



Article

Toward Enhanced Reliability: An Efficient Method for Link-Local Retransmission in a Programmable Data Plane

Chenxiao Kong ^{1,2}, Lei Song ^{1,2}  and Yifei Li ^{1,2,*} 

¹ National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, No. 21, North Fourth Ring Road, Haidian District, Beijing 100190, China; kongcx@dsp.ac.cn (C.K.); songl@dsp.ac.cn (L.S.)

² School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, China

* Correspondence: liyf@dsp.ac.cn

Abstract: In wide-area networks (WANs) with high-speed, lossless transmission requirements, avoiding packet loss is crucial for ensuring link reliability, and maintaining link utilization over long distances is equally important. In this paper, we explore strategies for leveraging malfunctioning network links through link-local retransmission, optimizing our approach specifically for wide-area networks. To enhance performance, we select the duplicate threshold (dupthresh) based on the mean length deviation, which helps avoid triggering a certain portion of spurious fast retransmissions and reduces link bandwidth usage in an out-of-order environment. We evaluated our implementation on a programmable switch platform and found that this system maintained a low packet loss rate within a 10 Gbps line rate environment. It also reduces false retransmissions by 25% in the case of out-of-order links.

Keywords: link-local retransmission; packet reordering; in-network packet loss recovery; programmable switches



Academic Editor: Christos J. Bouras

Received: 13 November 2024

Revised: 27 December 2024

Accepted: 28 December 2024

Published: 31 December 2024

Citation: Kong, C.; Song, L.; Li, Y. Toward Enhanced Reliability: An Efficient Method for Link-Local Retransmission in a Programmable Data Plane. *Electronics* **2025**, *14*, 131. <https://doi.org/10.3390/electronics14010131>

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

1.1. Motivation

Packet loss frequently occurs in lossy networks, such as wireless networks, wide-area networks, data center networks, and cross-data center networks. In Remote Direct Memory Access (RDMA) of wide-area networks, for example, traffic has high requirements for the integrity and reliability of data packets [1]. Packet loss can cause significant performance degradation for RDMA. When the packet loss rate is greater than 0.01%, the throughput performance is less than 86.5 Gbits [2]. Although some losses are attributed to congestion, which has led to the development of various congestion control algorithms, research indicates that a substantial portion of packet loss is actually due to link failures. For example, a large-scale study conducted in 15 Microsoft data centers, which included 350,000 optical links, revealed that the number of packets lost due to corruption is comparable to the number lost due to congestion [3]. Furthermore, a recent study by Alibaba, which analyzed hundreds of real-world service tickets, found that approximately 18% of packet drops that cause network performance anomalies (NPAs) were attributable to packet corruption [4].

The root causes of damage are diverse, such as contaminated connectors, bent or damaged fibers, rotten or loose transceivers, and faulty shared components. This phenomenon can significantly impact many real-time or lossless networks, underscoring the importance of solutions for addressing packet loss caused by link corruption. The corruption of links

in networks is random and difficult to predict. The typical approach to handling link corruption involves physical repairs; however, these repairs can be time-consuming [3]. If the strategy is to reroute packets to bypass the faulty link [5], the original route will also require some time to detect the link failure and switch to an alternative route. During this transition period, the removal of the damaged link can lead to a reduction in network capacity, and the selection of new routing paths can significantly impact real-time networks that have stringent requirements for integrity and timeliness.

Therefore, relying solely on end-to-end recovery mechanisms is insufficient. The reliability of a link is closely tied to timely and efficient packet loss detection and recovery processes, which can keep the link operational until physical repairs are completed. This approach is crucial to maintaining the quality of the link and ensuring that network performance remains stable, even in the face of unexpected interruptions.

In addition, link corruption can significantly influence congestion control algorithms, ultimately affecting overall network performance. Traditional TCP congestion control mechanisms have predominantly been designed around loss-based strategies. This dependency can lead to a decrease in throughput, as packet loss and latency do not always correlate with actual network congestion levels. For example, under conditions where the link rate is 100 Mbps and the packet loss rate reaches 10^{-3} , the throughput of TCP Westwood+ experiences a substantial decline of approximately 50%. In contrast, TCP NewReno and SACK (Selective Acknowledgment) exhibit only a 10% reduction in throughput under the same network conditions [6]. There are also some congestion control mechanisms based on delay or explicit congestion notification [7]. But in wide-area network communication across data centers, the traffic is often a mix of flows. Due to the heterogeneity of the data, methods based on network measurement (ECN or delay) may not be suitable [8]. In this case, the network still needs to rely on packet loss to detect congestion, which further increases the demand for network quality.

These factors highlight the need for improved network reliability solutions to address both data integrity and performance in WANs. The reliability problem also involves some out-of-order problems on the link. An out-of-order packet is a network phenomenon in which the order of packet arrival is inconsistent with the order of packet sending. On a lossless network, reordering does not increase worst-case delay or jitter, but on a lossy network, reordering increases worst-case delay and jitter and has a significant impact on the TCP protocol on the receiving side [9]. The appearance of out-of-order packets has a great impact on the traditional protocols, causing unnecessary retransmissions at the transport layer and wasting bandwidth. The congestion window is unnecessarily reduced, reducing the network utilization rate. Therefore, when we solve the reliability problem of the link layer, we also need to take out-of-order packets into account.

1.2. Related Works

Researchers have a long history of studying packet loss recovery and network reliability, which involves Forward Error Correction (FEC), automatic repeat request (ARQ), frame replication and elimination for reliability (FRER), and so on. The choice of mitigation measures for each system depends mainly on the transmission medium. The first approach is FEC [10–12]. In particular, there is a study titled *Wharf that Uses Field-Programmable Gate Arrays (FPGAs) to Implement FEC in Faulty Links* [12]. However, FEC typically employs fixed redundancy parameters, which may not be suitable for dynamic packet loss rates. This rigidity can lead to inefficiencies, as the predetermined level of redundancy may not align with the actual network conditions. Furthermore, increased redundancy requires higher bandwidth, which can increase costs for certain network application scenarios like RDMA [2]. Consequently, while FEC provides a valuable mechanism for error correction,

its limitations in terms of adaptability and bandwidth consumption must be carefully considered when designing solutions for varying network environments. FRER reduces the impact of packet loss through redundant transmissions [13,14], but it also has a similar problem to FEC in that it requires more bandwidth.

Another existing method for fast packet recovery involves retransmitting lost packets directly at intermediate nodes. Link-local retransmission has been extensively studied and widely deployed in wireless networks [15]. With the development of network technology, both switches and relay nodes now have larger packet cache spaces and network processing capabilities integrated with turn-save-compute. Numerous previous efforts have implemented buffers at intermediate nodes to facilitate such link-local retransmissions (LL-ReTx). Therefore, in addition to its deployment and application in wireless networks, link-local retransmission has increasingly been recognized by researchers in recent years for its potential value in other types of networks, such as data centers [16,17] and satellite networks [18], which can significantly reduce recovery delays and conserve bandwidth. A comparison of link-layer packet loss recovery schemes in recent years is shown in Table 1. Some of these schemes are optimized for Flow Completion Time (FCT), while others are optimized for throughput and bandwidth.

Table 1. Comparison of link-layer packet loss recovery strategies.

Solution	Method	Application	Platform	Optimization
Wharf [12]	FEC	DCN ¹	FPGA	Throughput
LinkGuardian [16]	LL-ReTx	DCN	P4	FCT
SQR [17]	LL-ReTx	DCN	P4	FCT
SatGuard [18]	LL-ReTx	LSN ²	Simulation	FCT
UEC [19]	LL-ReTx	AI and HPC	Unknown	FCT
S-FRER [14]	FRER	FlexE	Simulation	Bandwidth

¹ Data Center Network (DCN). ² Low-Earth-Orbit (LEO) Satellite Network (LSN).

