从而影响性能。（3）QUIC依赖于应用层数据包处理和加密，这对硬件的计算能力提出了要求。因此QUIC在较老版本的手机上表现性能都有所下降。（4）在带宽波动的情况下，QUIC性能优于TCP。QUIC的ACK实现消除了ACK的歧义性，所有包的Packet Number都是单调增加，即便是重发的包也不再是原来的序号，而是在当前包的基础上加一，从而保证了QUIC能更精确地估计RTT和带宽。（5）QUIC的公平性不好。当与TCP流竞争时，QUIC占据的带宽是理想平均带宽的两倍。（6）QUIC在流媒体上的性能提升并不大，只有在传输高比特率视频的时候有一定的优势。（7）未优化的QUIC代理在丢包环境下，可以提高大文件的传输性能。但由于缺少了0-RTT特性，小文件的传输性能会受到影响。（8）自2016年以来，QUIC的性能改善主要源于扩大了原先保守的最大拥塞窗口。

*[索引]Kakhki A M, Jero S, Choffnes D, et al. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols[C]//Proceedings of the 2017 Internet Measurement Conference. 2017: 290-303.*

2017年Bhat等人*[索引]*研究了QUIC和TCP在DASH流媒体上面的性能。作者他们对比了不同网络环境下，运行在QUIC和TCP之上的自适应算法的QoE，其中自适应算法包括：BBA*[小索引1]*，SQUAD*[小索引2]*，BOLA*[小索引3]*算法。实验平台搭建依赖CloudLab*[小索引4]*，带宽控制使用红牛数据集*[小索引5]*（RedBull dataset）

他们发现QUIC并没有为当前的DASH算法提供一个提升，相反会带来一个QoE的下降。作者认为：大多数高质量的自适应算法都经过了优化，以很好地与TCP协作。它们没有充分利用QUIC提供的特性。比如ABR算法都基本并不会考虑QUIC协议的多路复用特性等。将在TCP上成熟的ABR算法直接搬移到QUIC上，使得性能反而有所倒退。文章提出了几个根据QUIC特性的DASH改进思路：（1）原本基于TCP的自适应码率选择算法可以考虑QUIC的多路复用等特性。（2）目前所有的DASH系统对于拥塞控制包括两个层面，DASH系统本身层面与TCP传输层层面。而QUIC支持使用自定义与标准的拥塞控制协议（如TCP Cubic, TCP NewReno和BBR*[小索引6]*）。DASH客户端可以禁用QUIC层面的拥塞控制或者对拥塞控制进行适合流媒体系统结合QUIC协议的修改，以提高性能。

此外，关于流媒体系统的自适应算法运行于QUIC协议之上的研究，Timmerer等人*[小索引7]*的研究也得出了类似的结论。

*[索引]Bhat D, Rizk A, Zink M. Not so QUIC: A performance study of DASH over QUIC[C]//Proceedings of the 27th workshop on network and operating systems support for digital audio and video. 2017: 13-18.*

*[小索引1]Huang T Y, Johari R, McKeown N, et al. A buffer-based approach to rate adaptation: Evidence from a large video streaming service[C]//Proceedings of the 2014 ACM conference on SIGCOMM. 2014: 187-198.*

*[小索引2] Wang C, Rizk A, Zink M. SQUAD: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP[C]//Proceedings of the 7th International Conference on Multimedia Systems. 2016: 1-12.*

*[小索引3]Spiteri K, Urgaonkar R, Sitaraman R K. BOLA: Near-optimal bitrate adaptation for online videos[J]. IEEE/ACM Transactions on Networking, 2020, 28(4): 1698-1711.*

*[小索引4]Ricci R, Eide E, Team C L. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications[J]. ; login:: the magazine of USENIX & SAGE, 2014, 39(6): 36-38.*

*[小索引5]Lederer S, Müller C, Timmerer C. Dynamic adaptive streaming over HTTP dataset[C]//Proceedings of the 3rd multimedia systems conference. 2012: 89-94.*

*[小索引6] Cardwell N, Cheng Y, Gunn C S, et al. BBR: Congestion-based congestion control[J]. Queue, 2016, 14(5): 20-53.*

*[小索引7] Timmerer C, Bertoni A. Advanced transport options for the dynamic adaptive streaming over HTTP[J]. arXiv preprint arXiv:1606.00264, 2016.*

2019年Ruth等人*[索引]*研究了用户对于QUIC的体验质量。他们是第一个调研真实用户基于QUIC协议体验的研究，并将数据公开在*[小索引]*。他们对于用户进行调研，发现虽然事实上在大多数情况下QUIC确实比TCP性能更好一些，但TCP和QUIC带给用户的体验的差距并没有多大。作者他们在修改的mahimahi*[小索引2]*框架下进行实验,调研用户对于QUIC的体验（QoE），用户被要求回答网页加载过程中是否有协议的切换，并对网页加载速度进行打分。实验结果表明：在网络环境足够好的情况下，用户并不会感到两种协议的明显区别。而网络环境中存在延时或丢包时，QUIC协议的优势会体验出来，其对于长尾的糟糕体验（long-tail of bad experiences）有很大的提升，更多的用户可以区分出两个协议的性能的区别。然而总的来说，在较慢的网络中，人们才倾向于使用QUIC，其对于用户的体验提升并不是大幅度的。另外作者提出：如果QUIC协议在隐私性等其他方面有着更好的设计，或许可以使更多用户选择QUIC协议。

