基于帧保序缓存的时间敏感网络FRER机制

蔡岳平，胡绍柳，韩笑
（重庆大学微电子与通信工程学院，重庆 400030）

摘　要：为解决时间敏感网络帧复制与消除可靠性（FRER）机制的帧乱序导致缓存区溢出帧丢弃问题，提出了一种基于相交节点处帧保序缓存的FRER机制。该机制在FRER路径对的相交节点处通过帧到达事件和定时器超时事件提前对无序帧进行重排序，保证帧的有序转发。同时还对保序缓存区大小和定时器的设计进行了分析。仿真结果表明，与传统帧排序机制相比，所提机制有效降低了帧丢失率、平均时延及抖动。

关键词：时间敏感网络；帧复制与消除；保序缓存
中图分类号：TN91
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025039

FRER mechanism based on frame preservation caching in time-sensitive networking

CAI Yueping, HU Shaoliu, HAN Xiao
School of Microelectronics and Communication Engineering, Chongqing University, Chongqing 400030, China

Abstract: An improved FRER mechanism based on frame preservation caching at intersecting node (FRER-FPCIN) was proposed to solve the frame disorder problems, which resulted in the cache overflow and frame discarding. The disordered frames were resorted at the intersection nodes through frame arrival events and timer timeout events to ensure orderly forwarding of frames. The design of cache size and timer was also analyzed. Simulation results show that the proposed mechanism effectively reduces the frame loss rate, average frame delay and jitter compared with traditional frame ordering mechanisms.

Keywords: time-sensitive networking, frame replication and elimination, preservation and caching

0　引言

随着工业自动化和智能制造的不断发展，人们对数据处理和传输的及时性、可靠性要求越来越高。对于实时性和低时延有严格要求的场景[1]，标准以太网无法保证有界低时延和低丢包率，因而无法满足高质量的服务需求[2]。对此，IEEE标准协会成立了音视频桥接（AVB, audio video bridging）工作组[3]，随后演进为时间敏感网络（TSN, time-sensitive networking）工作组。时间敏感网络技术的发展体现在2个方面。一方面，这一工作组发展的一系列标准[4]通过改良以太网，提高其在音视频传输、工业控制等高实时性和可靠性场景中的性能。时间敏感网络标准提供时间同步、低时延流控、可靠性和网络管控4个技术规范，确保数据传输具有确定性时延、高可靠性和精准时间同步，扩展了以太网的应用范围，如自动驾驶、实时控制系

收稿日期：2024–11–26；**修回日期：**2025–02–21
通信作者：蔡岳平，caiyueping@cqu.edu.cn
基金项目：国家重点研发计划基金资助项目（No.2020YFB1710900）；重庆市技术创新与应用发展专项5G重大主题专项基金资助项目（No.cstc2019jscx-zdztzxX0023）
Foundation Items: The National Key Research and Development Program of China (No.2020YFB1710900), Chongqing Technology Innovation and Application Development Key Project on 5G (No.cstc2019jscx-zdztzxX0023)

统和音视频流媒体传输等领域[5]。另一方面，学术界的一些学者深入地研究了IEEE 802.1工作组所提供的TSN标准技术，通过对这些标准技术进行改进和扩展，实现了在不同环境下满足网络的确定性和可靠性需求。

2005年，AVB工作组提出流预留协议标准，通过预留带宽减少数据帧丢失和阻塞。2012年，TSN工作组成立并致力于提升以太网性能，确保低时延和高可靠性。2018年，TSN工作组制定IEEE 802.1CB标准，引入帧复制与消除可靠性（FRER, frame replication and elimination for reliability）机制，最大程度减少拥塞和故障，提高帧传输可靠性。IEEE 802.1CB标准中制定的FRER机制为以太网提供无缝冗余特性[6]，通过在源节点和网络中继节点中对每个帧进行编号并复制，在目的节点和其他中继节点中消除这些复制帧，为流提供了更高的可靠性，将拥塞和故障影响降到最低。FRER机制通过在不相交的路径上发送重复的帧，允许关键流量的帧冗余[7-10]。即使工作路径发生故障，复制帧也可通过冗余路径到达目的节点，并在2个帧都到达目的地时，提供重复包消除机制。

