CellFusion: Multipath Vehicle-to-Cloud Video Streaming with Network Coding in the Wild

Yunzhe Ni[*1,2], Zhilong Zheng[*1], Xianshang Lin[1], Fengyu Gao[1], Xuan Zeng[1], Yirui Liu[1], Tao Xu[1], Hua Wang[1], Zhidong Zhang[1], Senlang Du[1], Guang Yang[1], Yuanchao Su[1], Dennis Cai[1], Hongqiang Harry Liu[3], Chenren Xu[2], Ennan Zhai[1], Yunfei Ma[1†]

[1]Alibaba Cloud [2]Peking University [3]Uber Technology

ABSTRACT

This paper presents CellFusion, a system designed for high-quality, real-time video streaming from vehicles to the cloud. It leverages an innovative blend of multipath QUIC transport and network coding. Surpassing the limitations of individual cellular carriers, CellFusion uses a unique last-mile overlay that integrates multiple cellular networks into a single, unified cloud connection. This integration is made possible through the use of in-vehicle Customer Premises Equipment (CPEs) and edge-cloud proxy servers.

In order to effectively handle unstable cellular connections prone to intense burst losses and unexpected latency spikes as a vehicle moves, CellFusion introduces XNC. This innovative network coding-based transport solution enables efficient and resilient multipath transport. XNC aims to accomplish low latency, minimal traffic redundancy, and reduced computational complexity all at once. CellFusion is secure and transparent by nature and does not require modifications for vehicular apps connecting to it.

We tested CellFusion on 100 self-driving vehicles for over six months with our cloud-native back-end running on 50 CDN PoPs. Through extensive road tests, we show that XNC reduced video packet delay by 71.53% at the 99th percentile versus 5G. At 30Mbps, CellFusion achieved 66.11% $\sim$ 80.62% reduction in video stall ratio versus state-of-the-art multipath transport solutions with less than 10% traffic redundancy.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Cross-layer protocols**.

KEYWORDS

Multipath QUIC, Network Coding, Video Streaming, Vehicular Networks, Self-driving

ACM Reference Format:
Yunzhe Ni[*1,2], Zhilong Zheng[*1], Xianshang Lin[1], Fengyu Gao[1], Xuan Zeng[1], Yirui Liu[1], Tao Xu[1], Hua Wang[1], Zhidong Zhang[1], Senlang Du[1], Guang Yang[1], Yuanchao Su[1], Dennis Cai[1], Hongqiang Harry Liu[3], Chenren Xu[2], Ennan Zhai[1], Yunfei Ma[1†]. 2023. CellFusion: Multipath Vehicle-to-Cloud Video Streaming with Network Coding in the Wild. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3603269.3604832

[*]Equal contributions.
[†] Project lead and corresponding author. Email: yunfei.ma@alibaba-inc.com.

1 INTRODUCTION

The rise of autonomous and electric vehicles driven by companies like Tesla [1], Waymo [2], Porsche [3], and Toyota [4] is reinventing cars as "smartphones on wheels" [5]. As this shift unfolds, establishing reliable connections between these vehicles and the cloud is becoming increasingly essential. Video streaming from vehicles to the cloud unlocks a host of novel applications, spanning from in-vehicle entertainment and gaming to mission-critical tasks that require a higher degree of reliability and performance. For example, teleoperated driving (ToD) [6, 7], a mechanism which remotely overtakes autonomous vehicles when algorithms cannot effectively handle complex situations [8], requires real-time transmission of high-definition camera feeds from the vehicle to the cloud. Another application is remote diagnostic procedures [9, 10], where healthcare professionals guide paramedics in administering urgent treatment remotely. Such procedures can become a reality by transmitting high-definition views from inside an ambulance, along with the patient's vital signs, to the cloud.

The fundamental challenge in realizing the new applications described above is how to continuously support high-bitrate low-latency video streaming over highly fluctuating cellular links as a vehicle drives. However, we face two conflicting situations:

- **High delay & limited rate of fragile cellular links.** Cellular connection is known to be fragile, and many factors, such as proximity to a cell tower and obstacles, impact its bandwidth, jitter, and loss [11, 12]. The limited coverage often leads to a limited data rate and high packet delay. Moreover, as a vehicle drives, it can easily find itself in cellular "dead spots" [13]. The advent of 5G does not address this problem. Since 5G signals operate on higher frequencies than 4G, they suffer from greater fluctuation and attenuation, resulting in even smaller coverage [14, 15].
- **Data-intensive & low-latency vehicular applications.** In terms of latency and bandwidth, emerging vehicular applications can be much more demanding than traditional real-time applications such as VoIP and video conferencing [16, 17] that typically need 300Kbps - 2Mbps at < 300ms latency. Taking ToD as an example, in order for a vehicle to clearly "see" surrounding environments with sufficient sensing depth, it has to rely on the aggregated view of many high-definition cameras, requesting ~30Mbps at < 100ms one-way delay [1] [18, 20].

[1]For example, the Tesla model 3 is equipped with 8 cameras [18], whereas the 5GAA ToD model [19] assumes to use four 8Mbps cameras.

Today's deployed vehicle-to-internet solutions [21–23], such as Verizon connected car and Tesla premium connectivity, are limited by a single carrier's coverage. As a result, they suffer from frequent signal loss and limited link rate [24, 25]. Past research proposals [26–28] on vehicle connectivity focused on downloading non-real-time loads, and were not capable of serving high throughput traffic in real-time.

In this paper, we introduce CellFusion to meet the emerging challenge and enable continuous, high-quality, low-latency vehicle-to-cloud video streaming for the first time. Our idea is to go beyond the limitation of a single cellular network to build a resilient overlay last mile that efficiently fuses multiple heterogeneous cellular networks to the cloud. To realize this idea, CellFusion synergizes two fields: *multipath transport* and *network coding*.

First, CellFusion builds a novel hardware-software system that aggregates multiple heterogeneous cellular network resources (§3) as shown in Fig. 1(a): it deploys a specialized Customer Premise Equipment (CPE) box in a vehicle (§5), as shown in Fig. 2, and a group of distributed edge-proxies at CDN Point-of-Presence (PoPs) (§6) to establish multipath tunnels on top, as shown in Fig. 1(b). Our implementation aggregates four cellular networks combining 5G and LTE from three carriers. In this way, CellFusion embraces two types of diversity: (1) the geographical diversity of different carriers and (2) the frequency diversity of different technologies, leading to diversified cellular coverage. As a result, CellFusion significantly reduces the likelihood of cellular "dead spots" by utilizing diversified cellular resources that compensate for each other.

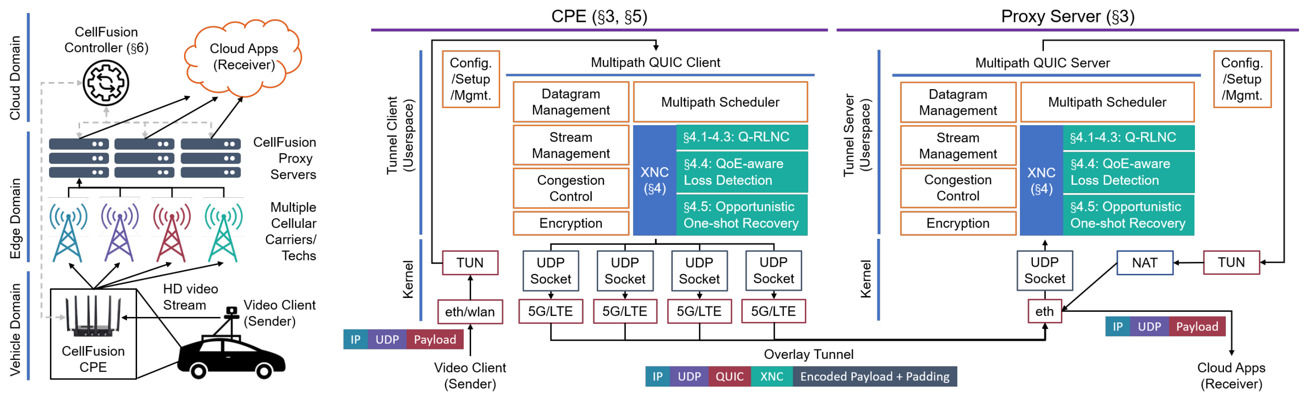
Second, CellFusion introduces XNC, as highlighted in the blue and green parts in Fig. 1(b), to efficiently fuse these underlay network resources to the cloud (§4). A fundamental problem for multipath transport in vehicle-to-cloud streaming is that when a vehicle drives, each cellular link becomes highly volatile and unpredictable, which makes it difficult to ensure the delay of a packet before the deadline [29]. As we show later, the cellular link of a high-speed vehicle suffers from *heavy bursty losses that can go up to 100% and high latency spikes that can reach a few seconds unexpectedly.* Generally, a multipath scheduler predicts path characteristics such as bandwidth and delay to pick the path for transmitting a packet. As a result, when a path becomes fluctuant, the scheduler is likely to make a *wrong* prediction on a path that might later degrade, causing excessive delay and poor bandwidth resource utilization [30, 31]. To address the problem, XNC leverages network coding, which mixes data across time to obtain resilience and efficiency against fragile cellular links through a unique collaboration with multipath.

Despite the fact that network coding [32, 33] and multipath [30] transport are relatively mature research areas, past solutions are intrinsically incapable when it comes to vehicle-to-cloud streaming. Firstly, traditional network coding-based protocols such as FEC [34], COPE [32], randomized linear code [35], TCP-NC [36], evolution code [26], LT code [37], PACE [38], and streaming code [39] fall short for one or more of the following reasons: (1) The code designed for random or bounded loss performs poorly in recovering the heavy bursty loss. (2) Encoding introduces high latency as a relay node (e.g., a CPE) has to wait for enough packets to perform block coding, (3) The code brings significant redundant traffic costs and reduces bandwidth for effective video transport. (4) The code is too computationally expensive to use at high bitrates. Secondly, today's deployed multipath transports such as MPTCP [40], MPQUIC [41], and XLINK [29] are designed as fully reliable transport, and hence, they cause excessive delay under high loss when sending real-time traffic.