Shared Queue Ring (SQR) is an on-switch mechanism that recovers packets that could be lost during the period from the detection of a link failure to the completion of the subsequent network configuration [17]. However, while SQR can cache lost packets during link corruption and facilitate their recovery, it opts to switch routes instead of utilizing the potentially damaged link. This approach may lead to suboptimal resource utilization, as the original link could still be viable for certain types of traffic or under specific conditions. By not leveraging the existing link, SQR may miss opportunities to maintain network capacity and efficiency, particularly in scenarios where the damage is intermittent or localized. Link Guardian, which leverages faulty links, primarily focuses on managing short flows, incorporating specific designs, and optimizing for them. Furthermore, it does not offer detailed optimizations for handling out-of-order packets and timeouts, and the recovery mechanism has some potential bandwidth utilization problems like fake retransmissions. These issues are especially important in bandwidth-hungry WANs. Despite P4's ability to achieve extremely high forwarding line rates, it has certain limitations in terms of buffer space and protocol flexibility. In contrast, SatGuard is specifically optimized for the dynamic node characteristics of satellite networks, whereas our work remains concentrated on static networks in wide-area networks (WANs) and network communication across data centers.

In summary, for certain scenarios, link-layer recovery strategies and LL-ReTx still present opportunities for further exploration and optimization. We explored an approach to LL-ReTx and achieved some promising results.

1.3. Proposed Approach

To address the aforementioned challenges, we propose and implement a link-local retransmission design on a programmable data plane. Our proposed solution is based on a prototype programmable software switch, into which we have integrated this module. This module enhances link-layer reliability through retransmissions and extends the switch's bandwidth via link aggregation technology. In addition, it incorporates a cleverly designed feedback mechanism to optimize for out-of-order packets resulting from link aggregation. We employ a basic feedback mechanism that is distinct from existing link-layer retransmission methods for wired networks and introduce a new special protocol field. This field enhances the granularity of feedback provided to the sender, allowing for more precise control over retransmission decisions. The main contributions of this paper include the following:

1. We develop a feedback message transmission strategy that dynamically adjusts timing based on the out-of-order packet conditions of the link. By including the mean length deviation, we select dupthresh to avoid triggering a certain portion of spurious fast retransmissions.
2. The implementation leverages the Data Plane Development Kit (DPDK) to achieve high-performance packet processing. The retransmission logic is integrated into the switch's architecture, enabling it to monitor packet delivery and initiate retransmissions as needed, thereby improving overall network performance and reducing packet loss.

2. Methods

2.1. Reliable Transmission Design Mechanism

2.1.1. Basic LL-ReTx Mechanism

Our unit is integrated into each node, and the unit structure of each node is identical. However, one acts as the receiver and the other as the sender.

The mechanism can be summarized as follows: the sender temporarily stores the data packets in a cache while assigning to each packet a sequence number that increments by 1. When packet loss occurs, the receiver identifies the discrepancy between the sequence number of the received packet and its own expected sequence number, as outlined in Algorithm 1. In response to this detection, the receiver goes through the packet processing flow on the switch, as shown in Figure 1, and then sends a negative acknowledgment (NACK) to the sender, indicating which packet was lost.

Algorithm 1 Packet processing algorithm in the receiver

```

1: Input: pkt
2: Variables: Expected_Num
3: if pkt.pkt_num == Expected_Num then
4:   Expected_Num ← Expected_Num + 1
5:   forward()
6: else if pkt.pkt_num != Expected_Num then
7:   Expected_Num ← max(pkt.pkt_Num) + 1
8:   Call "Processing Flow of Reliable Transmission"
9:   send NACK back to Sender()
10: end if

```

Upon receiving the NACK, the sender retrieves the corresponding cached packet and retransmits it to the receiver. This approach, which utilizes the receiver to send NACKs, is specifically designed to conserve bandwidth and represents a significant departure from

the previous LL-ReTx mechanism [16,17]. By implementing this scheme, we enhance the efficiency of packet retransmission while minimizing unnecessary network traffic.

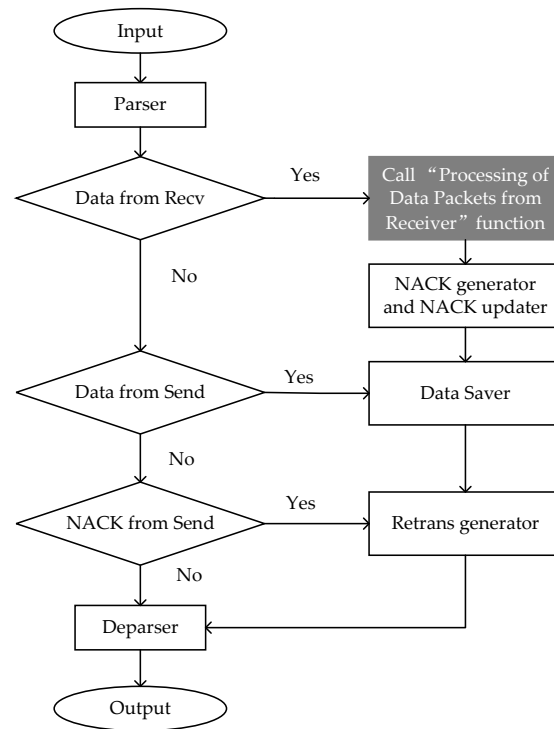


Figure 1. Processing flow of reliable transmission.

2.1.2. NACK Dynamic Adjustment Transmission Mechanism

Packet reordering occurs when the receiving order of a flow of packets (or segments) differs from the order in which they were sent. Studies have shown that packet reordering is not a rare phenomenon and can have significant implications for data integrity and application performance [20]. Most previous research focused on the transport layer, including research on out-of-order algorithms related to the TCP protocols on the receiver side. Some of these processing algorithms have certain reference value for our out-of-order scenarios. Some algorithms, like state reconciliation [21], only solve the problem of congestion window reduction caused by out-of-order packets, but they do not solve the problem of false retransmissions.

We compared several algorithms [20,22] for handling packets out-of-order packets under the same link conditions and evaluated their deviation from the actual waiting time using the Root Mean Square Error (RMSE). The Fixed K Algorithm 2 adjusts the value of K by either increasing or decreasing a fixed amount based on the previous calculation. Its advantage is its simplicity in computation, but the downside is its slow convergence, especially during sudden changes in link reordering. The Half-Growth Algorithm 3, on the other hand, converges more quickly to the actual situation by calculating the average, but it is susceptible to the impact of sudden changes, particularly during packet loss situations, where it may lead to an overestimation of the sending time due to prolonged transmission. In addition, there is a Fixed Time Algorithm that tolerates fixed sending times, meaning it does nothing to handle reordering. We perform a comparative study of these algorithms in Section 3.1.

We used link aggregation to scale the original bandwidth and throughput to 10 G, but it also caused some problems. It is important to note that link aggregation can also lead to the out-of-order delivery of packets, which can cause unnecessary false retransmissions, resulting in a large number of NACKs and non-essential retransmission packets consuming a lot of bandwidth.

Algorithm 2 Fixed K Algorithm

```

1: Input: L, dupthresh
2: if L > avg then
3:   avg ← avg + K
4: else
5:   avg ← avg - K
6: end if
7: dupthresh ← avg

```

Algorithm 3 Half-Growth Algorithm

```

1: Input: L, dupthresh
2: avg ← (avg + L)/2
3: dupthresh ← avg

```

In our scenario, packet reordering caused by link aggregation often leads to spurious retransmissions, which can result in significant bandwidth consumption, especially when the packet loss rate is high. To address this issue, we have implemented a mechanism during the processing of data packets that reduces the impact of reordering. The core idea of this mechanism can be summarized as dynamically setting the transmission time for a NACK based on the average arrival time of out-of-order packets. This approach aims to minimize excessive waiting while tolerating a certain degree of reordering.

Figure 2 illustrates the internal data packet processing. In particular, this system includes a process for handling out-of-order packets. The method employed in this paper effectively reduces the occurrence of spurious retransmissions to some extent. The gray boxes in the flowchart provide further clarification of Algorithms 4 and 5 that we implemented.