*[索引]Rüth J, Wolsing K, Wehrle K, et al. Perceiving QUIC: Do users notice or even care?[C]//Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies. 2019: 144-150.*

*[小索引]https://study.netray.io*

*[小索引2]Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-andReplay for HTTP. In USENIX Annual Technical Conference (ATC). https://www. usenix.org/conference/atc15/technical-session/presentation/netravali.*

另外还有很多对于QUIC性能的研究*[小索引1，2，3，4]*总得来说大家都认为：QUIC性能在大多数网络环境下都优于传统TCP。

*[索引1]Biswal P, Gnawali O. Does quic make the web faster?[C]//2016 IEEE Global Communications Conference (GLOBECOM). IEEE, 2016: 1-6.*

*[索引2]Cook S, Mathieu B, Truong P, et al. QUIC: Better for what and for whom?[C]//2017 IEEE International Conference on Communications (ICC). IEEE, 2017: 1-6.*

*[索引3]Megyesi P, Krämer Z, Molnár S. How quick is QUIC?[C]//2016 IEEE International Conference on Communications (ICC). IEEE, 2016: 1-6.*

*[索引4] Yu Y, Xu M, Yang Y. When QUIC meets TCP: An experimental study[C]//2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). IEEE, 2017: 1-8.*

2018年，Marx等人*[索引]*为了增强QUIC协议的可测试性（Testability）和可调试性（Debuggability）,提出了QUIC通用日志格式的第一个版本，称为qlog。qlog可以帮助用户更好地了解QUIC协议的运行状态。他们开发QUICvis软件工具集（toolset）*[小索引1]*将运行状态可视化。显示状态包括：时间线图（timeline）, 序列图（sequence diagram）和拥塞/流控制图（congestion/flow control graph）。最后，他们将实验成果全部开源于*[小索引2]*。

*[索引]Marx R, Lamotte W, Reynders J, et al. Towards QUIC debuggability[C]//Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC. 2018: 1-7.*

*[小索引1]*[*https://github.com/rmarx/QUICvis*](https://github.com/rmarx/QUICvis)

*[小索引2]https://quic.edm.uhasselt.be*

1.3.4 QUIC协议的其他方面研究

2018年Hussein等人*[索引]*基于QUIC实现了一种增强的软件定义网络（Software Defined Networking，SDN）。在传输大量数据时，可以在更高的带宽利用率下提高安全性。文章认为当前的QUIC存在着在高带宽、高流量条件下性能较差、安全性不足、FEC机制在可变网络下性能较差等问题。于是引入QUIC感知的SDN架构。通过SDN架构，可以有效地均衡网络请求，测试目标地址的支持协议，以此更好的减小网络延迟以及增强安全性。

*[索引]Hussein A, Kayssi A, Elhajj I H, et al. SDN for QUIC: an enhanced architecture with improved connection establishment[C]//Proceedings of the 33rd Annual ACM Symposium on Applied Computing. 2018: 2136-2139.*

2018年Wang等人*[索引]*通过将QUIC协议部署到内核中，将QUIC和TCP进行了性能比较。测试在虚拟机和定制的WIFI测试床(testbed)中完成。结果表明实验：QUIC性能在大多数情况下都优于TCP。并且在并行运行QUIC协议时，它们可以公平地共享所有的带宽。

*[索引]Wang P, Bianco C, Riihijärvi J, et al. Implementation and performance evaluation of the quic protocol in linux kernel[C]//Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. 2018: 227-234.*

此外，复旦大学计算机学院王新老师实验室的李学兵同学等人*[索引]*于2020年也完成了QUIC的综述。它们对QUIC协议从总体上做了一个比较宽泛的介绍。

*[索引]李学兵, 陈阳, 周孟莹, 等. 互联网数据传输协议 QUIC 研究综述[J]. 计算机研究与发展, 2020, 57(9): 1864.*

# 2前向纠错编码的发展与应用

本章主要介绍了前向纠错编码的概念与分类；自适应前向纠错算法的发展；前向纠错编码的历史与应用现状。最后介绍了QUIC协议结合前向纠错编码的相关研究。

## 2.1前向纠错编码

在现实生活中存在一些高丢包的复杂网络环境，如高铁上的移动设备用户频繁地进行基站间的切换；无人机高速运动；与异国的用户视频连线等。在这种存在大量丢包且用户对数据实时性要求较高的场景中，数据传输的延迟以及数据包丢失造成的卡顿将大大降低用户的体验质量（Quality of Experience，QoE）。保证数据的实时性也就成为了新的挑战。

传统解决丢包问题的策略是自动请求重传（Automatic Repeat-reQuest，ARQ），然而等待超时后再请求数据包重传的过程非常低效，往往不能满足用户对于数据实时性的要求。在这种背景下，前向纠错算法是一种较为理想的解决方案。前向纠错编码通过在数据包中加入冗余，可以让数据接收端在丢包情况下，直接从冗余中恢复丢失数据，而不需要发送端重传。这对数据传输的实时性有了很大的提升。