TSN在IEEE 802.1CB-2017中引入FRER作为可靠性标准，重点关注帧传输的可靠性，以防故障安全和故障操作行为。尽管FRER通过帧的空间冗余实现了帧传输的无缝冗余，但在开销、时序、FRER机制内部以及对网络其他特性的影响等方面仍存在未解决的问题。其中一个关键方面是时序分析，目前在该领域的研究主要包括以下内容。

Thomas等[11]指出，FRER通过复制功能将帧复制到冗余路径上，之后路径再次合并，消除函数删除重复的帧。在这种冗余方案中进行复制可能导致复制路径上的突发增长和帧乱序，影响冗余和调度机制之间的相互作用。然而，该文并未提出具体的帧重排序方法来解决这一问题。Mohammadpour等[12]证明了如果流在其源和重排序缓存区之间是无损的，则重排序缓存区不会增加最坏情况时延和时延抖动。该理论成立的条件较特殊，且节点和链路发生故障的时间和位置是不可控的，但该文并没有给出帧重排序对流传输时延产生影响时完备的减缓或消除方案。Benes等[13]针对网络中数据包乱序问题提出了一种高效硬件实现的数据包重排序方案，并给出了该方案的资源消耗。这种重排序方案是在传输的接收端对数据包进行重排序，且重排序单元使用了最小的现场可编程门阵列（FPGA, field programmable gate array）资源。Tian等[14]指出TSN通过每队列时分复用实现实时通信，无须考虑传入帧的细节。然而，实际转发序列可能与窗口调度不同，因此存在帧乱序问题。为解决此问题，Tian等将一种帧排序整形器（FOS, frame ordering shaper）放置在接收端，以期望的序列对传入的帧进行排序。尽管进行了不同大小帧的仿真验证，但该方案未对帧丢失率和时延进行仿真测试，且FOS机制对帧传输的影响需要量化表示。Wang等[15]研究了动态波长和带宽分配（DWBA, dynamic wavelength and bandwidth allocation）技术对帧重排序的影响，提出了一种不影响带宽利用率并能减轻帧乱序的DWBA算法。此外，Wang等还分析了该算法对帧重排序光网络单元（ONU, optical network unit）数的理论上界，并通过仿真对DWBA算法的性能进行了评价，包括帧重排序ONU数和帧时延。

本文主要的研究工作如下。

1) 保序缓存算法对帧进行重排序时考虑了重排序缓存区设置的位置，在所有相交节点后设置保序缓存区对帧进行提前排序处理。同时保序缓存算法为无序帧设置有定时器T，使无序帧不再无限等待序号靠前的帧到达，而是在等待T时间后便被转发。

2) 对保序缓存区的大小和定时器的设置进行了分析，避免缓存区参数设置导致的帧丢失。

3) 仿真结果表明，所提机制有效降低了网络高负载时接收端缓存区排队帧过多以及等待时间过长导致的缓存区溢出。与传统的在接收端利用滑动窗口法和保序缓存算法进行帧排序相比，所提机制的帧丢失率最高分别降低了3.2%和1.4%。

1　帧乱序及缓存区溢出问题分析

TSN需要对一定数量的帧进行重排序，如Thomas等[11]所述，帧复制与消除可靠性机制利用空间冗余实现帧的冗余传输，但由于网络元素的并行性，不同路径路由的帧可能引起排序错误和突发增加。解决方案通常是在接收端设置缓存区，并在源端每帧头部添加序列号，以便接收端根据序列号进行重排序。然而，这种方案存在2个主要问题：

一是缓存区中的提前到达帧必须等待所有序列号较小的帧到达后才能转发，可能在网络拥塞时导致缓存区溢出丢弃后续帧；二是所有无序帧均需在接收端缓存区进行重排序处理，增加了接收端缓存区负载，也是网络拥塞导致缓存区溢出的原因之一。

缓存区溢出示意如图1所示，序列号大于N+1的帧在缓存区中排队等候，当缓存区已满时，序列号为N+1的帧还未到，新到达的所有其他帧均将被丢弃。因此，一种新的重排序方法需要被设计，实现在接收端之前对帧的顺序提前进行调整，并重新规划重排序缓存区中帧调度规则以解决缓存区溢出导致的帧丢失问题。