Algorithm 4 Send NACK packet to outside

```

1: Input: nack_timer, dupthresh
2: if nack_timer > dupthresh then
3:   call back “send nack to outside”
4:   clear nack
5: else
6:   send task back to the loop
7: end if

```

Algorithm 5 Update dupthresh

```

1: Input: L, dupthresh
2: if L > avg then
3:   avg ← (1 - α) · avg + α · L
4: else
5:   avg ← (1 - α · x) · avg + (α · x) · L
6: end if
7: dupthresh ← avg

```

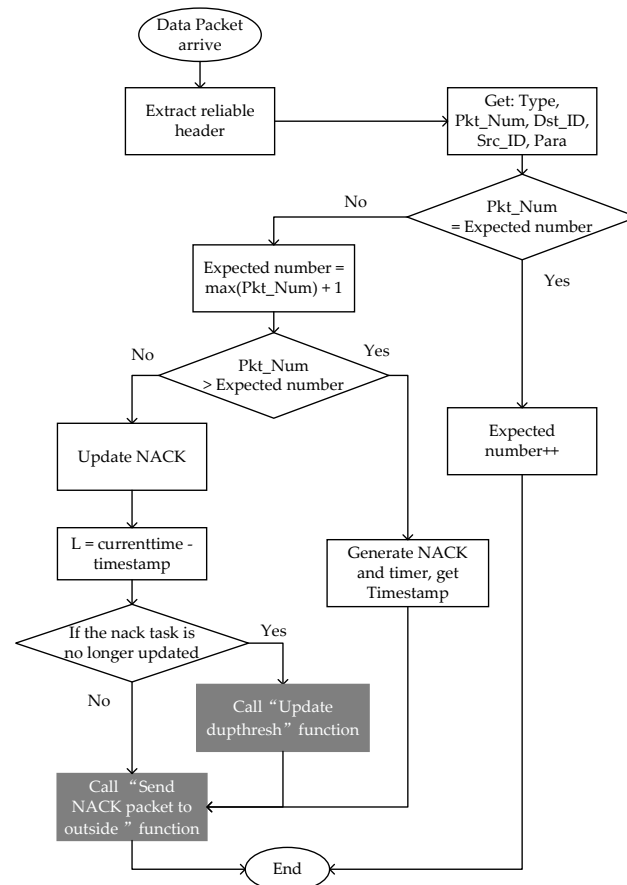


Figure 2. Processing flow of data packets from the receiver.

When the receiving end detects a missing packet, it starts recording the time L of the out-of-order packet until the missing packet arrives. When the received packet sequence number is greater than the expected sequence number, a new retransmission request task is created, and a timestamp is recorded. If out-of-order packets occur at this time, the arriving packet will have a sequence number smaller than the current retransmission request task identifier. This will update the previous retransmission request task and record the current time of the last update before the retransmission request task. Each task will record a timestamp when it is created and a last updated timestamp when out-of-order packets occur. The time difference between these timestamps can be used as the retransmission request packet's sending time. Setting this to the average of ten measurements can avoid the impact of a single measurement being too large or too small. If no out-of-order packets occur, the current time is not recorded, and the sending time is set to a default value. We use $(\text{current time} - \text{timestamp})$ as L in the algorithm shown below Algorithm 4.

The core of this algorithm relies on two key timings: one is the transmission time, dupthresh , which is established when the NACK timer is created, and the other is the actual waiting time, L , that the task needs to endure due to link reordering. The value of dupthresh is determined by the previously established L . Once the NACK timer is set with this dupthresh , it remains unchanged. Any modifications to L will only affect the dupthresh assigned to the next newly created NACK timer.

The specific algorithm is shown in Algorithm 5.

In addition, we employ a clever Exponentially Weighted Moving Average (EWMA) method to calculate the averages for this model, thereby enhancing its robustness [23].

Among them, α is the EWMA factor, usually $1/3$, and x is the multiplicative factor, usually 4. Subsequently, the dupthresh is set to the average out-of-order packet length, avg .

The advantage of this algorithm is that dupthresh can be dynamically updated according to the network state, and it avoids the excessive influence of a single out-of-order packet length on dupthresh.

As shown in Figure 3, whether the message generation unit qualitatively confirms the processing description of data messages depends on the context. When the expected sequence number is greater than the packet number, a confirmation task for the retransmission range is created based on the actual situation. The sequence number of the mapping bitmap is 1–8 (with the start sequence number determined by the expected sequence number when the packet is created), and the timer is used to set the sending time. When the packet with seven serial numbers is received at the last update, the time difference for updating the sending time of subsequent negative acknowledgment packets is recorded. The actual mapping bitmap to be sent is determined using the previously set sending time through EWMA. For example, when packet 5 is received, the mapping bitmap in the negative acknowledgment packet is 0000110. That is, packets numbered 6 or 7 need to be retransmitted. Note that when a retransmission acknowledgment task is created, the expected sequence number must be outside the acknowledgment range of an existing task. Figure 3 shows the third build.

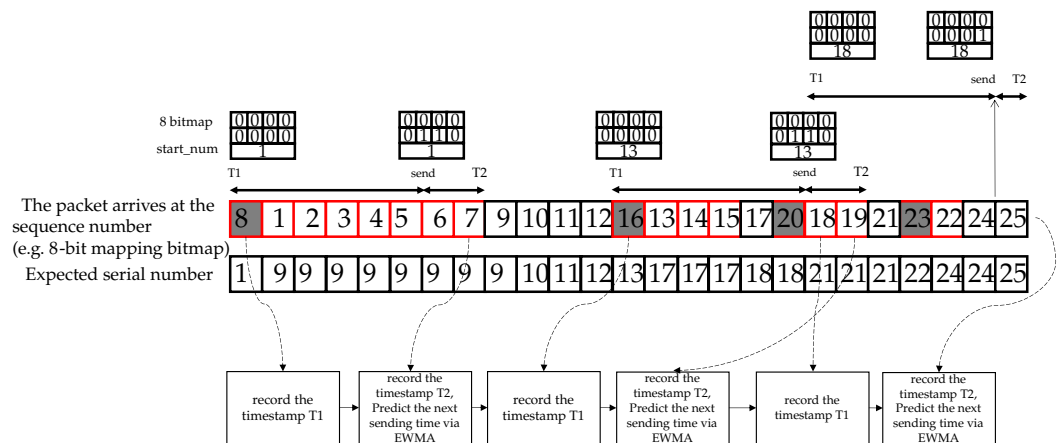


Figure 3. Sending instructions for NACK.

2.2. Discussion of Implementation Details

2.2.1. Implementation Platform: Programmable Data Plane-Enabled Switch

The Data Plane Development Kit (DPDK) is a software programming framework designed for data plane operations on general-purpose CPU platforms. It is commonly employed to accelerate packet processing in Software-Defined Networking (SDN) software switches [24,25]. The advantages of utilizing general-purpose CPU platforms include their inherent flexibility, programmability, advanced computational capabilities, and strengths in parallel processing. Software-based solutions can readily adapt to emerging network protocols and evolving service requirements, enhancing the flexibility of network design and maintenance. General-purpose CPUs are particularly adept at managing complex computational tasks, and with the widespread adoption of multi-core processors, parallel processing can significantly improve performance. With careful design, this approach can achieve line-rate speeds of up to 10 Gbps.

This paper presents the development and validation of an enhanced transmission unit based on a DPDK programmable switch, which we refer to as a reliable transmission unit. While our work is applicable to various types of networks, we believe it holds particular significance for addressing failures in data center links and facilitating communication between data centers over wide-area networks that demand high bandwidth utilization [2].