目前对于前向纠错编码的研究主要聚焦于两方面，分别是编码的策略研究与自适应前向纠错冗余度控制算法的研究。

## 2.2前向纠错编码策略

编码策略的本身是期望通过设计前向纠错算法改善恢复丢包的效率。前向纠错编码主要分为：分组码与卷积码。

### 2.2.1分组码

分组码在网络前向纠错编码中是将数据包分成不同的组（Block），每组数据进行独立的编码。例如将原始数据包中每k个数据包分为一组。每组中添加r个数据包，从而使分组长度变为n。

一般我们用符号(n,k)表示分组，n表示分组中的数据包总数，k表示码组中原始数据包的个数，r=n-k则表示冗余数据包的个数。

对于分组码而言，每个分组中可以最多容许r个数据包的丢失。超过r个数据包的丢失将无法通过分组内的数据进行恢复。

XOR编码是一种很简单原始的分组码，它的策略是将k个数据包分为一组，在k个数据包的最后添加1个冗余数据包，该数据包由分组中第一个数据包按顺序与其后数据包进行异或（XOR）操作求得。若在k+1个数据包中丢失了1个数据包，则可以通过自后向前的异或操作恢复丢失的数据包。XOR编码虽然逻辑简单实现方便，但是只能容许一个分组中丢失至多一个数据包，这在使用上带来了诸多不便。

Reed-Solomon编码*[索引][索引]*于1960年被提出，是一种经典的分组码。Reed-Solomon主要利用范特蒙矩阵或柯西矩阵的特性，对k个数据包进行分组，并根据k个数据包生成r个冗余数据包，k个数据包和r个冗余数据包组成一个长度为n个数据包的分组。在一个分组中，小于等于r个数据包的丢失可以通过其他数据包恢复出来。超过r个数据包的丢失则无法恢复。

*[索引]Reed-Solomon codes and their applications[M]. John Wiley & Sons, 1999.*

*[索引]Sudan M. Decoding of Reed Solomon codes beyond the error-correction bound[J]. Journal of complexity, 1997, 13(1): 180-193.*

目前IETF对于Reed-Solomon编码已经明确了添加冗余数据的执行方法*[索引]*。

*[索引]Roca V, Cunche M, Lacan J, et al. Simple reed-solomon forward error correction (FEC) scheme for FECFRAME[J]. 2013.*

### 2.2.2卷积码

卷积码最早是1955年由Elias提出的。其生成方式是将待传输的信息序列通过线性有限状态移位寄存器。它与分组码的主要区别在于：卷积码是有记忆编码，即编码器的n个输入不仅与当前时段的k个输入有关，还与存储器之前的m个输入有关。一般卷积码被表示为（n，k，m）编码。这表示编码中的n个输出， k个输入和m个移位寄存器。为了增强纠错能力，卷积码通过卷积引入了时间相关性，使得编码结果与之前的输入有关。

现在一般采用的译码方法是1967年Viterbi提出的最大似然译码法。

2017年Roca*[索引]*等人在3GPP多媒体组播（Multimedia Broadcast/Multicast Service，MBMS）环境中对Reed-Solomon码，卷积码的性能进行对比。他们在仿真环境OpenFEC下模拟3GPP标准的信道进行测试，发现是卷积码相比于分组码在解码的延时上更短。

*[索引] Roca V, Teibi B, Burdinat C, et al. Less latency and better protection with al-fec sliding window codes: A robust multimedia cbr broadcast case study[C]//2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE, 2017: 1-8.*

目前IETF已经明确了如何针对RLC算法添加冗余数据的执行方法*[索引]*。

*[索引]Roca V, Teibi B. Sliding window random linear code (rlc) forward erasure correction (fec) schemes for fecframe[J]. 2018.*

## 2.3自适应冗余度选择策略

上述的编码策略大多考虑的是固定的丢包率，传输的每组数据中添加的冗余量都是固定的。然而事实上，真实的网络环境非常复杂与多变。有时候添加冗余过多将导致带宽无谓的浪费，添加冗余过少则仍然不够恢复丢失的数据包，同样造成带宽浪费。

于是在2000年，Padhye等人*[索引]*率先提出自适应编码的概念。即根据网络环境的不同，动态地调整每份数据中的冗余量，以尽量保证冗余数据包被充分利用，不浪费多余的带宽。

*[索引]Padhye C, Christensen K J, Moreno W. A new adaptive FEC loss control algorithm for voice over IP applications[C]//Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference (Cat. No. 00CH37086). IEEE, 2000: 307-313.*

2002年，Barakat等人*[索引]*便研究了TCP中如何平衡FEC数据量的问题。他们的研究表明：在丢包环境中适当地增加冗余数据包可以增加网络通信的吞吐量。然而添加冗余数据包的比例存在一个最优值，超过这个最优值之后继续添加的冗余数据包都会使吞吐量下降。另外，如果链路中平均丢包率不变，对于分组码而言，若不增加冗余数据包的冗余度，增加分组的长度可以使数据更好地应对突发性丢包（burst loss）。