2　相交节点处帧保序缓存机制

2.1　相交节点处帧保序缓存分析

帧复制与消除可靠性机制允许帧通过多条路径传输，这些路径通常不完全不相交。由于网络拓扑和状态的多样性，在源节点和目的节点之间可能找不到完全不相交的2条路径[16]。路径对之间的相交节点具有帧消除能力，且位于多条路径的交汇处，在此处帧的突发性高且易发生帧乱序。因此，基于相交节点处帧保序缓存的FRER机制在每个路径对的相交节点后设置保序缓存区，在保序缓存区中，帧的顺序被提前调整，从而影响帧到达下一个相交节点处的到达曲线，并最终降低接收端处帧无序的情况。对于期望到达的帧，保序缓存区直接转发。对于提前到达的帧，保序缓存区为其设置一个定时器。当所有比该帧序列号小的帧都已到达保序缓存区时，该帧被释放。同时，当定时器超时，提前到达的帧被释放。另外，当保序缓存区中比该提前到达的帧序列号大的帧定时器超时，该提前到达的帧被释放。因此，对于一个帧A，在保序缓存区中其释放顺序遵循以下调度原则。

1) 若所有序列号比A小的帧都已经接收到了，则释放A。

2) 若帧A的定时器超时，则按顺序释放等于或小于帧A序列号的所有帧。

3) 若有接收到序列号更大的帧B定时器超时，则释放帧A以及其余比帧B序列号小的帧。

由上述可知，在保序缓存区中，有2类事件可以触发保序缓存区的保序规则：帧到达事件和定时器超时事件。

Thomas等[11]分析了FRER机制突发增长和帧乱序的原因，但是并没有给出具体的帧重排序方案。本文针对帧乱序的原因，提出了基于相交节点处帧保序缓存的FRER机制。该方案的新颖性在于每个相交节点都可以进行帧重排序，并且设置定时器超时值和保序缓存区大小以限制当前节点排队帧的排队时间以保证不会产生过高时延。Tian等[14]在接收端放置了一种FOS来应对帧乱序问题，该方案虽然可以在一定程度上降低帧丢失率，但是仅在接收端进行重排序很有可能会发生缓存区溢出从而导致帧丢失。本文在相交节点处进行帧重排序，减轻了接收端缓存区的重排序压力，改善了缓存区溢出导致帧丢失的问题。

2.2　帧到达事件

当一个新的帧p到达保序缓存区时，执行算法1。其中，变量buf代表一个保序缓存区列表，包含所有正在等待更小序列号帧到达的帧，变量N代表该保序缓存区期望接收的下一个序列号。

首先，判断帧p的序列号p.id是否大于或等于N−1。如果该判断不成立，则帧p是无效的，将该帧丢弃。这是因为N为保序缓存区期望接收的下一个序列号，新到达的帧p的序列号p.id要么是序列号为N−1的帧的副本，要么是期望的序列号的帧，要么是更大的序列号的帧。因此若该判断不成立，传输这种更小序列号的帧将违反传输顺序。

其次，判断帧p的序列号p.id是否大于或等于N。由第一个判断得知，此时帧的序列号均是有效值。如果该判断不成立，说明帧p是序列号为N−1的帧的副本，应当释放该帧。

接着，判断帧p的序列号p.id是否大于N。如

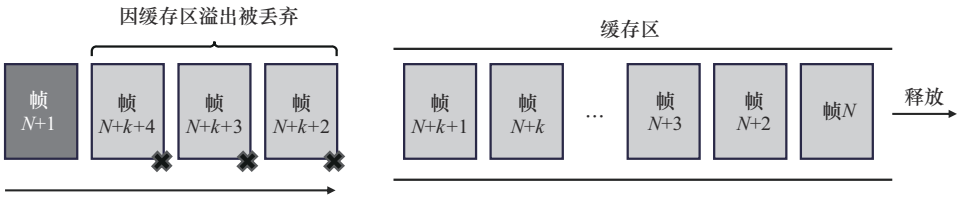

图1　缓存区溢出示意

果该判断不成立，则帧p是序列号为N的帧，且该序列号是保序缓存区期望到达的帧序列号。因此，将N增加1作为下一个序列号，然后释放该帧。和上面情况不同，此时还需检查保序缓存区中帧的排队情况，若此时保序缓存区中存在序列号为N（此时的N为已经增加1后的N）的帧，则应继续释放这些在保序缓存区中等待的帧。筛选符合序列号条件的帧由Check.buffer函数完成，在该函数中，首先判断保序缓存区中是否包含一个序列号等于更新后的N值的帧。若存在符合条件的帧，则使该帧从保序缓存区中退出队列，然后将N再次增加至下一个序列号。同时，退出保序缓存区的帧相应的定时器也会停止，最后释放符合条件的帧。接着继续调用Check.buffer函数对更新后的N值继续判断。这样递归调用函数Check.buffer，每次执行该函数时，只要有符合序列号条件的帧，都会再次更新N值，直到保序缓存区中没有序列号等于最后更新值N的帧。这样，所有序列号小于新N值的帧都已经从保序缓存区中释放。