2.2.2. New Reliable Header

As networks evolve to accommodate increasing data traffic and diverse service requirements, there is a pressing need for innovative solutions to enhance communication and optimize resource utilization. The introduction of new protocol fields within Software-Defined Networking (SDN) frameworks presents an opportunity to address these challenges by providing additional context and control mechanisms that can improve packet processing, error handling, and overall network reliability [26,27].

As illustrated in Figure 4, we have developed a novel header, referred to as the reliable header, which is of variable length and is situated between the upper-layer header (e.g., TCP, UDP) and the lower-layer header (e.g., IP). This new header facilitates intermediate nodes, specifically programmable data plane-enabled switches, to self-estimate link loss using the information contained within the header. Consequently, these nodes can send control packets utilizing NACK to the receiver, thereby requesting retransmission. There are five fields in the header, which are described below.

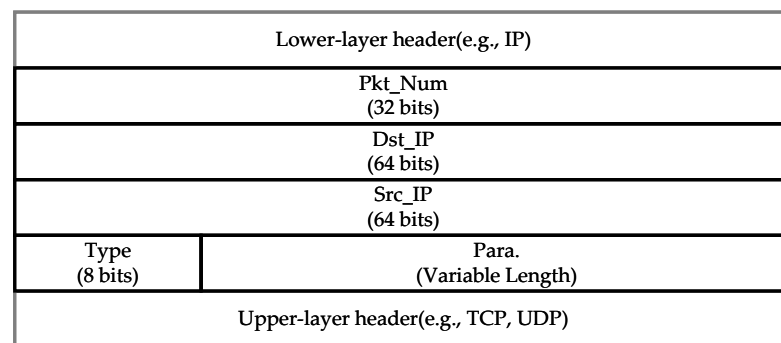


Figure 4. Reliable header.

The Type field specifies the protocol type of the packet and is used to indicate whether the reliable transmission feature is enabled. Additionally, it distinguishes between regular data packets and control packets like NACK used to trigger retransmissions and performs different processing.

The Dst_ID field represents the next hop address identifier of the packet, which can be an IP address or a custom overlay source-destination identifier. Similarly, the Src_ID field denotes the current node address identifier, which can also be an IP address or a custom overlay source-destination identifier.

The Type, Dst_ID, and Src_ID fields are utilized to identify a reliable transmission flow through mechanisms such as flow table matching. The reliable transmission flow refers to the data stream that employs the reliable transmission method provided by this invention.

The Pkt_num field indicates the sequence number of the reliable transmission data packet. The Pkt_num field is employed for packet loss detection and data packet retransmission. The usage and mechanisms of this field are discussed later.

The variable Para. field can be used for critical control information in control packets. We enable a bitmap within the NACK packet as a field to inform the sender which data packets need to be retransmitted. This approach allows a single NACK to carry more information, enhancing its efficiency. The length of the bitmap we utilize is 64 bits.

2.2.3. Implementation Details

Our switch can function both as a sender and a receiver. When a data packet enters the node, it first acts as a receiver, forwarding the packet to the reliable transmission unit. After that, the packet proceeds through the other components of the switch, such as flow table matching and other processing routines. Once the packet is ready to exit the switch, it

is sent back to the reliable transmission unit, where it is then processed as a sender and renumbered for sending. Now, we provide a detailed explanation of how this system ensures a low packet loss rate between two nodes. The packet transmission topology is illustrated in Figure 5.

The structure of the reliable transmission unit is shown in Figure 6, connected to the prototype DPDK switch via a virtual network interface. We first discuss how LL-ReTX ensures a low link-loss rate between nodes.

The previous hop node increments the sequence number of the transmitted packet and caches it, while also handling negative acknowledgment (NACK) packets received from the next hop. When a retransmission request from the next hop arrives, the node searches for the corresponding data packet in the cache.

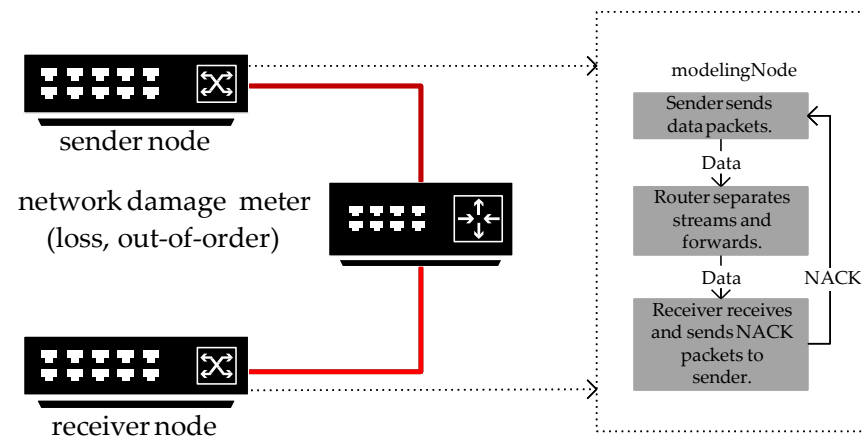


Figure 5. Topology of the 10 Gb/s testbed, using DPDK to model LL-ReTx and faulty links.

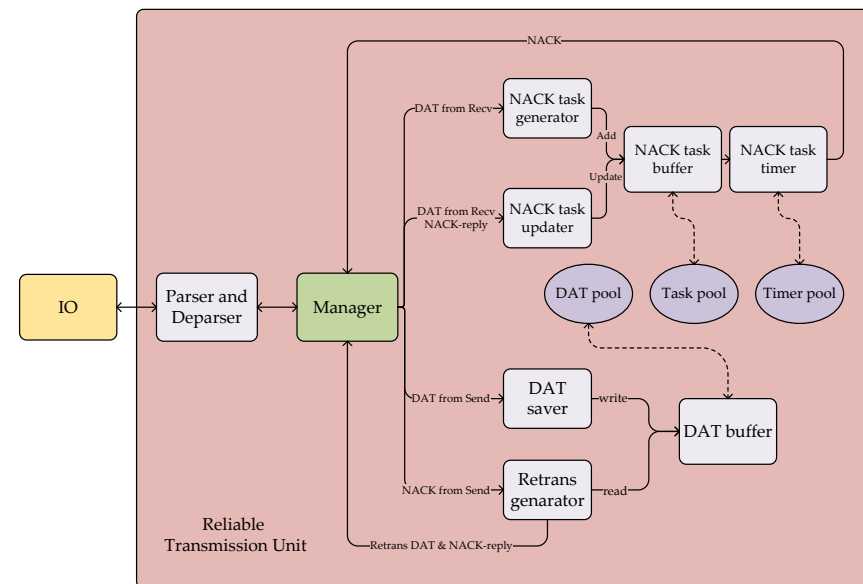


Figure 6. Processing flow of the reliable transmission unit.

The next hop node maintains an expected packet sequence number and monitors for anomalies in the data packets. If the packet sequence number does not match the expected sequence number, it is considered an anomaly and is handed over to the data reception unit for processing. Upon receiving a normal data packet or handling a packet with a sequence number greater than the expected sequence number, the next hop node updates the current expected sequence number.

When the next hop node detects an anomalous packet, it forwards the information to the NACK generation unit for processing. If the packet sequence number is greater than the expected sequence number, a new retransmission confirmation task is created, or an existing task is updated, using the starting sequence number and a bitmap to mark the sequence numbers of packets that need retransmission. If the packet sequence number is lower than the expected sequence number, the bitmap in the existing task is updated.

Each retransmission confirmation task is controlled by a timer to manage the transmission time. The bitmap is used to merge retransmission information for multiple packets, tolerating potential out-of-order packets on the link and thus reducing bandwidth consumption. The transmission time can be adjusted based on the degree of packet reordering on the link.

The response message from the previous hop's transmission unit to the NACK can trigger the reception unit to release the corresponding retransmission confirmation task, ending the generation of NACK packets.