CellFusion's XNC introduces multiple thrusts of innovations to address the above challenges all at once. At the base transport layer, instead of using reliable QUIC [42], XNC leverages the newly standardized QUIC-Datagram [43] as an unreliable medium and integrates multi-path features (§4.2). In doing so, it reuses as much as possible the QUIC transport features (e.g., congestion control, encryption, traversal of middleboxes) that already exist. On top of the base layer, XNC introduces a partially reliable transport mechanism with QUIC-based random linear network coding (Q-RLNC) (§4.3), QoE-aware loss detection (§4.4), and opportunistic one-shot recovery (§4.5). In a nutshell, XNC quickly detects video loss based on a QoE-aware policy and maximizes its recovery probability with opportunistic one-shot recovery by retransmitting a sufficient number of random linear combinations (equations) of lost packets to utilize all paths' instantaneous spare capacity opportunistically. As we will show later in this paper, XNC is designed with the following nice properties:

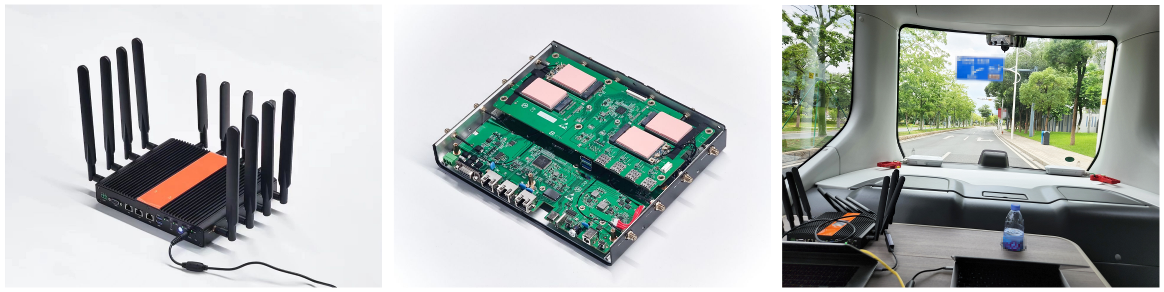
- *Robustness to bursty loss.* Instead of direct forward error correction, our scheme retransmits lost packets via Q-RLNC, which has a very flexible redundancy rate that matches the unpredictable loss rate in real-time, so XNC is highly robust to heavy bursty loss even up to a 100% loss rate.
- *Resilient to unpredictable multipaths.* Each coded packet is equivalent to a linear equation of lost packets. So long as a receiver gets enough equations, it can recover the entire lost range. In this way, XNC greatly reduces the impact of "bad" path scheduling because the loss impact of a coded packet on any path becomes identical and can be immediately remedied when another equation arrives.
- *Low latency on-the-fly coding.* In our scheme, a new packet is immediately forwarded, while coding is only applied to lost packets. Thus, no additional delay is required to accumulate a code block before encoding. Our retransmission scheme is one-shot and opportunistically uses paths' instantaneous spare capacity, so we also avoid delaying new packets due to bandwidth consumed by coded packets.
- *Low redundancy and low complexity.* Our scheme is basically a systematic code that mostly contains original packets. Therefore, it achieves almost zero traffic redundancy under good network conditions with no loss. Moreover, our coding scheme can be efficiently accelerated using SIMD instructions on embedded systems.

Architecturally, CellFusion is designed as a connectivity-as-a-service solution (§6). CellFusion's back-end is cloud-native. Thanks to the user-space nature of our transport stack, deploying CellFusion's proxy is as simple as running a lightweight container on edge clouds, which can be easily auto-scaled and managed to ramp up capacity and provide highly available edge access. Importantly, CellFusion is secure and transparent by nature. Unlike past works such as split-TCP [44], which breaks a user connection, CellFusion tunnels raw IP packets, so it acts transparently in the background and preserves the security of the original traffic. As an IP tunnel,



(a) End-to-end architecture (b) Component diagram and packet flow

Figure 1: The overview of CellFusion: (a) end-to-end architecture and (b) component diagram and packet flow.



(a) CellFusion CPE box (b) CellFusion CPE's customized board (c) Deployment in a self-driving vehicle

Figure 2: Photos of CellFusion CPE, its hardware board, and deploying CellFusion CPE in a self-driving vehicle.

CellFusion supports a broad range of passenger protocols, including RTP/RTCP, RTSP, and HTTPs.

Results: We have tested CellFusion on 100 self-driving vehicles for over six months, with our back-end running at 50 CDN PoPs across three states. A deployment photo is shown in Fig. 2(c). Through extensive road test evaluations, we show XNC achieved a 99th percentile video packet delay at 73.8ms, representing a 71.53% reduction vs. 5G. When streaming video at 30Mbps and 30fps over a 5000km driving distance, CellFusion achieved an average framerate, video stall ratio, and normalized structural similarity index measure (SSIM) score of 29.11fps, 0.99%, and 0.93, respectively, with less than 10% traffic redundancy. It achieved 66.11% ~ 80.62% reduction in video stall ratio versus state-of-the-art multipath transport solutions.

Contribution: To the best of our knowledge, CellFusion is the first system to support high-quality, real-time vehicle-to-cloud streaming in-the-wild, meeting today's ToD requirements. The core technical contribution of CellFusion is our design, implementation, and cloud-native solution that uniquely synergize multipath transport and network coding. We present the detailed software and hardware design and XNC transport algorithm. We believe that CellFusion holds great potential for widespread deployment by Cloud and CDN providers. Offering connectivity-as-a-service on a universal scale, CellFusion can provide the robust streaming infrastructure necessary to shape the future of mobility.

Claim: *This work does not raise any ethical issues.*

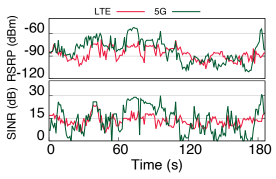
2 BACKGROUND AND MOTIVATION

2.1 Teleoperated driving for autonomous vehicles

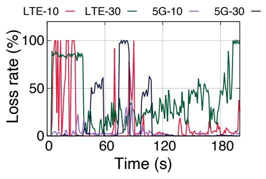
Full self-driving was anticipated to enable a market size of 2.3 trillion dollars by 2030 [45]. However, despite big companies and numerous startups pouring billions of dollars over the past decade, it has not come close to a wide-scale commercial deployment [46]. The real gap is the so-called "endless" edge cases [47]: situations such as poor visibility, temporary detours, a fallen tree, sinkholes on the road, and unexpected events may exceed today's algorithms' capability and cause damage, injury, and even death.

Teleoperated driving (ToD) [6, 7, 48], which allows remote intervention and control of a vehicle from a teleoperator when human experience is needed, is an emerging technology and is seen as the answer to various problems that today's autonomous driving encounters, especially in the early stages when the algorithms are still learning. The detailed use cases and requirements of ToD are currently being investigated by several initiatives, projects, and associations, including European Union's 5GMobix and 5GCoCor [49], and the 5G automobile association (5GAA) [19].

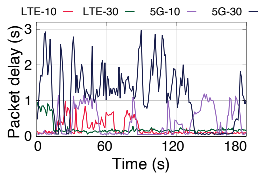
The success of ToD, however, critically relies on our ability to support high-quality, high bitrate, and real-time vehicle-to-cloud video streaming to capture the 360^o view of a vehicle in the first



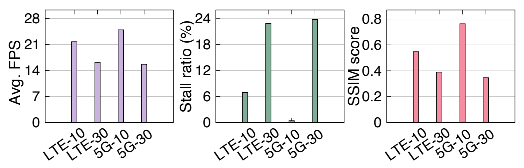
(a) RSRP and SINR fluctuation



(b) Packet loss rate



(c) Packet delay (one-way)



(d) QoE performance

Figure 3: Characteristics of a single 5G or LTE cellular link when upstreaming videos at 10 and 30Mbps from a moving vehicle.

place. 5GAA needs the support of ∼ 30Mbps video bitrate with <100ms latency. The 5GCoCor requires an uplink at 8-30Mbps and a latency of <80ms for indirect control. While ToD is crucial to make the self-driving era practical, the above requirements are visionary and are beyond the capability of today's vehicle solutions.

2.2 Challenges in vehicle-to-cloud streaming via a single cellular link

To understand the challenges, we characterized cellular links in 5G and LTE when upstreaming a real-time video to a cloud server from a moving vehicle. In the course of our measurements, we conducted experiments by streaming an RTSP video at 30fps with bitrates of 10Mbps and 30Mbps over a single cellular link. These experiments took place while driving a vehicle around a typical metropolitan area. For each second during the streaming session, we collected two types of signal metrics at the physical layer from the cellular module driver: Received Signal Received Power (RSRP) and Signal to Interference and Noise Ratio (SINR). At the transport layer, we determined the packet loss rate and packet delay per second by utilizing the appPacketID and timestamp fields. The packet loss rate was calculated as the number of received application packets (*recv_app_pkt_cnt*) divided by the range of the appPacketID field[2]. The packet delay was calculated as the difference between the received time and sent time[3]. At the application layer, we calculated the QoE metrics as defined in Appx. C. Our measurements in Fig. 3 reveal the following:

Unstable signal coverage. Fig. 3(a) shows that both the RSRP and SINR exhibited significant and random fluctuations that could vary more than 30dB within a few seconds. We also note that 5G's signal experienced more considerable fluctuations than LTE, and the SINR of 5G dropped to 0dB multiple times within 3 minutes. These observations showed that the cellular signal coverage becomes very unstable as a vehicle drives.

Heavy bursty loss and large latency spikes. Fig. 3(b) and Fig. 3(c) show the measured packet loss rate and the one-way packet delay at the transport layer, respectively. In terms of loss, both the 5G and LTE links showed a heavy bursty loss, which could be as high as 100% and last tens of seconds. Regarding the delay, both 5G and LTE suffered from many latency spikes that could go up to a few seconds. More surprisingly, due to its shrunk coverage, 5G's loss and delay can sometimes be even worse than LTE. The loss and delay also became worse as the video bitrate increased.

[2] $loss\_rate = \frac{recv\_app\_pkt\_cnt}{max\{appPacketID\}-min\{appPacketID\}}$

[3] We periodically use NTP to synchronize the clocks and connect to the NTP server by selecting the cellular interface that has the lowest RTT to NTP server to minimize the synchronization error.

Poor video QoE. Fig. 3(d) shows the corresponding video QoE measurements, including the frames per second (FPS), video stall ratio, and the SSIM score. As expected, under such a heavy bursty loss and high latency spikes, all cases suffered from reduced framerate, high video stall ratio, and low SSIM score, leading to an unsatisfactory user experience.

Remark: As a result, neither the 5G link nor the LTE link was able to support real-time streaming above 10Mbps consistently. It is worth noting that the heavy bursty loss is significantly different from the traditional notion of slight random loss in a packet erasure channel, which makes many past coding schemes ineffective. Understanding such cellular characteristics is critical to design the right transport solution and motivates us to rethink the transport layer design.

3 OVERVIEW

To address the unpredictable nature of cellular signals in vehicle-to-cloud streaming, we introduce CellFusion. The high-level end-to-end architecture of CellFusion is shown in Fig. 1(a), which is split into three domains: (1) the vehicle domain that consists of the video client (sender) and CellFusion CPE, (2) the edge domain that comprises multi-cellular networks and CellFusion edge proxy servers, and (3) the cloud domain that hosts CellFusion controller and cloud apps (receiver). Below, we describe their functionality followed by CellFusion's packet flow shown in Fig. 1(b).

3.1 Core components

CellFusion CPE: A specialized hardware installed in a vehicle to serve as the packet gateway for in-vehicle LAN to the Internet. It has four cellular modules to enable multipath transport over heterogeneous mobile carriers.

CellFusion proxy server: Proxy servers are container programs distributed at the cloud's edge PoPs (CDNs) and serve as edge access points for vehicle-to-cloud traffic.

CellFusion tunnel-client: A program that runs on top of CellFusion CPE. It sets up the multipath QUIC (MPQUIC) client. In the uplink, it is the start-point of the unreliable multipath QUIC tunnel, which drives XNC's client-end stack to perform packet encoding, loss detection, and loss recovery.

CellFusion tunnel-server: A program that runs in CellFusion proxy servers. It sets up the multipath QUIC server. In the uplink, it is the end-point of the unreliable multipath QUIC tunnel, which drives XNC's server-end stack to perform packet decoding and forwarding.

CellFusion controller: The controller is deployed on the central cloud and serves as the control and management plane of CellFusion.

3.2 CellFusion's packet flow

To understand how multipath and XNC are applied to video traffic, we walk through CellFusion's packet flow as shown in Fig. 1(b). Our explanation below focuses on the uplink flow (vehicle-to-cloud). The downlink flow is similar to the uplink but in the reverse direction.

In the vehicle domain, the *video client* (sender) sends out original IP packets destined to the *cloud app* (receiver) through the *CPE*. These packets are captured from the *CPE*'s virtual *tun* interface into the *tunnel-client* in user space and then handed to the *MPQUIC client*. The *XNC* in QUIC stack encapsulates packets in *XNC Datagram*, which extends the QUIC-Datagram. *XNC* applies network coding when loss retransmission is triggered and immediately forwards first-time-arriving packets. The outputs of the *tunnel-client* are QUIC packets whose destination IPs point to the *proxy server* and are transmitted via multi-cellular interfaces.