*[索引]Barakat C, Altman E. Bandwidth tradeoff between TCP and link-level FEC[J]. Computer networks, 2002, 39(2): 133-150.*

要确定每份数据中添加的冗余数据的量，这需要对网络状态能做到实时的精准预测与评估。

## 2.4 前向纠错编码发展现状

2014年Nagy等人*[索引]*提出利用FEC包结合RTP协议传输视频数据包。作者提到FEC不仅可以用于错误恢复，本身也可以用于拥塞控制。他们将控制逻辑分为STAY，PROBE，UP，DOWN。在STAY状态下若数据传输不丢包，则系统会进入PROBE状态，发送端尝试对数据添加冗余包进行网络的带宽探测，如果添加了冗余包之后数据发生了丢失就用多余的数据进行恢复，并在下次传输数据时继续保持原有的状态（STAY），如果添加了冗余包不发生丢包，则发送端进入UP状态，将视频传输码率提高。STAY状态下还存在丢包则进入DOWN状态。他们创新性地提出了一种新的拥塞控制算法(FBRA)。并与其他两种拥塞控制算法进行了性能比较（RRTCC与C-NADU）。他们在NS-2仿真平台，真实环境进行了测试，结果表明，RRTCC和C-NADU在带宽利用率和丢包率上做了相反的权衡。FBRA算法在性能上较为折中。它的goodput比C-NADU更好而丢包率比RRTCC更低。尽管都是相近类似的吞吐量，但FBRA为用户提供了增强的用户体验，因为它的包延迟变化和goodput变化都相对较低。

*[索引]Nagy M, Singh V, Ott J, et al. Congestion control using fec for conversational multimedia communication[C]//Proceedings of the 5th ACM Multimedia Systems Conference. 2014: 191-202.*

2004年Baldantoni等人*[索引]*针对无线环境提出一种新的自适应冗余添加策略。他们的码型选择了分组码。在算法方面他们参考了*[小索引]* 中提到的“平方根公式”，将吞吐量估计为：

*[小索引] Mathis M, Semke J, Mahdavi J, et al. The macroscopic behavior of the TCP congestion avoidance algorithm[J]. ACM SIGCOMM Computer Communication Review, 1997, 27(3): 67-82.*

其中T表示吞吐量，p是数据包发送的丢失概率，μ是瓶颈带宽。

在作者的FEC模型中，K是分组数据包个数，R是冗余包个数，N是总数据包个数。PN,K表示丢失一个TCP包又无法恢复的概率。即N个包中丢失超过R个包的概率。p则依旧是一个TCP包丢失的概率。

随后用PN,K置换掉（1）式中的p，得到更新的吞吐量模型：

则式中第一项与第二项相等时，吞吐量T是最优的解。

因此令两项相等。

其中R就是应该添加的冗余包的量。

在NS-2仿真环境中，使用Gilbert-Elliot模型模拟两个状态（two-state）离散时间的马尔可夫状态转移模型。对反馈信息提供的丢包率进行自适应地调整冗余度。发现在仿真环境中，模拟无线传输环境，自适应添加了冗余的TCP相比于默认的TCP以及NewReno等拥塞控制算法，有效吞吐量（goodput）会更高。然而相比于Westwood等针对无线网络的新型TCP拥塞控制策略，性能表现还有一定差距。

*[索引]Baldantoni L, Lundqvist H, Karlsson G. Adaptive end-to-end FEC for improving TCP performance over wireless links[C]//2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577). IEEE, 2004, 7: 4023-4027.*

2008年，第三代合作伙伴项目(3GPP)建议在组播环境*[索引][x]*中使用前向纠错算法，并采用旋风码（Raptor）*[xx]*作为纠错编码。Raptor编码最早在*[30]*中被提出。

*[索引]3GPP, TS 26.346 V7.8.0 Technical Specification Group Services and System Aspects; MBMS; Protocols and codecs (Release 7). Jun. 2008.*

*[x]3GPP, TSG SSA, IP Multimedia Subsystem (IMS) – Stage 2 (Release 7), TS23.228 v.7.3.0,2006-03.*

*[xx]Raptor Forward Error Correction Scheme for Object Delivery 【draft】*

*[30] A. Shokrollahi, “Raptor codes,” IEEE Trans. Inf. Theory, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.*

2010年，Alexiou等人*[索引]*首次对在LTE蜂巢网络环境下，3GPP标准化的Raptor-FEC机制应用于组播数据传输（MBMS）中的性能进行验证。Raptor编码本质是一种喷泉码，其恢复数据包失败的概率为：

上式中m为收到的数据包的个数，k为源数据包的个数，为恢复数据包失败的概率。他们在仿真环境下针对一定丢包率的组播环境，不同组播用户人数（Multicast User Population）计算传输一个完整的文件的总成本。总成本是初始文件传输成本、FEC编码导致的冗余数据包传输成本和丢失数据包的选择性重传成本的总和。实验结果表明，在有一定丢包率的组播环境中，根据丢包率适当地增加冗余数据包可以有效地减少组播网络中的传输成本。