2.2.4. Discussion About the Number of Retransmissions

Furthermore, the flexibility to configure the number of NACK transmissions and retransmissions according to specific application scenarios is crucial to minimizing packet loss. In scenarios with a high link corruption rate, there is a possibility that both a retransmitted packet and its corresponding NACK may be lost. To increase the likelihood of successful retransmission, the sender does not transmit only one copy of a buffered packet but rather multiple (N) copies, along with multiple (M) copies of the NACK in response to a loss notification. Considering that the original packet has already been transmitted and lost, the total number of copies sent for the lost packet is N + 1. If the NACK is also lost, this leads to an effective loss rate of

$$\frac{actual_loss_rate^{(N+1)} + actual_loss_rate^{(M+1)} - actual_loss_rate^{(M+N+1)}}{actual_loss_rate^{(N+1)}[1 + actual_loss_rate^{(M-N)} - actual_loss_rate^M]} = \quad (1)$$

Since the actual loss rate is less than 1, this formula is constrained by the smaller of M and N. Furthermore, both M and N are positive numbers. Our objective is not to completely eliminate corrupted packet losses but rather to reduce the effective loss rate to a target level specified by the operator. Therefore, we designate N as the smaller number. Ignoring higher-order small terms, we arrive at the following relation:

$$(actual_loss_rate)^{(N+1)} < (target_loss_rate) \quad (2)$$

For example, if a target loss rate of 10^{-8} is desired by a network operator and the actual loss rate on a corrupting link is 10^{-4} , then retransmitting a single copy of the buffered packet (N = 1, M > 1) would suffice to achieve an effective loss rate of 10^{-8} . Now, solving Equation (2), we obtain the number of retransmitted copies (N) as follows:

$$N > \frac{\log_{10}(target_loss_rate)}{\log_{10}(actual_loss_rate)} - 1 \quad (3)$$

Since N is an integer in practice, we assign N the next integer value by taking the *ceiling* of the RHS term of Equation (3). Also, note that since the loss rates are typically very low, this strategy to retransmit multiple copies adds a very small overhead.

3. Results

3.1. NACK Dynamic Adjustment Algorithm Simulation

The topology depicted in Figure 7 illustrates our simulation and experimental setup, where two nodes serve as the upstream and downstream entities, respectively. The two nodes are connected by a network impairment meter and configured with different parameters for out-of-order packets and packet loss damage. The network impairment meter is an abstract simulator that simulates out-of-order packets and packet loss in the network, and subsequent simulation parameters, such as packet loss rate and out-of-order degree, are configured by this impairment meter. This configuration forms the basis for our subsequent simulations and experiments, where we induced different degrees of re-ordering by manipulating the link parameters and setting specific packet loss rates using a network emulator.



Figure 7. Test topology.

We built a simulation environment in Python and used this algorithm to estimate a reasonable NACK sending time. The actual measured link delay was on the order of tens of ms. Therefore, we set the link latency to 40 ms and the link rate to 8 Mbps. The bitmap used by the receiver to generate the NACK was set to 64 bits. The topology of this simulation, as shown in Figure 7, involved an environment where the network impairment was 1%, and different out-of-order rates were set. This ratio increased gradually, with a depth of 20 out-of-order packets. The parameter K used in the comparison algorithm (Algorithm 2), as mentioned in Section 2.1.2, was typically set to 1, while the parameter α used by EWMA in Algorithm 5 was set to 13, with the multiplicative factor parameter x set to 0.5. The transmission waiting length L mentioned in the algorithm was the actual waiting time in the simulation, while avg was the predicted time, with an initial value of 0, which was calculated by the program in the subsequent simulation.

The sending time of the out-of-order tolerance control affects the bandwidth usage caused by the Flow Completion Time (FCT) and false retransmissions. Therefore, it is very important to fit the real sending time. A waiting time that is too long increases the FCT, while a waiting time that is too short leads to false retransmissions.

Although the FCT is an end-to-end concept, it can be inferred by accumulating values between nodes. In long flows, it is mainly related to the time of the last NACK transmission and the delay. We simulated and compared this in Figure 8. When looking solely at the FCT, the mean of the Fixed K Algorithm is smaller, which is an inevitable result of its slow convergence. When the delay suddenly increases, the NACK transmission time remains close to the previous result, but this approach causes an excessive number of false retransmissions, as shown in Figure 9; hence, it is not a good solution.

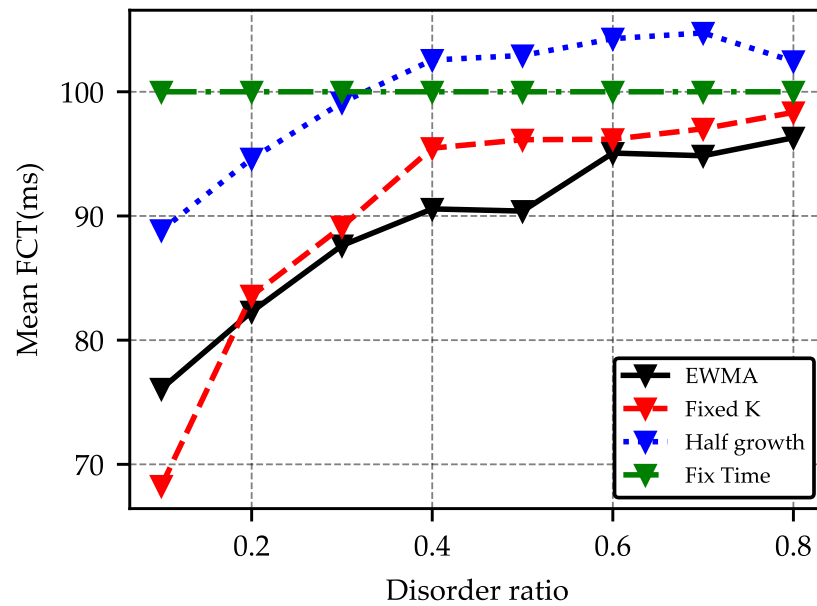


Figure 8. Mean FCT of different algorithms.

Therefore, we simulated some methods and compared their false retransmissions to further assess the effectiveness of our optimized link-local retransmission strategy. This comparison provided insights into the reduction in unnecessary retransmissions and the enhancement of network performance under various conditions.

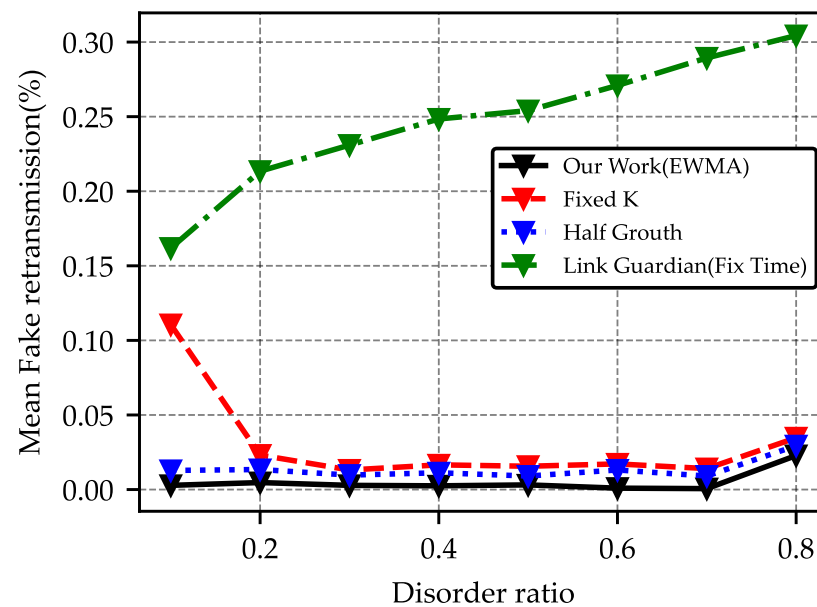


Figure 9. False retransmission simulation.

By simulating the packet disorder, we calculated the time for each NACK task to gather the packets and estimated the next time based on the previous times. We compared these algorithms for their prediction of sending times under the same environment, and the results are shown in Figure 10.