In the edge domain, packets received by the *proxy server* are passed to the *MPQUIC server*. The QUIC stack calls *XNC* to decode QUIC's payload and extract original IP packets. The decoded IP packets are then sent to the virtual *tun* interface and forwarded to the *cloud app*. Before a packet leaves the *proxy server*, Source-NAT is applied to it so that the return traffic from the *cloud app* will be routed to the *proxy server*.

The above flow effectively builds a transparent multipath overlay tunnel. Therefore, CellFusion doesn't require modification from either the video client or the cloud app. The IP packet between the video client and the cloud app can be end-to-end encrypted, and CellFusion won't be able to decipher its content. CellFusion further adds an extra layer of QUIC's TLS encryption and header protection mechanism, so security is never compromised.

3.3 Organization of the following sections

In the following sections, we start by introducing the design of XNC, our core network coding-based multipath transport on top of QUIC (§4). Then we describe CellFusion's CPE hardware design (§5). After that, we present CellFusion's cloud-native back-end service (§6). Finally, we discuss our current deployment and evaluations of CellFusion (§7, §8).

4 XNC DESIGN

4.1 Logical description

At a high level, XNC is designed with four major objectives:

A Supporting real-time videos at low latency.
B Attaining high data rate via efficient multipath usage.
C Overcoming bursty loss and link unpredictability.
D Low redundancy and low computational complexity.

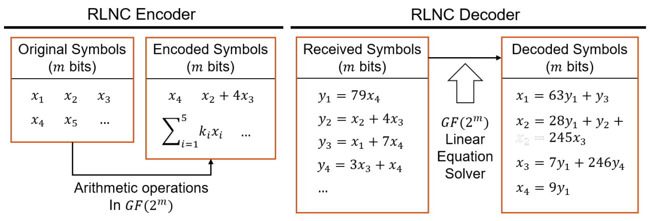
Objective *A* implies that XNC should not be fully reliable like TCP, which suffers from head-of-line blocking issues. Thus, at the base layer, XNC is built on top of QUIC-Datagram, the unreliable version of QUIC. The base layer provides essential building blocks such as encryption and congestion control. Concerning objective *B*, XNC incorporates multipath QUIC features with QUIC-Datagram. The multipath scheduling is done at a packet-level granularity to allow path aggregation for simultaneous transmission.

Given objective *C*, we further need a partial reliability mechanism to combat loss on top of the unreliable multipath QUIC. There are two basic approaches: the proactive approach, which sends feed-forward redundant packets when original packets are transmitted for the first time, and the reactive approach, which performs feedback-based loss recovery. The proactive approach does not fit for heavy bursty losses because to overcome the loss of many consecutive packets, it would need a very high redundancy rate. However, as it is nearly impossible to predict when a loss will happen and how many packets will be lost, one has to consistently apply such a high redundancy rate even when the loss rate is low. This leads to high redundancy and prohibitive computational complexity, violating objective *D*. The problem with the reactive approach, however, is that the recovery is naturally delayed due to the feedback loop. The delay can be further exacerbated if the retransmitted packets are lost (i.e., loss of loss-recovery packets).

Therefore, to use the reactive approach, we must shorten the feedback recovery loop and improve the retransmission success rate. To achieve this goal, XNC embraces the opportunity of being closer to vehicles at the cloud's edge and introduces three techniques to address this problem: quick loss detection based on QoE-aware policy, Q-RLNC for retransmission packets, and opportunistic one-shot recovery to improve the retransmission success rate. Our key insight is that when retransmission is needed, it should be triggered as early as possible and executed in one shot with maximum effort by leveraging coding protection and fully utilizing multipath resources. To translate this idea into practice, first, we deploy CellFusion's proxy at CDN PoPs to cut the round-trip time of the tunnel physically. Second, we quickly detect which packets are lost based on the video's QoE requirement (bitrate and latency requirements) to start lost packet retransmission as timely as possible. Third, we introduce RLNC to the QUIC stack for the first time, and during the retransmission, our Q-RLNC encodes each retransmitted packet as a random linear combination from a range of lost packets over QUIC. We apply Q-RLNC because of the following nice properties:

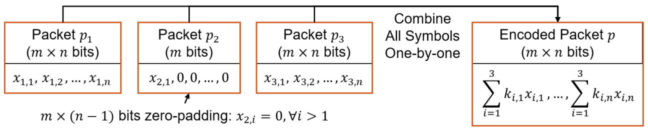
- It has a flexible coding range that is not limited to fixed length or sliding window.
- It is rateless, so XNC can encode arbitrary number of packets for any range.
- Each coded packet is equivalent to a linear equation. With enough equations received, we can recover all lost packets.
- It is designed to seamlessly work with QUIC.

Finally, we manifest the advantage of Q-RLNC with the opportunistic one-shot recovery, which sends out on each path a number of random linear equations of lost packets proportionally to that path's instantaneous available window. Such a recovery approach utilizes path diversity by opportunistically exploiting each path's available network resource and is also highly resilient to the potential loss of loss-recovery packets. Since XNC disperses each original packet's information onto all available paths, the failure of a packet on one path can be remedied when a coded packet from any of the other paths arrives. Note that the actual encoding is only performed on a range of lost packets. Our encoding is on-the-fly as we don't need to purposely accumulate packets before encoding. Moreover, XNC utilizes bandwidth resources efficiently because it achieves almost zero redundancy under good network conditions with no loss.



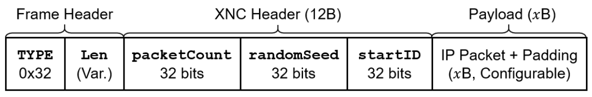
RLNC Encoder
RLNC Decoder
Original Symbols (m bits)
x_1 x_2 x_3
x_4 x_5 ...
Encoded Symbols (m bits)
x_4 $x_2 + 4x_3$
$\sum_{i=1}^{5} k_i x_i$ ...
Arithmetic operations In $GF(2^m)$
Received Symbols (m bits)
$y_1 = 79x_4$
$y_2 = x_2 + 4x_3$
$y_3 = x_1 + 7x_4$
$y_4 = 3x_3 + x_4$
...
$GF(2^m)$ Linear Equation Solver
Decoded Symbols (m bits)
$x_1 = 63y_1 + y_3$
$x_2 = 28y_1 + y_2 + 245x_3$
$x_3 = 7y_1 + 246y_4$
$x_4 = 9y_1$

Figure 4: RLNC encoding/decoding process ($m = 8$ in this example).



Packet p_1 ($m \times n$ bits)
$x_{1,1}, x_{1,2}, \ldots, x_{1,n}$
Packet p_2 (m bits)
$x_{2,1}, 0, 0, \ldots, 0$
Packet p_3 ($m \times n$ bits)
$x_{3,1}, x_{3,2}, \ldots, x_{3,n}$
Combine All Symbols One-by-one
Encoded Packet p ($m \times n$ bits)
$\sum_{i=1}^{3} k_{i,1}x_{i,1}, \ldots, \sum_{i=1}^{3} k_{i,n}x_{i,n}$
$m \times (n-1)$ bits zero-padding: $x_{2,i} = 0, \forall i > 1$

Figure 5: Applying RLNC to packets.



`Len` (Variable length): Total length of `XNC_NC` frame, in bytes.
`packetCount` (32 bits): Length of encode range.
`randomSeed` (32 bits): Random seed used to compute the coefficient of each original packet.
`startID` (32 bits): ID of the first packet in the encode range.

Figure 6: `XNC_NC` frame structure.

4.2 Base protocol

XNC uses QUIC-Datagram as its base layer protocol stack and incorporates multipath QUIC functionality on top of it. The QUIC-Datagram is an unreliable QUIC extension recently standardized by RFC9221 [43] while the IETF QUIC working group is standardizing a reliable version of multipath QUIC [41]. Once a multipath QUIC connection is negotiated, XNC extends the *Datagram Frame (type 0x30, 0x31)* to encapsulate video packets. Basing XNC upon QUIC provides several important benefits:

- Thanks to the user-space nature of QUIC, deploying and upgrading QUIC-based protocol is much easier compared to kernel-based protocol stacks (e.g., kernel TCP).
- We can reuse as much as possible the mechanism of QUIC, such as TLS 1.3 encryption, RTT and bandwidth measurements, congestion control, and PMTU discovery.
- QUIC is end-to-end encrypted, and the header of QUIC is also protected [42], which prevents it from being inspected by WAN middleboxes. Such a property is important for CellFusion's traffic to traverse middleboxes on the public Internet without being interfered.

Multipath scheduler. Our default multipath scheduler is min-RTT scheduler [30], which puts first-time sending packets on the lowest delay path with available congestion window. However, as discussed later, we do not use this scheduler in our opportunistic one-shot recovery.

Congestion controller. We use BBR [50] as the congestion controller due to its resilience to packet losses and its ability to quickly grab available bandwidth.

4.3 Coding scheme of Q-RLNC

4.3.1 *Basic RLNC operations.*

RLNC [36] performs computations in Galois field $GF(2^m)$, which is illustrated in Fig. 4. When encoding, RLNC computes a linear combination of original symbols with random coefficients to obtain an encoded symbol (m-bit integer), so each encoded symbol equals to a linear equation in $GF(2^m)$. When decoding, RLNC recovers the original symbols by solving a system of linear equations, given it has received enough linear-independent encoded symbols. To perform RLNC on packets instead of individual symbols, we treat each packet as an array of m-bit symbols, as shown in Fig. 5. Before encoding, we first ensure each original packet has an equal length by zero-padding shorter packets. Then we generate the encoded packet by separately combining symbols of the same index in each array. In XNC, we set m to 8, where the length of symbols is one byte. This value is chosen to enable SIMD acceleration of $GF(2^m)$ arithmetic (§5.2).

4.3.2 *Packet format.*

The encoded packets must carry information about the linear combination for a receiver to perform decoding. In XNC, we always apply RLNC on a range of contiguous packets so that we can encode such information in `XNC_Header` shown in Fig. 6, which consists of three 32-bit integer fields:

- `packetCount`: the count of packets in the range of contiguous original packets used to encode that packet.
- `randomSeed`: the random seed used to generate the sequence of random coefficients to encode that packet [51].
- `startID`: the ID of the first packet in the sequence of original packets used to encode that packet.

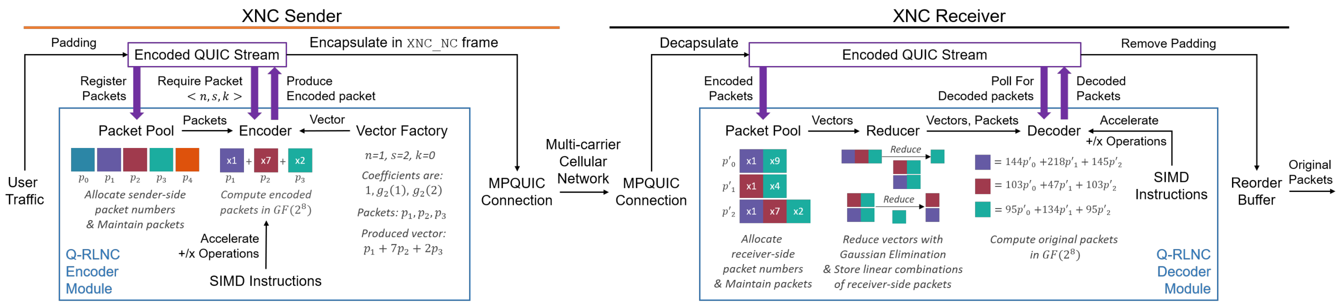
The sender and receiver need to agree on the same random coefficient, so upon connection negotiation, we initialize two identical pseudo number generators (denoted as g) at both the sender and the receiver, which derives the sequence of coefficients generated by g with seed value s as $\{g_s(1), g_s(2), \ldots\}, \forall s, i, g_s(i) \in GF(2^8) \setminus \{0\}$. Assume p_k denotes original packet with ID k, p is the coded packet, and the values of `packetCount`, `randomSeed`, and `startID` are n, s, and k, respectively, then:

$$p = \begin{cases} p_k & \text{n=1,} \\ p_k + \sum_{i=1}^{n-1} g_s(i) p_{k+i} & \text{n>1} \end{cases}$$

A special case, as shown above, is when $n = 1$. In this case, p is equal to an original packet, p_k, and s is ignored. We discuss more details about our encoding algorithm in Appx. A.