*[索引]Alexiou A, Bouras C, Kokkinos V, et al. Adopting FEC for reliable multicasting over LTE networks[C]//Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems. 2010: 307-310.*

2005年Ahn等人*[索引]*针对有损的无线信道环境（802.11标准），提出AFECCC自适应FEC算法。他们初始化一个定时器(Drop Timer，DT)，如果在一个Drop Timer期间所有的数据包都成功传输，那么冗余级别就降低一级。如果检测到丢包，那么冗余级别就会立即上升一级。并在每次更新状态时利用参数与对定时器时长进行修改。他们所采用的码型为Reed-Solomon编码。

文章在仿真实验中证实了在双状态马尔可夫链模型、GE模型以及混沌图（chaotic map）模型中，AFECCC算法比固定冗余度FEC策略和先前两个ARQ结合FEC冗余控制算法（LA-IR及LA-IR Ⅱ *[小索引]*）性能更佳，比较参数为不同误码率（Bit Error Rate）下的传输性能比（Performance ratio）。

另外，作者他们将算法部署在真实系统上利用天线（sensor）进行传输，在真实的无线传输环境中（sensor networks）进行测量，将几种自适应FEC策略调整方法的性能放在一条有损信道上进行测试，设置带宽为512kbps，IEEE 802.11标准持续传输100 bytes的数据包。与LA-IR及LA-IR Ⅱ等ARQ结合FEC的算法及固定冗余比的RS冗余码进行性能对比，比较参数为在传输距离（TR Distance）下的传输性能比，该性能比的参照为无损环境下100bytes的数据包完全接收到。发现都是AFECC性能更优。

*[小索引] Levisianou A, Assimakopoulos C, Pavlidou F N, et al. A recursive IR protocol for multicarrier communications[C]//Int. OFDM Workshop. 2001.*

*[索引]Ahn J S, Hong S W, Heidemann J. An adaptive FEC code control algorithm for mobile wireless sensor networks[J]. Journal of Communications and Networks, 2005, 7(4): 489-498.*

2007年，Tsugawa 等人*[索引]*指出：AFECCC算法中定时器的更新公式中>0，0<<1，算法效果很依赖于这两个参数的设置。另外该算法与TCP直接结合时效果并不好，比如TCP拥塞控制算法在未丢包的情况下不断地扩大拥塞控制窗来试探带宽，在这个过程中AFECCC的冗余级别不断降低，当拥塞控制窗过大超过网络瓶颈带宽时，冗余级别应该足够高来保证丢包的恢复，然而这时AFECCC的冗余级别却是很低的。他们基于AFECCC提出了一个新的自适应FEC方案，称为TCP-AFEC，主要是针对流媒体（Video Streaming）场景。期望通过自适应添加冗余数据，在实时的视频传输场景中，减少延时，加速数据传输速率，提高用户体验。TCP-AFEC首先计算应用层期望的窗长Wapp=RTsrtt，R表示上层APP要求的传输速率。Tsrtt是最近的平滑处理过的RTT值。随后TCP-AFEC考虑到：丢包后TCP拥塞控制算法会使拥塞控制窗（Window Size，W）被降低很多，可能比期望的窗长要小，那么为了更快地恢复拥塞控制窗，TCP-AFEC考虑添加比平常更多的冗余。TCP-AFEC引入一个阈值来判断冗余级别是否需要更大。当发生丢包或重传时，就会更新这个阈值。

因为最小的拥塞控制窗为1，所以阈值Rth(t)的范围是：

若当前拥塞控制窗小于Rth(t)时，即使没有在DT中丢包，TCP-AFEC也不会降低冗余级别，这样避免了当拥塞控制窗减小时，冗余度变得过低，这样在随后拥塞控制算法增大拥塞控制窗时，冗余度还相对较高。其他的更新算法和AFECCC相同。即在AFECC算法上添加约束，否决不正确的决策。他们在NS-2仿真环境下，搭建了两个路由器连接多个客户端和服务器端构成的拓扑进行测试。他们在不同的链路上配置了一定的网络瓶颈带宽，传播时延以及一定的随机丢包概率（Random packet loss probability）。比较参数为不同的视频请求码率（Required Rate）下的数据到达比（Archievement ratio）。他们将TCP-AFEC与固定冗余度的静态（Static）FEC算法及TCP-SACK及TCP-AV性能进行对比，实验结果显示：TCP-AFEC在数据到达比方面性能相比于其他算法都更加优秀，性能更佳；另外要达到要求的视频码率，在固定瓶颈带宽的情况下，TCP-AFEC可以并发的连接相比于TCP-SACK及TCP-AV要更多。

*[索引]Tsugawa T, Fujita N, Hama T, et al. TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee[C]//Packet Video 2007. IEEE, 2007: 294-301.*