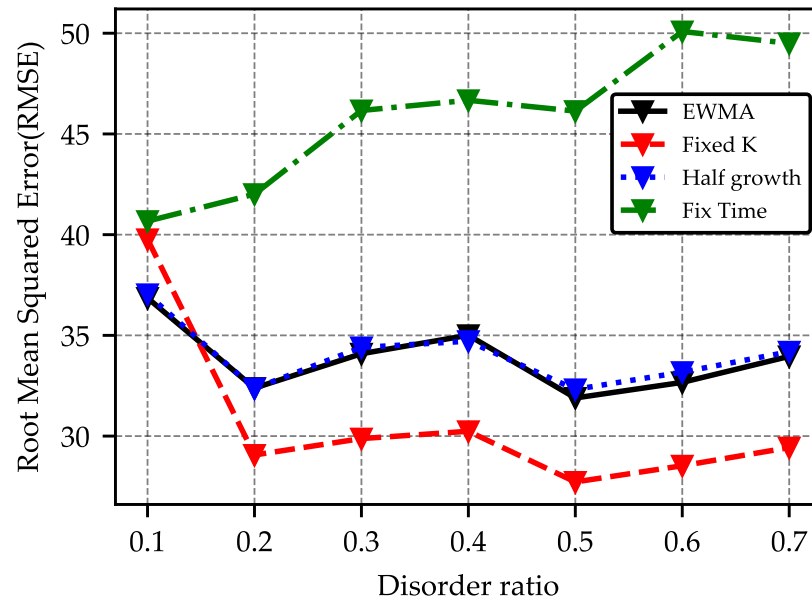


Figure 10. RMSE of different algorithms.

This method achieved better stability and lower false retransmission rates of around a quarter compared to other methods. Considering the above indicators, EWMA is indeed more suitable for this retransmission system.

EWMA achieved better stability and a better fit and was more sensitive to changes in data. The fit of the Fixed K Algorithm relied heavily on the value of K, and the stability performance was poor. When the delay of each link was stable, the disorder also remained relatively stable. We set three different disorder conditions and simulated three aggregated links. The fitted curves are shown in Figure 11. The true time refers to the actual waiting time of this NACK, while the estimated time refers to the predicted time using the EWMA algorithm. It can be observed that this algorithm is capable of dynamically adjusting the sending time of NACKs.

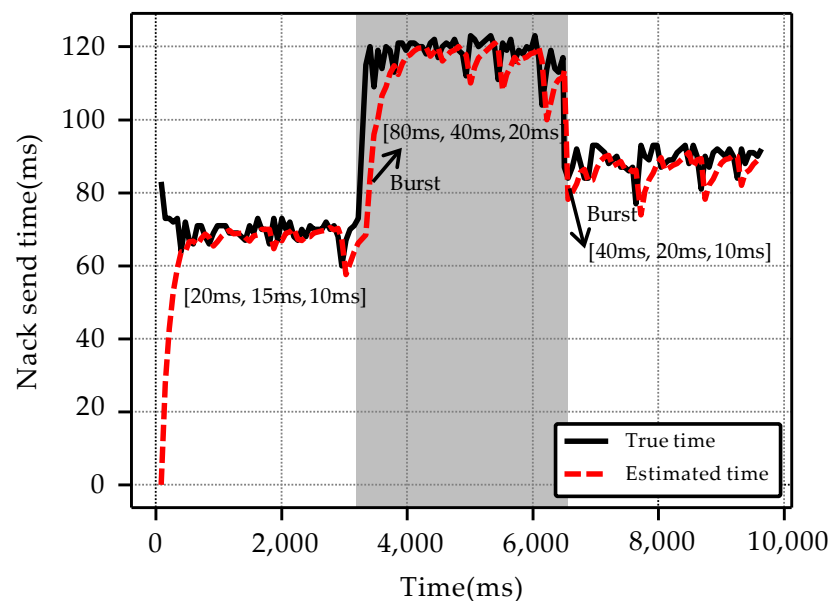


Figure 11. EWMA estimated time.

3.2. LL-ReTx Measurement of Real Packet Loss Scenarios

We continued utilizing the topology illustrated in Figure 7. This configuration represents a real-world link where we deployed our Data Plane Development Kit (DPDK) program across two switches. In this setup, we intentionally enabled random packet loss on the network interface card to simulate adverse network conditions. Furthermore, we established multiple lines through routers, resulting in a total of 16 distinct lines. The data traffic was segmented into multiple streams as it traversed the intermediate router, allowing for a comprehensive analysis of network performance.

The one-way propagation delay was measured to be 40 ms. As illustrated in Figure 12, it is evident that under the specific conditions of transmitting four negative acknowledgment (NACK) messages and subsequently retransmitting three data packets, the system demonstrated a remarkable ability to mitigate packet loss. With a real link packet loss rate maintained at a high 1% loss rate, the actual effective packet loss rate was reduced to an impressive extent, achieving a reduction of at least three orders of magnitude.

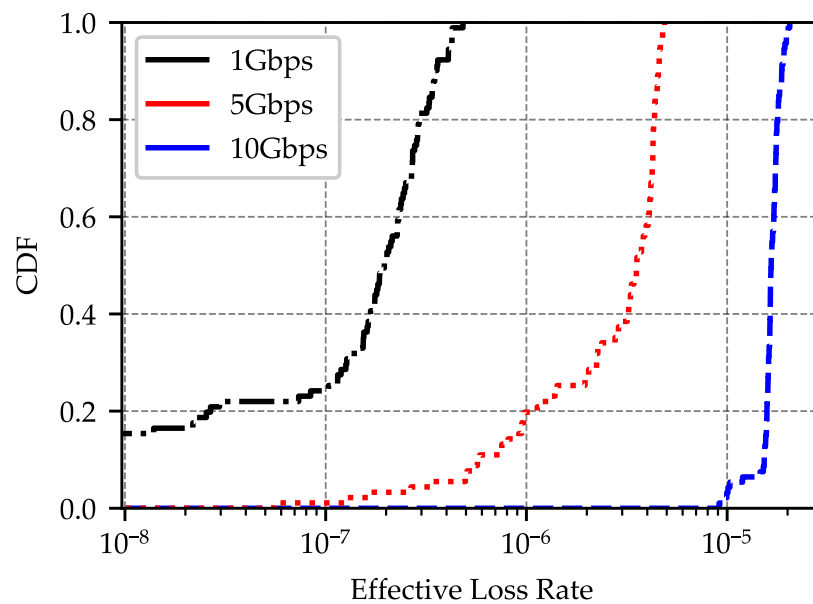


Figure 12. Effective packet loss rate.

The algorithm we used achieved good bandwidth utilization when the packet loss rate was below a certain magnitude. We observed that the length of the extra header bytes for ACKs or NACKs containing the necessary information was about one-hundredth of the maximum transmission unit (MTU). Considering the median packet size of 256 B observed in networks [28], Table 2 shows the bandwidth utilization calculated based on this parameter. We take into account the proportion of packets that were false retransmissions and the proportion of bandwidth used by NACKs and ACKs. The parameters were set according to the ratio of the false retransmission rate of the 3.1 simulation, which was nearly a quarter.

Table 2. Comparison of theoretical values of bandwidth utilization.

	Our Work	LinkGuardian [26]
Utilization (loss rate = α , count = n , retrans = m , length ratio = β)	$1/(1 + n\alpha\beta + m\alpha)$	$1/(1 + n(1 - \alpha)\beta + 1.25 m\alpha)$
(10^{-2} , 5, 3, 1/100)	96.9%	87.9%
(10^{-2} , 10, 3, 1/100)	96.9%	84.3%
(10^{-2} , 15, 3, 1/100)	96.8%	80.9%

We deployed our program on the nodes of a real-world long-distance transport network, whose topology is shown in Figure 13.