4.3.3 *Enabling RLNC in QUIC.*

XNC implements the encoder and decoder as software modules called by the QUIC stack, and we illustrate the workflow in Fig. 7. Packets' encoding and decoding are performed above QUIC's Datagram layer. Specifically, on the sender side, incoming packets are first padded and registered in the encoder module before they can be used for encoding. In this process, the encoder module saves a copy of each original packet in its packet pool and assigns a packet number and timestamp. To send a packet, the QUIC layer calls the encoder API for an encoded packet and encapsulates it into a QUIC-Datagram frame, marked by a new type: `XNC_NC` (0x32), shown in Fig. 6. On the receiver side, received payloads of `XNC_NC` frames are passed to the decoder module. Each time a payload is passed to the decoder module, XNC calls the decoder API for any possibly decoded original packets. The



XNC Sender
XNC Receiver
Padding
Encoded QUIC Stream
Encapsulate in XNC_NC frame
Register Packets
Require Packet $<n, s, k>$
Produce Encoded packet
Packet Pool
Packets
Encoder
Vector
Vector Factory
Allocate sender-side packet numbers & Maintain packets
Compute encoded packets in $GF(2^8)$
$n=1, s=2, k=0$
Coefficients are: $1, g_2(1), g_2(2)$
Packets: p_1, p_2, p_3
Produced vector: $p_1 + 7p_2 + 2p_3$
Accelerate +/x Operations
SIMD Instructions
Q-RLNC Encoder Module
User Traffic
MPQUIC Connection
Multi-carrier Cellular Network
MPQUIC Connection
Decapsulate
Encoded Packets
Poll For Decoded packets
Decoded Packets
Remove Padding
Vectors
Reducer
Vectors, Packets
Decoder
Accelerate +/x Operations
SIMD Instructions
Allocate receiver-side packet numbers & Maintain packets
Reduce vectors with Gaussian Elimination & Store linear combinations of receiver-side packets
Compute original packets in $GF(2^8)$
$= 144p'_0 + 218p'_1 + 145p'_2$
$= 103p'_0 + 47p'_1 + 103p'_2$
$= 95p'_0 + 134p'_1 + 95p'_2$
Q-RLNC Decoder Module
Reorder Buffer
Original Packets

Figure 7: XNC workflow: XNC implements the encoder and decoder as software modules called by the QUIC stack.

decoder obtains equations (vectors) from its packet pool, reduces vectors via Gaussian elimination, and stores reduced results. Once a new packet is decoded, the decoder hands it to the QUIC stack.

4.4 QoE-aware loss detection

XNC replaces the QUIC's default loss detection policy with a QoE-aware loss detection policy to accelerate the recovery process discussed above. The rationale behind this is that a real-time video frame is only useful if it is delivered before an application's deadline, so packets with a high risk of passing the deadline should be retransmitted immediately.

4.4.1 Set threshold for loss detection. XNC applies a new loss detection threshold, which is the smaller value of an application-defined time threshold and the QUIC's PTO [52], and thus, any packet that is not acknowledged after this threshold is marked as "lost" and added to the retransmission queue. The application-defined time threshold is adjusted based on the end-to-end latency need of the application. Note that our modified loss detection makes XNC's loss recovery more aggressive than legacy QUIC. However, the congestion window helps limit XNC's total number of in-flight packets and preserve its friendliness towards other users.

4.4.2 Determine encoding range borders among lost packets. XNC further divides lost packets in the retransmission queue into contiguous ranges, on which we apply Q-RLNC individually. Each range spans a continuous sequence of packets. Q-RLNC generates ranges according to the requirements of video streaming and computation cost. First, lost packets that belong to the same video frame should be recovered at the same time. To use bandwidth more efficiently, XNC should either recover a complete frame or not retransmit any packets in that frame; Second, a too-big range is also undesirable, as it may introduce additional delay in the encoding process and increase computational complexity. In XNC, we check for three conditions to insert the range border after the most-recently-sent packet accordingly.

- Current range contains at least r packets.
- Current range spans at least t seconds.
- If a video frame border is detected, e.g., an RTP header with extension marking.

Because XNC is transparent to the application's traffic and may not always detect frame boundaries in encrypted user traffic, the third condition is set as an option. In practice, for a 30Mbps video session, we set r to 10 and t to 60ms.

4.4.3 Expiration of ranges and lost packets. Unlike fully reliable transport that tracks lost packets until successful delivery, XNC only tracks lost packets for a given period because recovery of expired video packets would not contribute to video QoE but delay the delivery of newer packets instead. In XNC, we use a configurable application-dependent time threshold, t_{expire}, to determine whether a packet is expired. If the last packet in an encode range is expired, the range is expired. In this work, we empirically set t_{expire} to 700ms.

4.5 Opportunistic one-shot recovery

With the coding scheme and range borders defined, we now specify the behavior of XNC's opportunistic one-shot loss recovery, whose aim is to exploit all paths' instantaneous available bandwidth to maximize recovery rate in one shot.

4.5.1 Compute the minimum number of coded packets n'. Assuming n packets are detected as lost in a range, XNC first computes the number of coded packets n' needed by the XNC receiver to recover that n lost packets. We note that $n' \neq n$. Optimally, we may deliver exactly n *linearly independent* coded packets to decode all n original packets. However, due to the fact that XNC encodes packets with random coefficients, there is no absolute guarantee of the linear independence of *every* subset of generated packets. Thus, we must deliver $n' > n$ packets. Fortunately, we only need a small number of extra packets in order to obtain n independent packets under our random coefficient sampling scheme. Theoretically, we could prove that:

THEOREM 4.1. *If $n' = n+k$ coded packets are successfully delivered, the probability that XNC performs a successful decode is at least* $1 - \frac{1}{255^k \times 254}$.

The proof can be found in Appx. B. Based on the results above, we let $n' = n+3$ if $n > 1$. The value is chosen to balance bandwidth overhead and decode probability. When $n = 1$, the encoding range only contains one packet, and thus, the receiver does not need to decode, so we let $n' = 1$.

4.5.2 One-shot recovery. After obtaining n', XNC sums up all the available congestion windows on each usable path, denoted as b. If $b < n'$, the recovery operation is delayed. XNC may perform recovery anytime later, before the expiration of the lost packets (§4.4.3). In other words, we do not transmit recovery packets to waste bandwidth when we know the network bandwidth surely cannot meet our needs. If there are enough available congestion windows (i.e., $b \geq n'$), XNC distributes up to b encoded packets onto all available paths (proportional to each path's available window size). We further limit the number of packets sent on each path to be smaller than $\rho \times n'$, where $1 < \rho < 1.2$. When $n = 1$, we simply send one packet on each usable path to minimize the delay. After that, XNC forgets any lost packets involved in the recovery operation.

5 CELLFUSION'S CPE

5.1 Customized hardware design

CellFusion's in-vehicle CPE box is a specially designed hardware that enables high-performance communication over four cellular networks simultaneously, as shown in Fig. 2(a) and Fig. 2(b). The CPE consists of four subsystems: a CPU subsystem, a cellular networking subsystem, an interface and power management subsystem, and a Wifi/LAN subsystem. For the CPU subsystem, we choose Rockchip RK3399 [53] as the core CPU, which has dual Cortex-A72 quad Cortex-A53 cores. The core CPU also supports SIMD instructions that are critical to accelerating XNC operations. The cellular networking subsystem has 2 Quectel RM500Q-GL [54] as the 5G modules and 2 Quectel EP06-E [55] as the LTE modules. We use MIMO to improve cellular performance further: the 5G module has 2TX/4RX antennas, and the LTE module has 1TX/2RX antennas. The Wifi/LAN subsystem is used for connecting video streaming sources inside the vehicle. The peak power consumption of the CPE is less than 50W, and the standby power is less than 25W.

5.2 Acceleration of XNC

In order to achieve high bitrate video transmission, it is crucial to perform network coding as efficiently as possible. Recall that in XNC, each packet is treated as an array of m-bit symbols, and an encoded packet is generated by separately combining symbols of the same index in each array, which can be computed in parallel (§4.3.1). We perform RLNC with ARM NEON advanced SIMD[4] instructions [56] to leverage such parallelism. In XNC, we implement 8-way $GF(2^8)$ multiplication with the help of `vmull_p8` (8-bit polynomial multiplication) NEON intrinsic. Moreover, addition in $GF(2^8)$ is simply bit-level XOR and could be automatically vectorized by the compiler. With SIMD, we support 30Mbps video streaming at < 20% CPU usage on our CPE box.

6 CELLFUSION'S BACK-END SERVICE

6.1 Cloud-native architecture

We design CellFusion's back-end with a cloud-native architecture that is simple to deploy and manage. The back-end consists of two parts: CellFusion proxy servers and CellFusion controller.

Proxy servers. To reduce the access latency, CellFusion proxy servers are deployed at the cloud's edge (i.e., CDNs). The user-space nature of QUIC allows us to deploy them as containers running at CDN PoPs. Containerization makes it easy to autoscale these proxy servers to meet the change in demand, offering good scalability and cost-effectiveness.

Controller. CellFusion's controller implements the control plane and is deployed in the central cloud. It serves five main functions:

[4]SIMD uses a single instruction to perform the same operation in parallel on multiple data elements of the same type and size.

(1) It authenticates CPE devices so that only legal users are allowed to access the service. (2) It manages configurations for both CellFusion's CPEs and proxy servers, which obtain necessary configuration parameters before setting up the multipath tunnel and XNC. (3) It is responsible for achieving high availability. The controller continuously monitors each proxy server's health states and load and performs fail-over when needed. (4) It acts as an orchestrator that instructs a CPE to several servers that it should connect to based on the servers' availability and load. The CPE measures network delay to these servers and chooses the one with minimal delay.

6.2 Multi-tenant

Each proxy server is designed to serve many users to reduce deployment costs. Thus, CellFusion *tunnel-server* implements multi-tenant support. When multiple vehicles are connecting to the same proxy server, the controller will allocate a unique private IP address for their CPEs' virtual *tun* interface to perform Source-NAT on each packet that goes through it. In doing so, video packets from the same CPE will have the same source IP address, and the server will see different source addresses (after decapsulating outer headers) in packets from other CPEs. With this unique information, we build a mapping table that associates a CPE's address with its *tunnel-client*'s QUIC connection ID (CID) on the server. In this way, when the server receives a packet in the return direction, it can find the corresponding CID via this packet's destination address and send this packet to the correct vehicle via the QUIC connection. In other words, in the packet flow, we actually employ NAT two times, the first time at the CPE's *tun* interface and the second time at the proxy server's public network interface.