2006年，Convertino等人*[索引]*提出：在无线网络（IEEE 802.11）的视频传输场景下。数据包的丢失会导致视频质量的降低。虽然许多视频解码器可以掩盖少量随机丢包的影响，但是面对突发性丢包（burst loss）时是很难修正的。他们提出先前工作使用的独立损失模型或2状态马尔科夫模型并不能最优地描述损耗模式，前者的建模不适用于突发损失。因为它假设:丢失是独立的，并且以假定概率(平均丢包率)发生。后者是简单地将突发认为是连续的损失序列，而忽略了爆发性丢包时期的其他特征。在此基础上他们提出一个更准确的数据包丢失模型，用一个4状态马尔可夫模型去描述数据包达到的状态。四个状态包括：接收到了数据包；在一个突发性损失中收到了数据包；在一个突发性损失中丢失了数据包；在一个普通丢包（gap）状态中丢失了包。作者考虑的分组码是Reed-Solomon编码。他们的理念是分组添加冗余的数据包应该要尽量覆盖到突发性损失。提出一种新的自适应FEC冗余算法-XR算法

算法根据burst和gap状态来控制自适应FEC策略的冗余度n和k。其中gmin表示连续收到数据包的最小值，低于这个值便视为在burst loss状态，burst\_density表示突发性损失中丢失数据包的概率。他们在NS-2仿真环境下进行测试，模拟了802.11无线环境下不同的丢包模式。他们将自己提出的算法与平均丢包率模型（BasicPLR，basic packet loss rate）进行了比较，发现：XR算法能较好地平滑突发损失，并优于基于平均PER的自适应FEC调整方法。在恢复能力（Recovery Capcity，RC）上更强。引入了FEC效率（FEC Efficiency，FE）来评估算法性能，FE定义为：

其中FECredundancy表示FEC数据包的冗余比。实验表明XR算法的FEC效率在实验中在绝大多数环境下更高。

*[索引]Convertino G, Oliva S L. Adaptive FEC for 802.11 burst losses reduction[C]//Proceedings of the 2nd international conference on Mobile multimedia communications. 2006: 1-5.*

2014年，Flach等人提出TCP-IR的执行方案*[索引]*，他们将FEC（XOR）策略集成到了TCP中，通过固定的间隔插入XOR的冗余数据包，在接收端可以做到快速的数据恢复，以避免丢包后的重传。然而这在变化的网络环境中显然是不够合理的。

*[索引] Flach T, Dukkipati N, Cheng Y, et al. Tcp instant recovery: Incorporating forward error correction in tcp[J]. Working Draft, IETF Secretariat, Internet-Draft draft-flach-tcpm-fec-00, July, 2013.【这是一个draft】*

2018年，Ferlin等人*[索引]*提出基于XOR算法的动态FEC算法（TCP-dFEC），算法原型为TCP-IR，相比于TCP-IR，该算法实现了一个动态的FEC控制机制。另外他们将TCP-dFEC拓展为MPTCP-dFEC并在MPTCP环境下进行部署，最终在Linux内核中予以实现。

**if then**

**else**

**end if**

dFEC的核心思想就是控制剩余冗余（residual loss）。Ratio表示间隔多少个包加一个冗余数据包，T表示一个时间周期（interval）,N表示时间周期的序号。测试发现=0.33效果很好，实验中初始设置N=2，并且限制ratio>=4。进行冗余度的更新。Target参数自行配置，表示目标剩余冗余。他们使用CORE网络仿真器*[小索引]* ，在模拟的网络拓扑中进行实验。随后在真实的网络拓扑中，两个网络接口连接WIFI和蜂巢网络进行实验。实验结果表明：与传统TCP及TCP-IR相比，新算法有效提高了HTTP/2的web下载时间以及H.264格式视频的帧率。

*[小索引] Ahrenholz J. Comparison of CORE network emulation platforms[C]//2010-Milcom 2010 Military Communications Conference. IEEE, 2010: 166-171.*

*[索引]Ferlin S, Kucera S, Claussen H, et al. MPTCP meets FEC: Supporting latency-sensitive applications over heterogeneous networks[J]. IEEE/ACM Transactions on Networking, 2018, 26(5): 2005-2018.*

2006年，Lohmar等人*[索引]*讨论了多媒体多播/广播服务(MBMS)功能下FEC功能的性能，对于多媒体多播/广播服务而言，有3种纠错方式。（1）前向纠错编码（2）点对点纠错(Point-To-Point，PTP)（3）点对多点纠错(Point-To-Multipoint ，PTM)。

他们在仿真环境(Georgia Tech Network Simulator,GTNetS)*[小索引]*中，模拟多媒体多播/广播服务(MBMS)功能下，对100000用户，对不同大小的文件（2 Mbytes, 1 Mbytes and 512 Kbytes）进行分发传输测试。他们用一个逆指数分布函数来模拟每个接收器上的数据丢失概率。

实验结果验证了在组播环境下，相对固定的丢包状态下，冗余度的选择存在一个最优解，较少地添加冗余会导致丢失数据的重传，较多地添加冗余会造成带宽的浪费。另外，适当添加FEC后使用PTP结合PTM的修复数据方式比单纯FEC结合PTP的性能更好，可以更快地完成数据的传输。

*[索引]Lohmar T, Peng Z, Mahonen P. Performance evaluation of a file repair procedure based on a combination of MBMS and unicast bearers[C]//2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06). IEEE, 2006: 9 pp.-357.*

*[小索引] Georgia Tech Network Simulator (GTNetS), http://www.ece.gatech.edu/research/labs/MANIACS/GTNet S*