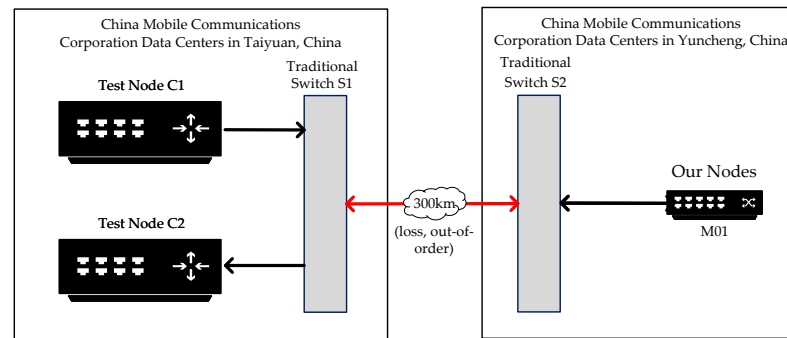


Figure 13. Real-world long-distance transport network topology.

We compared the transmission efficiency of our solution with that of traditional technologies in a real-world WAN environment, as shown in Table 3. At a distance of about 500 km and a packet loss rate of 1/1000 at 10 Gbps throughput, our approach maintained 76.8% of the network throughput, while the traditional TCP protocol, the Cubic algorithm, dropped to 0.30% of the network throughput, and the traditional RDMA technology dropped to 0.30% of the network throughput. The transmission efficiency of our approach was 256 times higher than that of the traditional TCP protocol and 256 times higher than that of the traditional RDMA technology, while the utilization rate of the entire network link increased to 77%.

Table 3. Comparison of the efficiency of different technologies.

Solution	Throughput (Gbps)
TCP Reno	0.038
TCP Cubic	0.030
TCP BBR	0.718
RDMA	0.03
Our Work	7.68

4. Discussion

4.1. Comparison with the Existing Literature

Our program has been running stably in the experimental environment for a long time. The reduction in the packet loss rate is inherently linked to the number of retransmissions, a concept discussed in Section 2.2.4 and further supported by the findings in Section 3.2. Our results align with those from other studies; for instance, Link Guardian [26] demonstrated a reduction from a 10^{-3} packet loss rate to a 10^{-6} rate through two retransmissions.

In comparison to existing link-layer retransmission mechanisms and algorithms, our approach excels in managing out-of-order packets, which are known to trigger false retransmissions and excessive waiting times. As detailed in Section 3.1, our simulations replicated the mechanisms used in other studies, including LinkGuardian [26]. Our mechanism, which includes processing for out-of-order packets, outperformed these mechanisms, reducing false retransmissions by 25%. Moreover, our NACK-based transmission strategy provided superior link utilization, as indicated in Table 2, by conserving resources and reducing overhead. Additionally, Table 3 illustrates that our solution mitigated the dramatic decrease in throughput caused by packet loss for various protocols on the devices discussed in the introduction.

Overall, by employing link retransmission, we prevent the significant reduction in throughput often associated with congestion control on the device side. Compared to other link-layer retransmission mechanisms, our approach enhances tolerance to out-of-order packets, thereby avoiding the bandwidth consumption incurred by false retransmissions.

4.2. Limitations and Future Work

While our research has made significant strides, there are certain limitations that warrant acknowledgment and attention in future studies. One such limitation is the handling of out-of-order packets resulting from retransmissions, which currently lacks a specific mechanism for reordering at the nodes. Additionally, although the algorithm and mechanisms we implemented in software provide a high degree of flexibility, the forwarding speed and throughput are somewhat constrained because of device platform limitations. This trade-off between flexibility and speed is a common challenge in software-defined networking, and we are committed to optimizing this in our future work. To this end, we plan to focus on the cache space required on the nodes for storing packets awaiting retransmission. The efficient utilization and management of this cache space will be a key direction for our subsequent optimization efforts.

Furthermore, while we have successfully integrated our code on programmable switches to facilitate forwarding, our mechanism is also designed to run on other types of nodes because the entire program is independent of the switch, and the input and output consist only of packets. During the testing phase, we discovered that only minor configuration and code changes are necessary to enable direct packet reception and transmission from the network port on these devices. However, this adaptation requires consideration of the storage capacity needed on these alternative devices. When applying our mechanism to other types of nodes, it is also necessary to configure and modify the corresponding interfaces within the code to accommodate the specific hardware capabilities and constraints. Additionally, some header configuration is required for the packet.

In summary, our future work will concentrate on refining the handling of out-of-order packets, optimizing the forwarding speed, and expanding the applicability of our mechanism to a broader range of network nodes. By addressing these limitations, we aim to enhance the overall performance and adaptability of our network retransmission scheme.

5. Conclusions

In this study, we implemented a novel link-local retransmission mechanism designed to enhance the reliability of data transmission in networks prone to packet loss. Our approach leverages continuous sequence numbers to facilitate efficient retransmission of lost packets, thereby maintaining the operational integrity of network links until physical repairs can be made. The implementation of this mechanism on a programmable switch platform demonstrated significant improvements in both packet loss recovery and overall network performance.

Through extensive simulations and real-world testing, we showed that our algorithm effectively mitigates the impact of packet loss caused by link corruption. The results indicate that by employing a feedback mechanism that dynamically adjusts retransmission strategies based on network conditions, we can significantly reduce the effective packet loss rate. This is particularly crucial in scenarios where traditional end-to-end recovery methods may fall short, as they often misinterpret packet loss as a sign of congestion, leading to unnecessary reductions in throughput.

Moreover, our findings highlight the importance of considering both data packets and control signals, such as ACKs and NACKs, in the calculation of bandwidth utilization. By optimizing the retransmission process and minimizing spurious retransmissions, we

achieved a high bandwidth utilization rate, even under significant packet loss conditions. This not only enhances the efficiency of the network but also ensures that real-time applications, which are sensitive to delays, disruptions, and bandwidth fluctuations, can operate smoothly.

In conclusion, we believe that our proposed solution is highly valuable for WANs. By addressing the challenges posed by packet loss and link corruption, we contribute to the development of more resilient and efficient networking solutions, ultimately enhancing the quality of service for users and applications that rely on stable and high-performance network connections. Future work will focus on further refining the feedback mechanisms and switch architecture design to predict and adapt to dynamically changing network conditions.

Author Contributions: Conceptualization, L.S.; methodology, implementation, and validation, C.K. and Y.L.; writing—original draft preparation, C.K.; writing—review and editing, L.S. and C.K.; supervision, L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China: Demonstration of Multimodal Network Application for Eastern Data and Western Computing (Project. No. 2023YFB2906404).

Data Availability Statement: All the necessary data are included in the article.

Acknowledgments: The authors would like to extend their sincere gratitude to Lei Song, Lei Liu, and Yifei Li for their insightful comments. Additionally, the authors are deeply appreciative of the anonymous reviewers for their constructive feedback on the earlier drafts of this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

WAN	Wide-Area Network
RDMA	Remote Direct Memory Access
NPA	Network Performance Anomaly
TCP	Transmission Control Protocol
SACK	Selective Acknowledgment
ECN	Explicit Congestion Notification
FEC	Forward Error Correction
DCN	Data Center Network
LSN	Low-Earth-Orbit (LEO) Satellite Network
HPC	High-Performance Computing
ARQ	Automatic Repeat Request
FRER	Frame Elimination for Reliability
FPGA	Field-Programmable Gate Array
LL-ReTx	Link-Local Retransmissions
FCT	Flow Completion Time
SQR	Shared Queue Ring
DPDK	Data Plane Development Kit
NACK	Negative Acknowledgment
RMSE	Root Mean Square Error
EWMA	Exponentially Weighted Moving Average
SDN	Software-Defined Networking
MTU	Maximum Transmission Unit