7 IMPLEMENTATION AND TESTING IN THE WILD

Implementation details. CellFusion's base protocol stack is based on RFC9000 [57], on top of which we implement unreliable QUIC-Datagram based on RFC9221 [43] and incorporate multipath features from IETF WG Draft [41]. We implement XNC as software modules that are called by the QUIC stack. The entire transport is written in C language with 30K LoC that is portable to different platforms with minor modifications. Our *tunnel-client* is running in OpenWRT [58] on the CPE, and our *tunnel-server* is running in CentOS containers. Our controller is an HTTPs server built on the Java Spring framework. One practical issue is the MTU problem, which we discuss in Appx. E.

Testing in the wild. CellFusion has been tested on 100 self-driving vehicles for over six months. A photo of such a deployment is shown in Fig. 2(c). These test vehicles operate daily in several metropolitan areas. The coverage and proximity of CellFusion edge access are key to ensuring low latency and service availability. We run proxy servers on 50 Alibaba Cloud's CDN PoPs across three states. The controller is deployed in the central cloud that serves both the CPEs and the proxy servers.

8 EVALUATION

In this section, we present the evaluations of CellFusion, which consists of three parts:

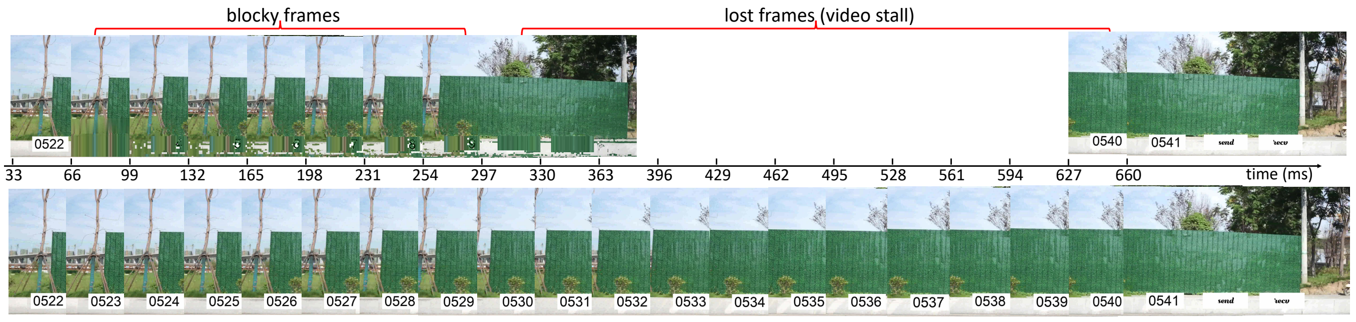


Figure 8: An example of the received streaming videos from the end-to-end road test: Multipath QUIC (top) and CellFusion (bottom).

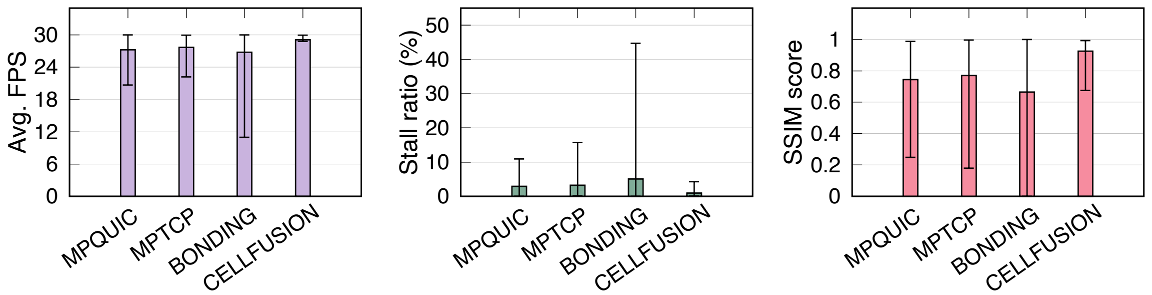


Figure 9: Measured video QoE metrics of Multipath QUIC, MPTCP, BONDING, and CellFusion in end-to-end road tests.

- **End-to-end road test:** Firstly, we present the results of end-to-end road tests, in which we upstreamed real-time high-bitrate videos from a moving vehicle to a cloud server through CellFusion. The streamed video was a specially made reference video that allowed us to extract video QoE metrics. We collected results from over 5000km of driving.
- **Statistical results from deployment:** Then, we present statistical results from the deployment of CellFusion on 100 self-driving vehicles for over six months. These vehicles were running in metropolitan areas routinely to test self-driving algorithms.
- **Benchmarks and ablation studies of XNC:** Finally, we further implemented a number of state-of-the-art multipath solutions and coding schemes, and ran experiments in controlled environments to compare with XNC. We also performed ablation studies to validate the design point of individual components in XNC.

General setup. Unless otherwise stated, in the road test, benchmarks, and ablation studies, we used ffmpeg [59] and UDP-based RTSP protocol for upstreaming a real-time video. The streaming bitrate was 30Mbps at 30fps.

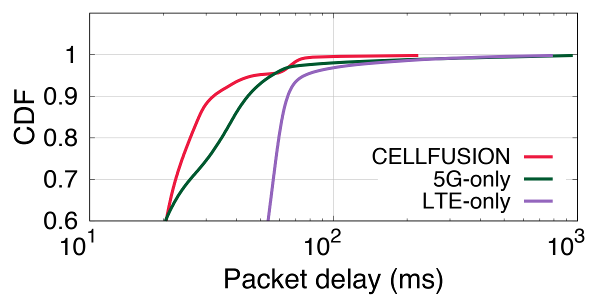
8.1 End-to-end road tests

8.1.1 Evaluation methods. In this test, our testers drove vehicles equipped with CellFusion's CPE in metropolitan areas while upstreaming a reference video through CellFusion's proxy services deployed in the CDN PoPs to an RTSP server in the cloud. The reference video was a specially made video clip, in which we assigned a sequence number stamp to each frame, so we could later analyze received videos against the original one to extract video QoE metrics such as video stall, framerate, and normalized SSIM score. We further discuss how video analytics is done with a tool we developed in Appx. C. Our testers performed testing on different roads, and the total testing distance was over 5000km.

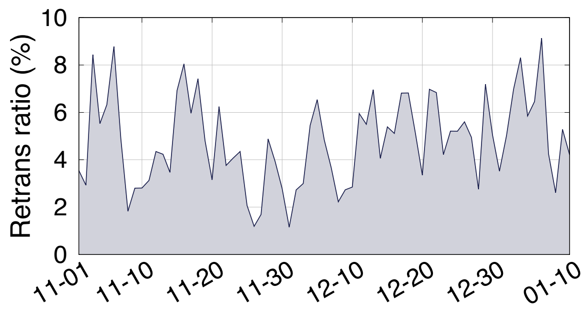
8.1.2 Other solutions used in the comparison. In this test, we also compare CellFusion against MPQUIC, MPTCP, and cellular bonding (BONDING). Note that BONDING performed load-balance on a video stream via hashing its 5-tuple to a cellular interface, so it did not use a proxy server. To compare different solutions fairly, we streamed videos through different CPEs to the RTSP server at the same time, such that evaluations were conducted under the same network conditions.

8.1.3 Video trace samples. We start by illustrating a sampled pair of received video traces in Fig. 8. The video trace on the top was from MPQUIC, while the video trace at the bottom was from CellFusion. The two video traces were aligned based on the sequence numbers stamp of video frames. We counted the video frame number per second to compute the framerate, and we further analyzed frame intervals to calculate the stall ratio. The alignment enabled us to measure the SSIM score by comparing a received video frame to a reference video frame with the same sequence number. As shown in Fig. 8, the trace from MPQUIC experienced blockiness and severe video stall, while the video from CellFusion was much clear and smooth, indicating much better QoE.

8.1.4 QoE metrics. Next, we show the overall QoE metrics, including framerate, stall ratio, and normalized SSIM score of the end-to-end road test in Fig 9. CellFusion achieved the highest framerate, the lowest stall ratio, and the most significant SSIM score among all solutions under test. It is worth noting that CellFusion's performance variation was also much smaller than others. BONDING exhibited the largest variations because it could not aggregate multiple links. Even though MPQUIC and MPTCP aggregated multiple



(a) CDF of video packet delays



(b) Traffic cost (from Nov.1st, 2022 to Jan.10th, 2023)

Figure 10: Results collected from deployed vehicles. Video packet delays were collected from vehicles that ran with CellFusion, LTE-only, and 5G-only. Traffic redundancy was collected from a vehicle's trace log.

links, they could not handle fragile cellular links as a vehicle drove. In contrast, CellFusion efficiently fused multiple cellular networks and overcame the challenge of heavy bursty loss and channel unpredictability. As a result, CellFusion achieved an average framerate, video stall ratio, and normalized SSIM score of 29.11fps, 0.99%, and 0.93, respectively. And CellFusion outperformed the comparative solutions on all QoE metrics. For example, compared to MPQUIC, MPTCP, and BONDING, CellFusion reduced the average stall ratio by 66.11%, 69.35%, and 80.62%, respectively.

8.2 Statistical results from deployment

8.2.1 Evaluation methods. In this part, we report the statistical results from a six-month testing of CellFusion in 100 self-driving vehicles. CellFusion was routinely used to enable remote control and intervention. The Collection of online metrics faces two challenges: (1) In the actual use, vehicles upstreamed real-time camera views instead of a reference video; (2) Storing $7 \times 24h$ videos is costly. Therefore, instead of using video QoEs, we report two metrics constantly logged in online environment: video packet delay and traffic redundancy.

8.2.2 Video packet delay. Fig. 10(a) shows the CDF of video packet delay when streaming with CellFusion vs. when streaming with LTE-only and 5G-only. We observed that CellFusion significantly reduced packet delay, especially at the tail. With CellFusion, the 95th, 99th, 99.9th percentile delays were 47.4ms, 73.8ms, and 222.3ms respectively. In contrast, the corresponding delays of LTE-only were 76.1ms, 267.2ms, and 791.9ms, while the delays of 5G-only were 55.8ms, 259.2ms, and 954.7ms. Compared to 5G-only, CellFusion reduced tail delay by 15.05%, 71.53%, and 76.72% at the 95th, 99th, and 99.9th percentile, respectively.

8.2.3 Traffic redundancy. Fig. 10(b) shows the traffic redundancy log trace of CellFusion deployed in a vehicle from Nov.1st, 2022 to Jan.10th, 2023. The daily redundancy cost varied between 1% to 9%. The variation stemmed from the fact that vehicles moved to different locations that had different network conditions. Because XNC only applied network coding to loss recovery, redundant traffic was only needed under poor network conditions. Therefore, CellFusion was highly cost-effective to use online.

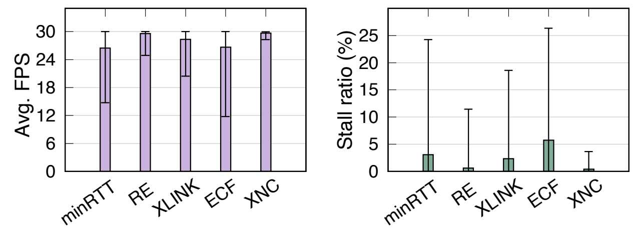
8.3 Benchmarks and ablation studies

We further performed a comparative study of XNC with several past multipath optimizations and coding schemes and an ablation study of XNC to validate our design points. We performed both studies in a controlled environment to ensure testing under the same network conditions.