最后，如果帧p的序列号p.id大于N，说明有比期望序列号更大的帧提前到达保序缓存区，这时应该让该帧在保序缓存区中等待序列号较小的帧到达。然后检查保序缓存区长度，如果保序缓存区的大小能容纳当前保序缓存区长度加上该帧的长度，则启动该帧的定时器，这意味着该大序列号的帧提前在保序缓存区中排队等待。若保序缓存区没有足够的容量容纳该帧，则该帧将因为保序缓存区溢出被丢弃。

帧到达事件的图形表示如图2所示。

算法1　保序缓存算法-帧到达事件

输入　帧p

变量　buf，N

```
1) if p.id ≥ N − 1 then  //如果成立，则p是一个有效帧，不成立则为无效帧
2)   if p.id ≥ N then  //如果成立，p为期望达到的帧或更大序列号的帧
3)     if p.id > N then //如果成立，p为比期望达到的帧序列号大的帧
4)       if buf.len() + p.len ≤ B then //检查缓存区长度
5)         TimerList.start(p.id, Time() +T) //启动帧定时器
6)         buf.enqueue(p) //帧排队
7)       else  discard(p) //出现溢出，丢弃帧
8)       end if
9)     else  //序列号为N，执行N+1，检查保序缓存区中帧排队情况
10)      N ← p.id + 1
11)      release(p)
12)      Check.buffer()
13)    end if
14)  else  //序列号为N−1，释放帧
15)    release(p)
16)    Check.buffer()
17)  end if
18) else discard(p) //出现无效帧
19) end if
20) function Check.buffer(void)  //筛选符合序列号条件的帧
21)   if buf.contains(N) then  //判断是否含序列号为N的帧
22)     p ← buf.dequeue(N)  //该帧退出队列
23)     N ← p.id + 1
24)     TimerList.stop(p.id)  //定时器停止
25)     release(p)
```

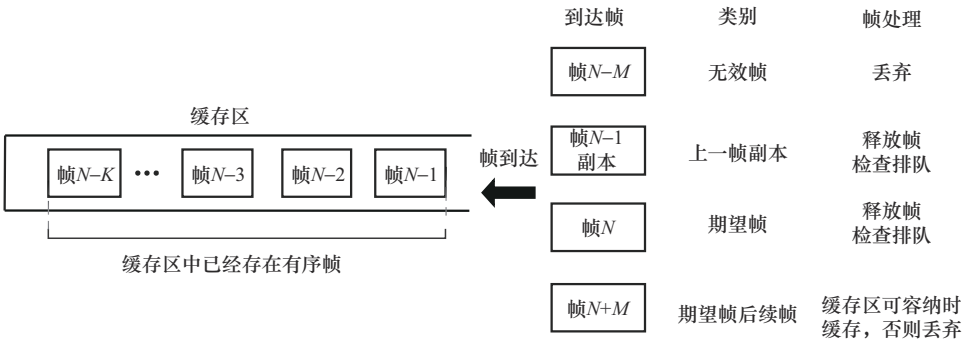

图2　帧到达事件的图形表示

26) Check.buffer()

27) end if

28) end function

算法 1 的复杂度集中在 Check.buffer 函数中。该函数依次检查缓存区中是否含有符合条件的帧，并处理这些帧。假设缓存区中的帧数量为 k，则该步骤可能需要遍历缓存区中的所有帧，最坏情况下时间复杂度为 $O(k)$。空间复杂度主要由缓存队列、定时器列表和变量存储组成。其中假设缓存区可以存储 k 个帧，则缓存区的空间复杂度为 $O(k)$。由于每个帧可能都需要一个定时器，因此定时器列表的空间复杂度也是 $O(k)$。其他变量的存储空间是常数级别的 $O(1)$。算法的空间复杂度主要由缓存区和定时器列表决定，因此该算法总的空间复杂度为 $O(k)$。