2012年，Bouras等人*[索引]*研究了第三代合作伙伴计划3GPP中最新纠错码RaptorQ(一种新的旋风码，一种喷泉码)的性能，在仿真环境下与3GPP使用的标准Raptor码进行对比。RaptorQ码的编码过程与Raptor码的编码过程几乎相同。但是区别在于，传统Raptor算法运行于GF(2)域*[小索引1]* ，增强的RaptorQ算法运行于GF(256)域而不是G(2)域。在更大的有限域上进行操作。使得RaptorQ可以将56403个源符号编码到一个源码分组（source block）中，而旋风代码只有8192个，而且RaptorQ可以生成多达16777216个编码符号(encoding symbols)，比旧的旋风代码多256倍。同时，RaptorQ引入了八进制运算并结合低复杂度的GF(2)操作，使得绝大多数符号操作都运行于GF(2)中，只有一小部分运行于GF(256)上*[小索引2]*。引入GF(256)使得收到k+m个symbols后恢复出一个symbol的概率为，原本理想恢复概率为。他们在一个开源仿真平台*[小索引3]*上模拟多个移动用户设备在组播环境中的情况，在带宽相同，冗余数量相同的情况下，RaptorQ保护的用户比例更高，即收到相同数量的额外的包解码成功概率更高。并且Raptor为了完成所有数据的下载需要额外传输数据的包会更多。

*[小索引1] Luby M, Shokrollahi A, Watson M, et al. Raptor forward error correction scheme for object delivery[R]. RFC 5053 (Proposed Standard), 2007.*

*[小索引2] 3GPP, “Rationale for MBMS AL-FEC Enhancements,” 3rd Generation Partnership Project (3GPP), Tdoc S4-110449, 2011.*

*[小索引3] Piro G, Grieco L A, Boggia G, et al. Simulating LTE cellular systems: An open-source framework[J]. IEEE transactions on vehicular technology, 2010, 60(2): 498-513.*

*[索引]Bouras C, Kanakis N, Kokkinos V, et al. Evaluating RaptorQ FEC over 3GPP multicast services[C]//2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2012: 257-262.*

2004年，Lundqvist等人*[索引]*提出：在端到端TCP传输中使用FEC编码时，编码分组的大小不能大于当前拥塞控制窗口的大小。最好保留一些宽裕度。当发送窗口很小时，最好不要使用FEC，这会增加丢包的风险。他们提出了自适应的FEC算法，其逻辑为：（1）如果当前win\_size < L1 (eg.10 pkt)，则关闭FEC策略。

（2）如果当前L1 < win\_size < L2(eg. 40 pkt)，则使block\_size为当前win\_size一半。（3）如果当前win\_size > L2，则使block\_size为设定的maximum value。在实际操作中，作者还为FEC更新策略添加更新滞后条件，以避免FEC开关过于频繁。作者控制只在窗口大小减少到8个包后禁用FEC策略，并在窗口大小增加到12个包以上后再次启用它。这样操作，发送冗余信息的比例大约在5% - 25%之间。作者在NS-2仿真平台上进行测试，与不添加FEC冗余包的TCP进行对比，发现在丢包和延时较大时利用自适应FEC策略可以有效地提高吞吐量。

*[索引]Lundqvist H, Karlsson G. TCP with end-to-end FEC[C]//International Zurich Seminar on Communications, 2004. IEEE, 2004: 152-155.*

2020年，为了更好地利用前后状态的关联性，Cheng等人*[索引]*提出了DeepRS自适应冗余控制算法，他们利用长短期记忆（LSTM）网络作为深度学习算法，通过预测包丢失的概率来自适应地调整RS编码器的冗余比。其中LSTM网络常用于处理序列化的数据。在*[小索引]*中被首次提出。

*[小索引]Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.*

在实验上，网络的输入为历史的包丢失模式，输出为预测的包丢失模式。LSTM网络在深度学习模型中常用于处理序列化的数据。仿真实验模型为Gilbert-Elliot信道，并利用真实的数据集*[15]*进行仿真环境设计。实验将算法与多个固定冗余度的FEC添加策略对比，并以恢复比（Recovery Ratio）作为性能衡量指标。实验表明：相比于固定冗余度的RS策略，DeepRS在实验测试中至多提升了恢复比70%。

*[小索引]Fu S, Zhang Y, Jiang Y, et al. Experimental study for multi-layer parameter configuration of WSN links[C]//2015 IEEE 35th International Conference on Distributed Computing Systems. IEEE, 2015: 369-378.*

*[索引]Cheng S, Hu H, Zhang X, et al. DeepRS: Deep-learning Based Network-Adaptive FEC for Real-Time Video Communications[J]. arXiv preprint arXiv:2001.07852, 2020.*

## 2.5 QUIC协议结合前向纠错编码发展现状

谷歌曾经在较早版本的QUIC协议中尝试使用前向纠错拓展，主要通过在分组中简单添加一个XOR冗余数据包实现*[5]*。但是由于在实际测试过程中，FEC虽然能降低重传率，但是在流媒体服务和网页服务中效用却并不明显，前向纠错编码反而给使用带宽造成了压力。所以最终谷歌公司在QUIC中舍弃了FEC*[6]。*但是由于其测试场景基本都是低延迟与低丢包，并没有充分考虑如高铁，无人机等延迟较高，丢包较大的情况。因此并不能完全地否定前向纠错编码的作用。