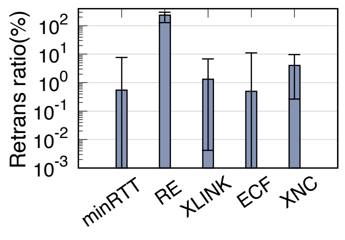
8.3.1 Controlled experiment. We built our controlled testing environment with a trace-driven network emulator *mpshell* from Mahimahi [60]. We extended this tool to support 4-path emulation. The traces in use were collected from a moving vehicle on different roads (more details are listed in Appx. D). Each result was gathered from an experiment with 100 traces. Our controlled testbed consisted of two servers in the same local area network. On the first server, we ran the *tunnel-client* and *tunnel-server*, which were connected through the *mpshell* emulator. The first server also hosted the *video-client* (using ffmpeg), which streamed video to *tunnel-client*. On the second server, we deployed an *RTSP server* that received stream traffic from the *tunnel-server*, and a *stream receiver* (using ffmpeg) that received streams from the *RTSP server*. At the beginning of each experiment, we started the emulator's shell with an assigned trace, then ran *tunnel-client* and *video-client* inside this shell to upstream a reference video.

8.3.2 Other solutions used in the comparison. We compared XNC with two categories of existing solutions: *multipath scheduling optimizations* and *multipath & network coding solutions*. In the first category, we chose four state-of-the-art schedulers that can support 4 paths (i.e., the scheduling algorithm is scalable): minRTT scheduler [30], fully redundant scheduler (RE) [61], XLINK [29] and ECF [62]. We note that there were other solutions in this category that were limited to 2 paths (e.g., Musher [63] and STMS [64]), and thus they were not chosen in this study. In the second category, we chose Pluribus [26], which practically designed network coding in multipath transport for web traffic.

8.3.3 Benchmark results vs. other multipath optimizations. We first compare XNC with past multipath scheduling optimizations. We show the QoE metrics in Fig. 11(a) and the redundant traffic cost in Fig. 11(b). XNC achieved the highest framerate, the lowest stall ratio, and the highest SSIM score among all solutions under this test. XNC's performance variation was also much smaller than others. In particular, XNC reduced the average stall ratio by 86.56%, 82.22%, and 92.75% against minRTT, XLINK, and ECF, respectively.

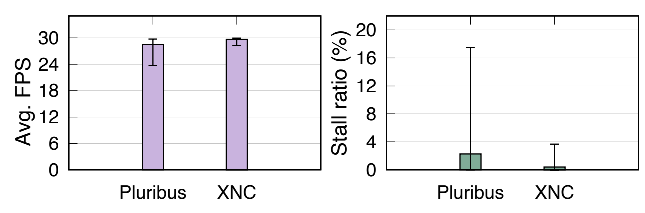


(a) QoE performance

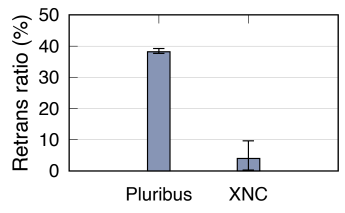


(b) Traffic cost

Figure 11: Compare XNC with state-of-the-art multipath scheduling optimizations.

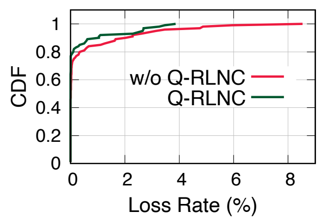


(a) QoE performance

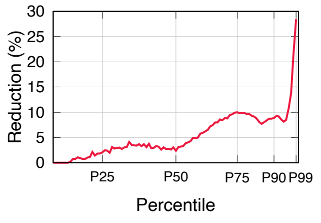


(b) Traffic cost

Figure 12: Compare XNC with a network coding based solution (Pluribus).



(a) Impact on loss rate by Q-RLNC



(b) Packet delay reduction by QoE-aware loss detection

Figure 13: Ablation studies on Q-RLNC and QoE-aware loss detection.

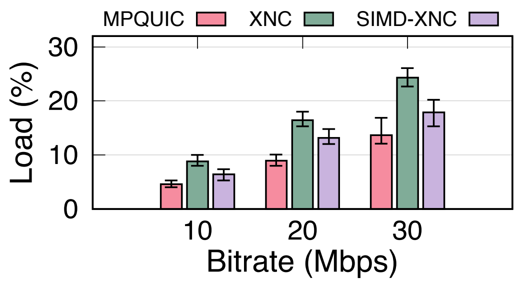


Figure 14: Average CPU load of MPQUIC, XNC and SIMD-XNC on different streaming bitrates.

Although RE had a reasonable average stall ratio, it required exceptionally high redundant traffic costs (up to 300%). As a result, when network link bandwidth was constrained, RE could not effectively use bandwidth for transmission, so at the tail distribution, its stall ratio was much higher than XNC. In contrast, XNC efficiently fuses multiple network links with a low redundancy rate (<10%), which helps it achieve a low stall ratio even at the tail.

8.3.4 Benchmark results vs. Pluribus. We further present the results comparing XNC to Pluribus. Fig. 12(a) shows the QoE metrics. We observed that XNC outperformed Pluribus in all the framerate, the stall ratio, and the SSIM score. For example, XNC reduced the average video stall by more than 81.67% compared to Pluribus. Meanwhile, as shown in Fig. 12(b), XNC also used 89.49% less redundant traffic. The above results reveal that video-to-cloud streaming is a much more challenging task than traditional web and file transfer-aware solutions, thus, calling for new innovations.

8.3.5 Ablation study. We performed an ablation study, in which we studied the impact of Q-RLNC and our QoE-aware loss detection modules. Fig. 13(a) shows the loss rate with and without XNC's Q-RLNC in which we retransmitted original packets instead of coding packets. Our Q-RLNC module significantly reduced the loss rate at tail distribution by improving loss-recovery probability, and the reduction was 15.55% and 41.70% at the 95th and 99th percentile, respectively. Fig. 13(b) shows the average packet delay reduction with QoE-aware loss detection vs. without QoE-aware loss detection at different percentiles. QoE-aware loss detection played a crucial role in reducing packet recovery time. The improvement was 8.48% and 28.44% at the 95th and 99th percentile, respectively.

8.3.6 CPU costs of XNC. The CPU costs of streaming videos through MPQUIC without network coding, XNC, and SSIM accelerated XNC (SIMD-XNC) at different bitrates are shown in Fig. 14. At 30Mbps, XNC consumed 43.77% more CPU than MPQUIC on average. With the help of hardware acceleration, SIMD-XNC consumed 23.44% more CPU than MPQUIC on average and reduced CPU usage by 26.56% from XNC.

9 RELATED WORK

Multipath transport. Multipath transport [40] provides the ability of a single connection to use multiple paths simultaneously. However, the adoption of kernel-based multipath transport such as MPTCP [40] and MPUDP [65] has been slow due to the need for OS-level support and middlebox ossification [30]. To address this problem, there has been growing interest in introducing multipath features to QUIC, but recent proposals like IETF Multipath QUIC [41], XLINK [29], and PQUIC [66] are designed as fully reliable transport for non-real-time traffic. In contrast, CellFusion is designed to meet the challenges of real-time applications.

Network coding for multipath. Network coding improves transmission reliability over lossy channels by mixing data across time and flows [36], and has received much attention in the past [32, 33]. However, there are few practical implementations and empirical studies to extend network coding to multipath. FMTCP[67], ADMIT [68], and SC-MTCP [69] explored the idea of applying FEC to multipath, but are restricted to either simulation or emulation. Pluribus [26] experimented with evolution code with two paths on a corporate bus but was limited to small non-real-time load (<86KB) and limited link rate (<1.5Mbps). CellFusion differs from past works in that: (1) It addresses the unique challenges of high-quality real-time video streaming under high mobility, which is a more complex problem. (2) It has been fully implemented and deployed at a large scale in the wild.

Cellular bonding. Several SD-WAN solutions [70–72] implement cellular bonding capability, which is sometimes confused with multipath transport. OpenWRT's mwan3 [73] also implements this capability. Cellular bonding works by load-balancing sessions across different interfaces, which is helpful in case of link failover. However, it lacks transport support for a single link to use multiple paths simultaneously and cannot adapt to link changes at packet-level granularity. In contrast, CellFusion is a multipath transport solution. CellFusion can support a much higher rate and is much more resilient to weak cellular links.

Mobile video streaming. Recently, various approaches including bitrate selection [17, 74–76], video pre-processing [77, 78], QoE-aware video re-encoding [79, 80], CDN placement optimization [81], using feedback from wireless APs [82] and server-side super-resolution [83] have been used for performance enhancement of video streaming applications. Different from these approaches, CellFusion optimizes the underlying network performance without modifying end-host applications. Our approach is orthogonal to, and could be jointly used with previous works.

10 DISCUSSIONS AND LIMITATIONS

CellFusion demonstrates new possibilities to bring high-quality real-time video streaming to vehicle-to-cloud communication. However, our current prototype still has few limitations that are left for future work.

Server migration. CellFusion establishes a connection between the CPE and an edge server to minimize communication delays. The server maintains its stability after the initialization process. However, when a vehicle has to move over a larger area, the need for server migration arises. Currently, RFC9000 [42] does not accommodate server migration, but it does allow for extensions to incorporate this feature in the future.

Venturing beyond cellular connectivity. Our system builds upon multiple cellular links. However, CellFusion's network coding based multi-path connectivity approach might not be confined to cellular connectivity alone and could potentially accommodate other wireless technologies, such as satellite communication. This could possibly extend its usefulness, especially in areas where cellular infrastructure is sparse. But of course, further investigation and development are needed to fully realize this potential.

11 CONCLUSION

In this paper, we introduce CellFusion, a system that enables high-quality, real-time video streaming from vehicles to the cloud. CellFusion adeptly integrates multiple heterogeneous cellular networks into a unified cloud connection through an innovative software-hardware design that marries multipath transport with network coding.

We put CellFusion to the test in a real-world scenario: over a six-month period, CellFusion was deployed on 100 self-driving vehicles. Our back-end infrastructure was maintained across 50 CDN Points of Presence (PoPs) spread over three states, and we conducted extensive road tests covering over 5000km. Our results demonstrate that CellFusion offers significant improvements over current technologies. We observed a 71.53% reduction in video packet delay at the 99th percentile when compared to 5G. Furthermore, when streaming at 30Mbps, CellFusion achieved a reduction in video stall ratio ranging from 66.11% to 80.62% compared to leading multipath transport solutions, and this was accomplished with less than 10% traffic redundancy.

We believe that CellFusion holds great potential for widespread deployment by Cloud and CDN providers. Offering connectivity-as-a-service on a universal scale, CellFusion can provide the robust streaming infrastructure necessary to shape the future of mobility.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and feedback. We thank all teams at Alibaba Group that help develop and test CellFusion, especially Ming Zhang, Xin Wu, Haitao Zhou, Shiye Liu, Taoping Li, Jingyi Xie, Renshan Li and Zizhao Lei. This work is supported by Alibaba Group through Alibaba Research Intern Program, National Key Research and Development Plan, China (Grant No. 2020YFB1710900) and National Natural Science Foundation of China (Grant No. 62022005 and 62172008).

REFERENCES

[1] Future of Driving. https://www.tesla.com/autopilot, 2022.

[2] Waymo. https://waymo.com/, 2022.

[3] Taycan. https://www.porsche.com/usa/models/taycan/taycan-models/taycan/, 2022.

[4] bZ4X. https://www.toyota.com/electrified/, 2022.

[5] The Washington Post. Behind the wheel of a Tesla Model 3: It's a giant iPhone — for better and worse. https://www.washingtonpost.com/technology/2018/08/02/behind-wheel-tesla-model-its-giant-iphone-better-worse/, 2022.

[6] Motor Trend. Tech Company Testing Remote Operators as Self-Driving Car Backups. https://www.motortrend.com/news/mira-self-driving-car-remote-control-car/„ 2022.

[7] Oussama El Marai and Tarik Taleb. Smooth and low latency video streaming for autonomous cars during handover. *Ieee Network*, 34(6):302–309, 2020.