2.3 定时器超时事件

当序列号为 p.id 的帧的定时器超时，执行算法 2。其中，变量 buf 代表一个保序缓存区列表，包含所有正在等待更小序列号帧到达的帧，变量 N 代表该保序缓存区期望接收的下一个序列号。

根据 2.1 节中所述帧的调度规则，当序列号为 p.id 的帧定时器超时，应该释放序列号为 p.id 的帧以及比其序列号更小的帧。为了按顺序传递帧，保序缓存区首先按顺序释放保序缓存区中序列号小于或等于 p.id 的所有帧。利用函数 buf.contains(N)判断保序缓存区中是否包含序列号为 N 的帧，如果序列号为 N 的帧在保序缓存区中，则返回 TRUE，否则返回 FALSE。如果序列号为 N 的帧不在保序缓存区中，则 N+1。如果判断完成后保序缓存区中存在序列号为 N 的帧，则释放该帧，停止其定时器。每次有帧从保序缓存区中释放时，都要有相应的定时器被终止。并且每次在释放序列号为 N 的帧的同时，也会将继续增加 N 的值用于下一次判断，直至 N 的值大于 p.id 完成保序缓存区中所有序列号的帧的判断以及顺序释放。最后，使用 Check.buffer 筛选符合序列号的帧，检查保序缓存区以释放正在等待 id 为 p.id 或更小的帧。通过循环判断序列号小于 p.id 的所有帧，就实现了按顺序弹出保序缓存区中排队的帧。

定时器超时事件的图形表示如图 3 所示。

算法 2 保序缓存算法–定时器超时事件

输入 帧 id 号 p.id

变量 buf，N

1) while $N \leqslant p$.id do //按顺序释放序列号小于 p.id 的帧

2) while !buf.contains(N) do //判断是否包含序列号为 N 的帧

3) $N \leftarrow N + 1$

4) end while

5) $p' \leftarrow$ buf.dequeue(N) //该帧退出队列

6) $N \leftarrow p'$.id + 1

7) TimerList.stop(p'.id) //定时器停止

8) release(p'.id) //释放帧

9) end while

10) Check.buffer()

11) function Check. buffer(void) //筛选符合序列号条件的帧

12) if buf.contains(N) then //判断是否包含序列号为 N 的帧

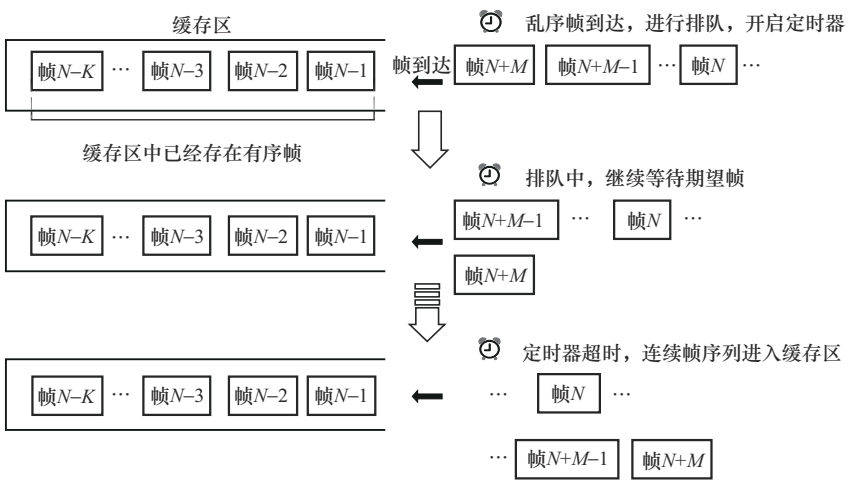

图 3 定时器超时事件的图形表示

```
13)     p ← buf.dequeue(N) //该帧退出队列
14)     N ← p.id + 1
15)     TimerList.stop(p.id) //定时器停止
16)     release(p)
17)     Check.buffer()
18)   end if
19) end function
```

算法2的复杂度由循环和Check.buffer函数决定。循环部分，外层while循环和内层while循环一起工作。外层循环遍历序列号小于N的帧，内层循环检查缓存中是否包含这些帧。假设最坏情况下缓存中的帧数量为k，那么这部分的最坏时间复杂度为$O(k)$（因为可能需要遍历所有帧）。Check.buffer函数的时间复杂度同样为$O(k)$，因此算法2的总时间复杂度为$O(k)$。空间复杂度由Check.buffer函数决定，因此与算法1一样为$O(k)$。