References

1. Lu, Y.; Chen, G.; Li, B.; Tan, K.; Xiong, Y.; Cheng, P.; Zhang, J.; Chen, E.; Moscibroda, T. Multi-Path Transport for RDMA in Datacenters. In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), Renton, WA, USA, 9–11 April 2018; pp. 357–371.
2. Zhao, J.; Li, F. Research on deterministic networking requirements and technologies for RDMA-WAN. *Telecommun. Sci.* **2023**, *39*, 39–51.
3. Zhuo, D.; Ghobadi, M.; Mahajan, R.; Förster, K.T.; Krishnamurthy, A.; Anderson, T. Understanding and Mitigating Packet Corruption in Data Center Networks. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17), Los Angeles, CA, USA, 21–25 August 2017; ACM: New York, NY, USA, 2017. [\[CrossRef\]](#)
4. Zhou, Y.; Sun, C.; Liu, H.H.; Miao, R.; Bai, S.; Li, B.; Zheng, Z.; Zhu, L.; Shen, Z.; Xi, Y.; et al. Flow Event Telemetry on Programmable Data Plane. 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), New York, NY, USA, 10–14 August 2020; pp. 76–89. [\[CrossRef\]](#)
5. Chiesa, M.; Kamisinski, A.; Rak, J.; Retvari, G.; Schmid, S. A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1253–1301. [\[CrossRef\]](#)
6. Ha, N.V.; Nguyen, T.T.T.; Tsuru, M. TCP with Network Coding Enhanced in Bi-Directional Loss Tolerance. *IEEE Commun. Lett.* **2020**, *24*, 520–524. [\[CrossRef\]](#)
7. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *Commun. ACM* **2017**, *60*, 58–66. [\[CrossRef\]](#)
8. Zeng, G.; Bai, W.; Chen, G.; Chen, K.; Han, D.; Zhu, Y.; Cui, L. Congestion Control for Cross-Datacenter Networks. *IEEE/ACM Trans. Netw.* **2022**, *30*, 2074–2089. [\[CrossRef\]](#)
9. Mohammadpour, E.; Le Boudec, J.Y. On Packet Reordering in Time-Sensitive Networks. *IEEE/ACM Trans. Netw.* **2022**, *30*, 1045–1057. [\[CrossRef\]](#)
10. Michel, F.; De Coninck, Q.; Bonaventure, O. QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC. In Proceedings of the 2019 IFIP Networking Conference (IFIP Networking), Warsaw, Poland, 20–22 May 2019; pp. 1–9. ISSN 1861-2288. [\[CrossRef\]](#)
11. Emara, S.; Fong, S.L.; Li, B.; Khisti, A.; Tan, W.T.; Zhu, X.; Apostolopoulos, J. Low-Latency Network-Adaptive Error Control for Interactive Streaming. *IEEE Trans. Multimed.* **2022**, *24*, 1691–1706. [\[CrossRef\]](#)
12. Giesen, H.; Shi, L.; Sonchack, J.; Chelluri, A.; Prabhu, N.; Sultana, N.; Kant, L.; McAuley, A.J.; Poylisher, A.; DeHon, A.; et al. In-network computing to the rescue of faulty links. In Proceedings of the 2018 Morning Workshop on In-Network Computing (SIGCOMM'18), Budapest, Hungary, 20–25 August 2018; ACM: New York, NY, USA, 2018. [\[CrossRef\]](#)
13. Varga, B.; Farkas, J.; Fejes, F.; Ansari, J.; Moldován, I.; Máté, M. Robustness and Reliability Provided by Deterministic Packet Networks (TSN and DetNet). *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 2309–2318. [\[CrossRef\]](#)
14. Zhang, J.; Gao, X.; Wu, K.; Ji, Y. Segment frame replication and elimination for redundant routing provision in the FlexE-over-WDM networks. *Opt. Switch. Netw.* **2023**, *47*, 100709. [\[CrossRef\]](#)
15. Li, L.; Liu, Y.; You, I.; Song, F. A Smart Retransmission Mechanism for Ultra-Reliable Applications in Industrial Wireless Networks. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1988–1996. [\[CrossRef\]](#)
16. Joshi, R.; Song, C.H.; Khooi, X.Z.; Budhdev, N.; Mishra, A.; Chan, M.C.; Leong, B. Masking Corruption Packet Losses in Datacenter Networks with Link-local Retransmission. In Proceedings of the ACM SIGCOMM 2023 Conference, New York, NY, USA, 10–14 September 2023; ACM: New York, NY, USA, 2023. [\[CrossRef\]](#)
17. Qu, T.; Joshi, R.; Chan, M.C.; Leong, B.; Guo, D.; Liu, Z. SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks. In Proceedings of the 2019 IEEE 27th International Conference on Network Protocols (ICNP), Chicago, IL, USA, 8–10 October 2019; IEEE: Piscataway, NJ, USA, 2019. [\[CrossRef\]](#)
18. Li, J.; Li, H.; Lai, Z.; Wu, Q.; Liu, Y.; Zhang, Q.; Li, Y.; Liu, J. SatGuard: Concealing Endless and Bursty Packet Losses in LEO Satellite Networks for Delay-Sensitive Web Applications. In Proceedings of the ACM Web Conference, Singapore, 13–17 May 2024.
19. Ultra Ethernet Consortium. UEC Progresses Towards v1.0 Set of Specifications. 18 March 2024. Available online: <https://ultraethernet.org/uec-progresses-towards-v1-0-set-of-specifications/> (accessed on 12 November 2024).
20. Leung, K.c.; Li, V.O.; Yang, D. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 522–535. [\[CrossRef\]](#)
21. Sarolahti, P.; Kojo, M.; Yamamoto, K.; Hata, M. Forward Rto-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP. Network Working Group. 2009. Available online: <https://datatracker.ietf.org/doc/html/rfc5682> (accessed on 12 November 2024).

22. Rani, A.R.; Lakshmi Nadh, K.; Sivanageswara Rao, S. An Analysis on Packet Reordering and Fast Retransmit schemes for TCP. *Int. J. Electr. Electron. Comput. Syst. (IJECS)* **2015**, *3*. Available online: <https://www.semanticscholar.org/paper/An-Analysis-on-Packet-Reordering-and-Fast-schemes-Rani-K.LakshmiNadh/efc6c62dbbebd29dd3380c5d45d4acad5a17a5df> (accessed on 12 November 2024).
23. Leung, K.C.; Ma, C. Enhancing TCP performance to persistent packet reordering. *J. Commun. Netw.* **2005**, *7*, 385–393. [[CrossRef](#)]
24. Shanmugalingam, S.; Ksentini, A.; Bertin, P. DPDK Open vSwitch performance validation with mirroring feature. In Proceedings of the 2016 23rd International Conference on Telecommunications (ICT), Thessaloniki, Greece, 16–18 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
25. Jin, M.; Wang, C.; Li, P.; Han, Z. Survey of load balancing method based on DPDK. In Proceedings of the 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), Omaha, Nebraska, 3–5 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 222–224.
26. Joshi, R.; Guo, Q.; Budhdev, N.; Mishra, A.; Chan, M.C.; Leong, B. LinkGuardian: Mitigating the impact of packet corruption loss with link-local retransmission. In Proceedings of the 6th Asia-Pacific Workshop on Networking, Fuzhou, China, 1–2 July 2022; ACM: New York, NY, USA, 2022. [[CrossRef](#)]
27. Ha, N.V.; Tuan Kiet, T.N.; Thanh Binh, B.N.; Tri, N.M.; Thao Nguyen, T.T.; Tsuru, M. Real-Time In-Band Network Link Loss Detection With Programmable Data Plane. In Proceedings of the 2024 16th International Conference on Knowledge and Smart Technology (KST), Krabi, Thailand, 28 February–2 March 2024; pp. 167–172. ISSN 2473-764X. [[CrossRef](#)]
28. Roy, A.; Zeng, H.; Bagga, J.; Porter, G.; Snoeren, A.C. Inside the Social Network’s (Datacenter) Network. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM’15), New York, NY, USA, 17–21 August 2015; pp. 123–137. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.