[8] Forbes. Whether Those Endless Edge Or Corner Cases Are The Long-Tail Doom For AI Self-Driving Cars. https://www.forbes.com/sites/lanceeliot/2021/07/13/whether-those-endless-edge-or-corner-cases-are-the-long-tail-doom-for-ai-self-driving-cars/?sh=595cfeaf5933„ 2021.

[9] Sotiris Pavlopoulos, Efthyvoulos Kyriacou, Alexandros Berler, Spiridon Dembeyiotis, and Dimitris Koutsouris. A novel emergency telemedicine system based on wireless communication technology-ambulance. *IEEE Transactions on information technology in biomedicine*, 2(4):261–267, 1998.

[10] Ericsson. The 5G Connected Ambulance g . https://www.ericsson.com/en/cases/2020/the-5g-connected-ambulance„ 2023.

[11] Waveform. What causes weak cell phone signal and dropped calls? . https://www.waveform.com/pages/causes-of-weak-signal„ 2022.

[12] Tao Jiang, Jianhua Zhang, Pan Tang, Lei Tian, Yi Zheng, Jianwu Dou, Henrik Asplund, Leszek Raschkowski, Raffaele D'Errico, and Tommi Jämsä. 3gpp standardized 5g channel model for iiot scenarios: A survey. *IEEE Internet of Things Journal*, 8(11):8799–8815, 2021.

[13] Christoph F Mecklenbrauker, Andreas F Molisch, Johan Karedal, Fredrik Tufvesson, Alexander Paier, Laura Bernadó, Thomas Zemen, Oliver Klemp, and Nicolai Czink. Vehicular channel characterization and its implications for wireless system design and performance. *Proceedings of the IEEE*, 99(7):1189–1212, 2011.

[14] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 479–494, 2020.

[15] Moinak Ghoshal, Z Jonny Kong, Qiang Xu, Zixiao Lu, Shivang Aggarwal, Imran Khan, Yuanjie Li, Y Charlie Hu, and Dimitrios Koutsonikolas. An in-depth study of uplink performance of 5g mmwave networks. In *Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, pages 29–35, 2022.

[16] Mega Meeting. What Internet Speed do you Need for Video Conferencing? . https://www.megameeting.com/news/what-internet-speed-for-video-conferencing/„ 2021.

[17] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. Gso-simulcast: global stream orchestration in simulcast video conferencing systems. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 826–839, 2022.

[18] Electrek. Here's what Tesla's Autopilot 2.0 can see with its 8 cameras . https://electrek.co/2017/05/16/tesla-autopilot-2-0-can-see/„ 2017.

[19] 5GAA. Teleoperated driving (ToD): System requirements analysis and architecture. https://5gaa.org/content/uploads/2021/09/5GAA_ToD_System_Requirements_Architecture_TR.pdf„ 2021.

[20] Simple guide of camera bitrate setting g . https://www.unifore.net/ip-video-surveillance/simple-guide-of-ip-camera-bitrate-setting.html„ 2015.

[21] Tesla. Tesla Premium Connectivity. https://www.tesla.com/en_eu/support/connectivity„ 2022.

[22] ATT in-car Wifi g . https://www.att.com/plans/in-car-wifi/„ 2023.

[23] Verizon connected car . https://www.verizon.com/plans/devices/connected-cars/„ 2023.

[24] Tesla LTE Connection is Unusable . https://teslamotorsclub.com/tmc/threads/tesla-lte-connection-is-unusable.84139/„ 2023.

[25] 4G/LTE speeds in NZ - painfully slow . https://teslamotorsclub.com/tmc/threads/4g-lte-speeds-in-nz-painfully-slow.283812/„ 2022.

[26] Ratul Mahajan, Jitu Padhye, Sharad Agarwal, and Brian Zill. High performance vehicular connectivity using opportunistic erasure coding. In *USENIX Annual Technical Conference*, 2012.

[27] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: Vehicular content delivery using wifi. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 199–210, 2008.

[28] Ning Lu, Nan Cheng, Ning Zhang, Xuemin Shen, and Jon W Mark. Connected vehicles: Solutions and challenges. *IEEE internet of things journal*, 1(4):289–299, 2014.

[29] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 418–432, 2021.

[30] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath {TCP}. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 399–412, 2012.

[31] HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. Raven: Improving interactive latency for the connected car. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, 2018.

[32] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 243–254, 2006.

[33] Sachin Katti, Shyamnath Gollakota, and Dina Katabi. Embracing wireless interference: Analog network coding. *ACM SIGCOMM Computer Communication Review*, 37(4):397–408, 2007.

[34] François Michel, Quentin De Coninck, and Olivier Bonaventure. Quic-fec: Bringing the benefits of forward erasure correction to quic. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[35] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on information theory*, 52(10):4413–4430, 2006.

[36] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and Joao Barros. Network coding meets tcp: Theory and implementation. *Proceedings of the IEEE*, 99(3):490–512, 2011.

[37] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[38] Sreekrishna Pandi, Frank Gabriel, Juan A Cabrera, Simon Wunderlich, Martin Reisslein, and Frank HP Fitzek. Pace: Redundancy engineering in rlnc for low-latency communication. *IEEE Access*, 5:20477–20493, 2017.

[39] Alexander E Mohr, Eve A Riskin, and Richard E Ladner. Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction. *IEEE journal on selected areas in communications*, 18(6):819–828, 2000.

[40] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[41] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-03, Internet Engineering Task Force, October 2022. Work in Progress.

[42] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[43] Tommy Pauly, Eric Kinnear, and David Schinazi. An Unreliable Datagram Extension to QUIC. RFC 9221, March 2022.

[44] Michele Luglio, M Yahya Sanadidi, Mario Gerla, and James Stepanek. On-board satellite" split tcp" proxy. *IEEE Journal on Selected Areas in Communications*, 22(2):362–370, 2004.

[45] Statista. Size of the global autonomous vehicle market in 2021 and 2022, with a forecast through 2030. https://www.statista.com/statistics/1224515/av-market-size-worldwide-forecast/„ 2023.

[46] CNN. Self-driving cars were supposed to take over the road. What happened? https://www.cnn.com/2022/11/01/business/self-driving-industry-ctrp/index.html„ 2022.

[47] Forbes. Whether Those Endless Edge Or Corner Cases Are The Long-Tail Doom For AI Self-Driving Cars. https://www.forbes.com/sites/lanceeliot/2021/07/13/whether-those-endless-edge-or-corner-cases-are-the-long-tail-doom-for-ai-self-driving-cars/„ 2021.

[48] Motortrend. Tech Company Testing Remote Operators as Self-Driving Car Backups. https://www.motortrend.com/news/mira-self-driving-car-remote-control-car/„ 2022.

[49] EU 5G-PPP. 5G Trials for Cooperative, Connected and Automated Mobility along European 5G Cross-Border Corridors - Challenges and Opportunities. https://5g-ppp.eu/wp-content/uploads/2020/10/5G-for-CCAM-in-Cross-Border-Corridors_5G-PPP-White-Paper-Final2.pdf„ 2018.

[50] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[51] Nikolaos Thomos and Pascal Frossard. Toward one symbol network coding vectors. *IEEE Communications letters*, 16(11):1860–1863, 2012.

[52] Jana Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. RFC 9002, May 2021.

[53] Rockchip. Rockchip RK3399 . https://www.rock-chips.com/a/en/products/RK33_Series/2016/0419/758.html„ 2016.

[54] Quectel. Quectel RM500Q-GL Specification. https://www.quectel.com/wp-content/uploads/2021/03/Quectel_RM500Q-GL_5G_Specification_V1.3.pdf,, 2021.
[55] Quectel. Quectel EP06-E Specification. https://www.quectel.com/wp-content/uploads/2021/03/Quectel_EP06_Series_LTE-A_Specification_V1.8.pdf,, 2021.
[56] Arm Neon. https://www.arm.com/technologies/neon, 2023.
[57] QUIC IETF working group. https://datatracker.ietf.org/wg/quic/about/, 2020.
[58] OpenWrt. OpenWrt Project) . https://openwrt.org/,, 2023.
[59] FFmpeg. A complete, cross-platform solution to record, convert and stream audio and video.) . http://ffmpeg.org/,, 2023.
[60] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for {HTTP}. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 417–429, 2015.
[61] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 159–170, 2013.
[62] Yeon-sup Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, 2017.
[63] Swetank Kumar Saha, Shivang Aggarwal, Rohan Pathak, Dimitrios Koutsonikolas, and Joerg Widmer. Musher: An agile multipath-tcp scheduler for dual-band 802.11ad/ac wireless lans. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, 2019.
[64] Hang Shi, Yong Cui, Xin Wang, Yuming Hu, Minglong Dai, Fanzhao Wang, and Kai Zheng. STMS: Improving MPTCP throughput under heterogeneous networks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018.
[65] Daniel Lukaszewski and Geoffrey Xie. Multipath transport for virtual private networks. In *10th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 17)*, 2017.
[66] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. Pluginizing quic. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, 2019.
[67] Yong Cui, Lian Wang, Xin Wang, Hongyi Wang, and Yining Wang. Fmtcp: A fountain code-based multipath transmission control protocol. *IEEE/ACM Transactions on Networking*, 23(2):465–478, 2014.
[68] Jiyan Wu, Chau Yuen, Bo Cheng, Ming Wang, and Junliang Chen. Streaming high-quality mobile video with multipath tcp in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing*, 15(9):2345–2361, 2015.
[69] Ming Li, Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, and Antti Ylä-Jääski. Tolerating path heterogeneity in multipath tcp with bounded receive buffers. *Computer Networks*, 64:1–14, 2014.
[70] DriveU. DriveU100 . https://driveu.auto/product/driveu-100,, 2022.
[71] Bonding Cell Networks Using SD WAN - Part 1 . https://www.spikefishsolutions.com/post/bonding-cell-networks-using-sd-wan-part-1,, 2020.
[72] Peplink. MAX BR1 ESN . https://www.peplink.com/products/max-br1-esn/,, 2023.
[73] OpenWRT. mwan3 (Multi WAN load balancing/failover) . https://openwrt.org/docs/guide-user/network/wan/multiwan/mwan3,, 2023.
[74] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 557–570, New York, NY, USA, 2020. Association for Computing Machinery.
[75] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 44–58, New York, NY, USA, 2018. Association for Computing Machinery.
[76] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.
[77] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438, 2015.
[78] Hao Wu, Xuejin Tian, Minghao Li, Yunxin Liu, Ganesh Ananthanarayanan, Fengyuan Xu, and Sheng Zhong. Pecam: Privacy-enhanced video streaming and analytics via securely-reversible transformation. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 229–241, New York, NY, USA, 2021. Association for Computing Machinery.
[79] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 394–407, New York, NY, USA, 2019. Association for Computing Machinery.
[80] Devdeep Ray, Jack Kosaian, K. V. Rashmi, and Srinivasan Seshan. Vantage: Optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 380–393, New York, NY, USA, 2019. Association for Computing Machinery.
[81] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, Chen Sun, Gareth Tyson, and Hongqiang Harry Liu. Livenet: A low-latency video transport network for large-scale live streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 812–825, New York, NY, USA, 2022. Association for Computing Machinery.
[82] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 193–206, New York, NY, USA, 2022. Association for Computing Machinery.
[83] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. Neuroscaler: Neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 795–811, New York, NY, USA, 2022. Association for Computing Machinery.
[84] Johannes Blömer, Richard Karp, and Emo Welzl. The rank of sparse random matrices over finite fields. *Random Structures & Algorithms*, 10(4):407–419, 1997.
[85] Colin Cooper. On the distribution of rank of a random matrix over a finite field. *Random Structures & Algorithms*, 17(3-4):197–212, 2000.
[86] Raspberry Pi 4. https://www.raspberrypi.com/products/compute-module-4, 2023.
[87] OpenCV. Video input with OpenCV and similarity measurement) . https://docs.opencv.org/4.7.0/d5/dc4/tutorial_video_input_psnr_ssim.html,, 2023.
[88] R. Netravali A. Sivaraman and K. J. Winstein. Mpshell. https://github.com/ravinet/mahimahi/releases/tag/old, 2020.
[89] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 459–471, 2013.
[90] Real-time communication for the web. https://webrtc.org/, 2020.
[91] QUIC implementation. https://github.com/quicwg/base-drafts/wiki/Implementations, 2020.