综合2种算法可知，帧p被保序缓存区释放只存在3种情况：1) 所有序列号小于帧p的帧都已经被保序缓存区接收到；2) 帧p的定时器超时；3) 保序缓存区中接收到序列号大于帧p的帧，且这个序列号较大的帧定时器超时，此时帧p以及其余比该帧序列号小的帧都被保序缓存区释放。

3　保序缓存区分析与设计

3.1　保序缓存区参数设置分析与设计

通过上述算法，可以实现保序缓存区中帧的有序释放。但当保序缓存区已满或到达的帧序列号小于N−1时，帧将被丢弃。这表明要考虑保序缓存区中的2个重要参数：保序缓存区大小B和定时器超时值T，这2个参数直接影响帧在保序缓存区中的处理情况。帧在以下2种情况下会被保序缓存区丢弃。

1) 保序缓存区大小B太小。

2) 定时器超时值T太小。

为了避免帧被丢弃，保序缓存区大小和超时值应被设置得足够大。这里引入RFC4737[17]中定义的2个包重排序度量：重排序时间偏移（REORDERING TIME OFFSET, RTO）和重排序字节偏移（REORDERING BYTE OFFSET, RBO），并利用它们对帧的重排序进行度量，定义如下。

令在源点观察到的第n个帧为帧p_n，E_n为在目的点观察到帧p_n的时间，若帧p_n在两点间丢失，则$E_n=+\infty$。由于网络中不是所有元素都是保序的，因此帧p_n不一定比帧p_{n+1}先到达目的点，那么这种时间上的偏移可以表示为

$$\lambda_n = E_n - \min_{j|j \geqslant n, E_j \leqslant E_n} E_j \tag{1}$$

其中，λ_n表示帧p_n重排序时间偏移，即序列号大于n的帧比帧p_n早到达的最大时间。如果帧p_n丢失，则λ_n无定义。

同样地，对于帧p_n，如果存在帧p_j使$j>n$且$E_j<E_n$，且帧p_n没有丢失，则帧p_n的重排序字节偏移量可以定义为

$$\pi_n = \sum_{j|j>n, E_j<E_n} l_j \tag{2}$$

其中，l_j为帧的字节大小，π_n表示帧p_n的重排序字节偏移，即序列号大于n的帧比帧p_n早到达的累计字节数。当帧p_n后没有出现乱序时，重排序字节偏移量π_n为0。

3.2　定时器超时值T的设置

假设保序缓存区中定时器的超时值为T，考虑保序缓存区大小B足够大，λ为源端到目的端的RTO，π为源端到目的端的RBO，A_n表示帧p_n到达保序缓存区的时间，D_n表示帧p_n从保序缓存区释放的时间，则帧p_n从保序缓存区释放的时间D_n可表示为

$$D_n = \begin{cases} X_n & ,n=1 \\ \max(X_n, Y_n) & ,n>1 \end{cases} \tag{3}$$

其中，X_n和Y_n为考虑不同参数情况下帧p_n从缓存区释放的时间，可由式(4)和式(5)给出。

$$X_n = \begin{cases} +\infty, A_n > \min\limits_{j \geqslant n} \{A_j\} + T \\ A_n \quad ,\text{其他} \end{cases} \tag{4}$$

$$Y_n = \min(D_{n-1}, \min_{j \geqslant n} \{A_j\} + T), n \geqslant 2 \tag{5}$$

则序列号小于n的帧不被缓存区丢弃的情况表示为

$$D_n \geqslant \max_{i<n|A_i \neq +\infty} D_i \tag{6}$$

假设对于$i<n$式(6)成立，则当n=1时，有$D_1=A_1$。当$n\geqslant 2$时，有

$$D_n = \max(A_n, Y_n) \tag{7}$$

对于集合$\xi_n=\{i<n|A_i\neq+\infty\}$，设$m$为其最大

值，$T \geqslant \lambda$，则有

$$\forall j > m: \quad A_m \leqslant A_j + \lambda \leqslant A_j + T \tag{8}$$

$$A_m \leqslant T + \min_{j \geqslant m} \left\{ A_j \right\} \tag{9}$$

因此，帧p_m从保序缓存区中释放的时间可表示为

$$D_m = \max(Y_m, A_m) \leqslant \max\left\{ T + \min_{j \geqslant m} \left\{ A_j \right\}, A_m \right\}$$

$$\overset{式(9)}{\Rightarrow} D_m \leqslant T + \min_{j \geqslant m} \left\{ A_j \right\} \tag{10}$$