1. *R. Hamilton et al., “Quic: A udp-based secure and reliable transport for http/2,” Working Draft, Internet-Draft draft-hamilton-early-deploymentquic-00, July 2016.*
2. *The QUIC Transport Protocol: Design and Internet-Scale Deployment sigcomm 2017*

2019年Michel等人*[索引]*为了研究QUIC可靠传输机制，实现并评估了QUIC内的前向纠错拓展。设他们在开源项目quic-go的基础上定义了一个FEC Frame帧，可支持XOR，Reed-Solomon与卷积RLC编码。并在Bitbucket上对代码进行了开源*[小索引1]*。

实验在Mininet仿真环境下进行，丢包模型调用的是Gilbert-Elliott模型。为了避免loss-bassed策略的拥塞控制信号错误，实验对收到的包(received packets)和恢复的包(recovered packets)做了区分。实验中他们根据当前IETF标准*[小索引2]*在quic-go项目基础上增加了packet-based 检测机制，不同于传统的time-based 丢失检测机制。实验中对不同的FEC策略，不同大小的文件等参量进行控制，最后比较量为下载完成时间。（Download Complete Time，DCT）

他们在实验中证明：传输大文件或当网络环境中延迟与丢包比较小时，FEC冗余编码会对下载完成时间产生负面影响。当传输小文件或是当网络环境中延迟与丢包比较大时，FEC冗余编码由于避免了等待重传的时间(timeout)可以极大程度地减少下载时间。

对于他们冗余方案的更加详细的介绍可以参考*[小索引3]*。

*[小索引1]*

*https://bitbucket.org/michelfra/quic-fec/src/networking\_2019/*

*[小索引2]Iyengar J, Swett I. Quic loss detection and congestion control[J]. Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-recovery-00, 2016.*

*[索引]Michel F, De Coninck Q, Bonaventure O. QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC[C]//2019 IFIP Networking Conference (IFIP Networking). IEEE, 2019: 1-9.*

*[小索引3]Michel F, De Coninck Q, Bonaventure O. Adding forward erasure correction to quic[J]. arXiv preprint arXiv:1809.04822, 2018.*

2019年Garrido等人*[索引]*提出了rQUIC，首次在QUIC中集成了FEC功能的框架。他们设计并实现了一种自适应XOR的FEC算法。

他们在定义一个FEC报头字段为4字节长，在报头中定义了:类型(Type)，分组ID（Block ID）, 冗余比(FEC ratio)和计数（Count）。类型表示数据包是原始数据包还是FEC包。冗余比确定了多少个包受到FEC包的保护。分组ID标记同一分组（block）的包，计数表示数据包在分组中的序号。

他们的自适应算法策略为：记录一段时间T内总的传输成功的数据包和重传的数据包,并且计算剩余残差（residual loss）。剩余残差表示为：重传的数据包个数除以传输成功的数据包的个数与重传的数据包的个数的差。

他们计算N个时间段内平均剩余残差：

如果平均剩余残差超过一个目标值，则冗余比增加。

他们在NS-3仿真环境下进行了实验，模拟了低时延（25ms），高带宽（20Mbps）的WiFi/LTE网络以及中等时延（100ms），中等带宽（10Mbps）的2G/3G网络。在实验中改变随机丢包率，设置每次记录的时间间隔period T=3RTT，参数 =0.33， =1%，并将rQUIC与QUIC对比传输20MB的大文件与1776KB的网站（www.flickr.com）的完成时间进行对比。

实验结果表示：对于存在随机丢包的网络环境下，rQUIC在传输大文件时至多可以减少60%的传输时间。对于传输小文件，则rQUIC的性能依旧比QUIC的性能更佳。

*[索引]Garrido P, Sánchez I, Ferlin S, et al. Poster: rQUIC-integrating FEC with QUIC for robust wireless communications[C]//2019 IFIP Networking Conference (IFIP Networking). IEEE, 2019: 1-2.*

目前IETF也给出了QUIC协议中添加RLC冗余编码的执行方案*[索引]*。

*[索引]Roca V, Swett I, Montpetit M J. Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC[J]. 2018.*

# 结束语

随着谷歌提出的QUIC协议近年来的不断发展，网络中越来越多的流量开始使用QUIC进行传输。本文首先对QUIC协议进行简要介绍，详细阐述前向纠错编码的概念和理论，对传统网络传输协议中应用自适应前向纠错编码的算法进行了全面的综述；文章的最后一部分对QUIC协议中应用自适应前向纠错编码的现状进行了介绍。目前我们的工作研究也正基于QUIC协议在不同网络环境下的自适应前向纠错编码，也希望在未来的工作中可以为QUIC协议前向纠错方面的性能做到更大的提升。

作者简介：

**林利祥** 复旦大学信息科学与工程学院硕士研究生。主要研究方向：流媒体视频传输，网络传输协议，前向纠错编码。