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A DETAILS ON XNC'S ENCODING OPERATIONS

XNC always uses $n = 1$ when transmitting original packets for the first time. For the case where $n > 1$, we note that given each $g_s(i)$ drawn i.i.d. from $GF(2^8) \setminus \{0\}$, our coefficient generation scheme is equivalent to $p = \sum_{i=0}^{n-1} a_i p_{k+i}$, with each a_i drawn i.i.d. from $GF(2^8) \setminus \{0\}$. This is because $\sum_{i=0}^{n-1} a_i p_{k+i} = a_0 \sum_{i=0}^{n-1} b_i p_{k+i}$, where $b_0 = 1, b_i = \frac{a_i}{a_0}, \forall i > 0$. Also, we have $\forall i > 0, x \in GF(2^8) \setminus \{0\}$,

$$\begin{aligned}
Pr(b_i = x) &= Pr(a_i = a_0 x)\\
&= \sum_{j=1}^{j=2^8-1} Pr(a_0 = j) Pr(a_i = jx)\\
&= (2^8 - 1) \times \frac{1}{2^8 - 1} \times \frac{1}{2^8 - 1}\\
&= \frac{1}{2^8 - 1}
\end{aligned}$$

which proves that b_i subjects to uniform distribution on $GF(2^8) \setminus \{0\}$. Compared to its original form, XNC's coefficient generation scheme allows us to omit one expensive packet multiply operation per encoded packet. This optimization is especially beneficial in the case of XNC, where we determine encoding range boarders so that n is small (§4.4.2).

B CHOICE OF REDUNDANT PACKET COUNT

Here we give the proof of theorem 4.1:

Proof. We begin with the case where the n lost original packets have continuous packet numbers and we denote them as $\bar{\mathbf{o}} = \mathbf{o}_1, ..., \mathbf{o}_n$. Suppose that during the recovery, $n + k$ coded packets, labeled as $\bar{\mathbf{c}} = [\mathbf{c}_1, \cdots, \mathbf{c}_{n+k}]$, are received. From Appx. A we know: each coded packet $\mathbf{c}_i$ is a random linear combination of $\mathbf{o}_1, \cdots, \mathbf{o}_n$ as,

$$\mathbf{c}_i = \sum_{j=1}^{n+k} a_{i,j} \mathbf{o}_j,\ a_{i,j} \in GF(2^8) \setminus \{0\}$$

with each $a_{i,j}, j \in [1, n+k]$ drawn i.i.d. from $GF(2^8) \setminus \{0\}$ uniformly. Then, if we denote $\mathbb{A} := [a_{i,j}]$, we have:

$$\bar{\mathbf{c}}^T = \mathbb{A} \bar{\mathbf{o}}^T$$

where $\mathbb{A} \in \mathbb{F}_{2^8}^{(n+k)\times n}$. Denote as E_j the event that the first j columns of $\mathbb{A}$ are linearly independent, then $Pr(E_1) = 1$ because the first column can't be all 0 provided that each element is sampled in $[1, 2^8 - 1]$. For $j > 1$, we have:

$$\begin{aligned}
Pr(E_{j+1}) &= Pr(E_{j+1}, E_j) + Pr(E_{j+1}, \overline{E_j})\\
&= Pr(E_{j+1}|E_j) Pr(E_j) + Pr(E_{j+1}|\overline{E_j}) Pr(\overline{E_j})\\
&\overset{T_1}{=} Pr(E_{j+1}|E_j) Pr(E_j)\\
&\overset{T_2}{=} \frac{(2^8 - 1)^{n+k} - (2^8 - 1)^j}{(2^8 - 1)^{n+k}} Pr(E_j)
\end{aligned}$$

where T_1 holds because $Pr(E_{j+1}|\overline{E_j}) = 0$, which corresponds to the fact that first $j + 1$ columns are linearly independent only if the first j columns are linearly dependent. T_2 is because under E_j, the first j columns span a space of dimension j, with volume $(2^8 - 1)^j$. Therefore, there are $(2^8 - 1)^{n+k} - (2^8 - 1)^j$ equiprobable realizations of column $j + 1$ that are linearly independent of the first j columns. Then,

$$\begin{aligned}
Pr(E_n) &= Pr(E_1) \prod_{j=1}^{n-1} \frac{(2^8 - 1)^{n+k} - (2^8 - 1)^j}{(2^8 - 1)^{n+k}}\\
&= \prod_{j=1}^{n-1} \left(1 - \frac{1}{(2^8 - 1)^{n+k-j}}\right)\\
&= \prod_{i=k+1}^{n+k-1} \left(1 - \frac{1}{(2^8 - 1)^i}\right)\\
&= (1 - \frac{1}{(2^8 - 1)^{k+1}})(1 - \frac{1}{(2^8 - 1)^{k+2}}) \cdots\\
&\quad (1 - \frac{1}{(2^8 - 1)^{n+k-1}})\\
&\overset{T_3}{\geq} 1 - \frac{1}{(2^8 - 1)^{k+1}} - \cdots - \frac{1}{(2^8 - 1)^{n+k-1}}\\
&\overset{T_4}{\geq} 1 - \frac{1}{(2^8 - 1)^k \times (2^8 - 2)}\\
&= 1 - \frac{1}{255^k \times 254}
\end{aligned}$$

where T_4 holds by applying the sum of geometric progression, and T_3 is due to a recursive application of the following inequality:

$$a(1 - x) = a - ax > a - x, \forall x > 0, a < 1$$

For each step, set $a_j = (1 - \sum_{i=1}^{j} \frac{1}{(2^8-1)^{k+i}})$, $x_j = \frac{1}{(2^8-1)^{k+j+1}}$:

$$\begin{aligned}
&(1 - \frac{1}{(2^8 - 1)^{k+1}})(1 - \frac{1}{(2^8 - 1)^{k+2}}) \cdots (1 - \frac{1}{(2^8 - 1)^{n+k-1}})\\
&> (1 - \frac{1}{(2^8 - 1)^{k+1}} - \frac{1}{(2^8 - 1)^{k+2}}) \cdots (1 - \frac{1}{(2^8 - 1)^{n+k-1}})\\
&\vdots\\
&> 1 - \frac{1}{(2^8 - 1)^{k+1}} - \frac{1}{(2^8 - 1)^{k+2}} - \cdots - \frac{1}{(2^8 - 1)^{n+k-1}}
\end{aligned}$$

Because XNC could perform a successful decode if and only if all columns of the coefficient matrix $\mathbb{A}$ are linearly independent, the probability that XNC performs a successful decode is $Pr(E_n) \geq 1 - \frac{1}{255^k \times 254}$.

□

We note that our proof of Theorem 4.1 borrows idea from previous results on rank of random matrices [84, 85]. The difference between previous work and ours is that previous work assumes $a_{i,j} \in \mathbb{F}_q$, while we assume $a_{i,j} \in \mathbb{F}_q \setminus \{0\}$.

C PERFORMANCE EVALUATION PLATFORM

We used an in-house platform to analyze video streaming QoE metrics. This platform included three parts: (i) the live-streaming client that uploaded a reference video, (ii) the streaming receiver, and (iii) the analysis tool.

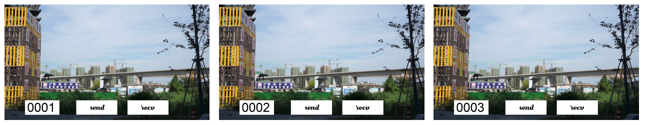


Figure 15: An example of three consecutive frames with increasing frame IDs in our reference video.

For part (i), we used *ffmpeg* [59], which could run on many different OSes and devices, as the video streaming client. In road tests, the video streaming client ran on a raspberry 4 [86] connected to our CPE via Ethernet. As for the reference video, we first used a camera on a moving vehicle to record the road scene (at 30fps). Then we assigned a frame ID to each frame. Fig. 15 shows an example of three consecutive frames marked with frame ID 1, 2, and 3, representing the first, second, and third frame in our reference video. The frame marks were used to identify each frame by the analysis tool.

For part (ii), we used a modified version of *ffmpeg* that printed useful information, including frame decoding timestamp and frame status (i.e., normal, corrupt, and missing). During the experiments, the stream receiver pulled the stream and saved it as a .mp4 file.

For part (iii), we developed a tool to analyze three QoE metrics: FPS, stall ratio, and structural similarity index measure (SSIM) score, from the receiver logs and recorded video files:

- FPS. We calculated the average number of decoded frames (normal frames) per second as FPS.
- Stall ratio. Stall ratio was calculated based on inter-frame delay interval. In our evaluation, we adopted a typical value that was used in streaming services, which was 200ms. Therefore, when observing an inter-frame delay interval larger than 200ms, our tool added it to the total stall time. It then calculated the stall ratio as: $\frac{total\_stall\_time}{total\_stream\_time}$.
- SSIM score. SSIM score was calculated as the structural similarity index between the reference video and the recorded video. However, these two videos might begin and end with different frame IDs, which made it difficult to calculate the correct structural similarity index. To solve this problem, we first used optical character recognition (OCR) to recognize the recorded video's beginning and ending frame IDs (with marked labels). Then, we aligned the recorded video with the reference video. Finally, we used OpenCV's APIs [87] to calculate the SSIM score based on the aligned videos.

D TRACE COLLECTION

As mentioned in §8, we use a trace-driven network emulator, *mpshell* [88], to conduct our controlled experiments. The original trace collector in *mpshell* (i.e., *saturatr* [89]) measures a link's capacity by saturating it with TCP traffic. This approach could not really saturate the fluctuating cellular links due to the reactive behavior of congestion control algorithms. In this work, we used a UDP-based trace collector instead. On each cellular interface, our collector sent packets at a configurable constant rate that responded to different capacities of 5G and LTE networks. The collector receiver recorded each packet's arrival timestamp and converted it to compatible formats with *mpshell*.

We set the sending rate of 5G paths as 100Mbps and LTE paths as 50Mbps (uplink). We argue that this is reasonable because 100Mbps is much larger than our testing application rate (30Mbps), and the theoretical maximum upload rate of LTE is no larger than 50Mbps. More than 100 traces on different road segments were collected and used for the controlled experiment part.

E HANDLING MTU PROBLEM

In CellFusion, the tunnel header overhead, including the extra bytes from IP header, UDP header, QUIC header, and XNC header, is 60 bytes at maximum, and our device interface MTU is 1500 bytes. Thus, to forward a full-sized (1500-byte) user packet, CellFusion sender must split it into two packets and reassemble them at the receiver side. To avoid this, we set the MTU of the CPE's *tun* interface to 1500 - 60 = 1440 bytes. With this setting, TCP packets would be smaller than 1408 bytes after PMTU negotiation; for UDP packets, we recommend our users to choose a small UDP packet size (e.g., 1300 bytes) as some popular UDP-based protocols do [57, 90, 91]. For worst cases where oversized (>1408 bytes) packets still present, the *tun* interface could automatically split them via IP fragmentation. Those fragmented packets would then be identified as different IP packets by CellFusion, and would not need to be further splitted/reassembled.