这里需要讨论m的取值范围，可分为2种情况：$m=n-1$和$m<n-1$。

当$m=n-1$时，因为有$n-1<n$，且$\min\limits_{j \geqslant n-1} \left\{ A_j \right\} \leqslant \min\limits_{j \geqslant n} \left\{ A_j \right\}$，所以有$D_{n-1} \leqslant T + \min\limits_{j \geqslant n} \left\{ A_j \right\}$。利用式(7)，则帧$p_n$从保序缓存区中释放的时间为

$$D_n = \max(Y_n, A_n) \geqslant Y_n = \min\left(D_{n-1}, T + \min_{j \geqslant n} \left\{ A_j \right\} \right) = D_{n-1} \tag{11}$$

因此，没有帧被保序缓存区丢弃。

当$m<n-1$时，因为有$A_{n-1} = +\infty$，且$D_{n-1} = \max(Y_n, A_{n-1}) = +\infty$。利用式(7)，则帧$p_n$从保序缓存区中释放的时间可表示为

$$D_n = \max(Y_n, A_n) \geqslant Y_n = \min\left(D_{n-1}, T + \min_{j \geqslant n} \left\{ A_j \right\} \right) = T + \min_{j \geqslant n} \left\{ A_j \right\} \tag{12}$$

又因为$m<n-1$且$\min\limits_{j \geqslant m} \left\{ A_j \right\} \leqslant \min\limits_{j \geqslant n} \left\{ A_j \right\}$，所以帧$p_n$从保序缓存区中释放的时间为

$$D_n \geqslant T + \min_{j \geqslant n} \left\{ A_j \right\} \geqslant T + \min_{j \geqslant m} \left\{ A_j \right\} \geqslant D_m \tag{13}$$

因此，没有帧被保序缓存区丢弃。由此可知，当$T \geqslant \lambda$时，保序缓存区不会丢弃帧。现在假设$T < \lambda$且$\lambda > 0$，考虑这样一个场景，帧p_1到达保序缓存区的时间为$A_1 = t_0$，帧p_2到达保序缓存区时间为$A_1 = t_0 + \lambda$，由于$A_1 > \min\limits_{j \geqslant n} \left\{ A_j \right\} + T = A_2 + T$，则一定有$D_1 = X_1$且$X_1 = +\infty$，即有帧$p_1$被保序缓存区丢弃。因此保证保序缓存区不丢弃帧的最小超时值T应大于或等于λ。

3.3 保序缓存区大小B的设置

假设保序缓存区的大小$B = +\infty$，帧p_n的大小为l_n，则时间t时保序缓存区中帧大小为

$$L(t) = \sum_{A_i < t, D_i \geqslant t} l_i \tag{14}$$

考虑一个帧p_n，$n>1$且其大小为l_n，则当帧p_n到达保序缓存区时，还在保序缓存区中排队的所有帧集合可定义为

$$S = (i \in \mathbb{Z}^+ | i < n, D_i \geqslant A_n) \tag{15}$$

如果S为空，不会发生缓存区溢出问题。如果S不为空，且集合S中的最小值为m，即$m=\min(S)$。则帧p_n到达保序缓存区时帧大小为

$$L(A_n) + l_n = \sum_{i, A_i < A_n \leqslant D_i} l_i + l_n = \sum_{i < m, A_i < A_n \leqslant D_i} l_i + \sum_{i \geqslant m, A_i < A_n \leqslant D_i} l_i + l_n = \sum_{i \geqslant m, A_i < A_n \leqslant D_i} l_i + l_n \leqslant \sum_{i \geqslant m, A_i < A_n} l_i + l \tag{16}$$

因为$m<n$，且$E_m > E_n$，则此时的保序缓存区中帧大小可表示为

$$\sum_{i \geqslant m, A_i < A_n} l_i + l_n \leqslant \sum_{i > m, A_i < A_m} l_i = \pi_m \tag{17}$$

综合式(17)可知，保序缓存区大小的上界为π，即

$$L(A_n) + l_n \leqslant \pi_m \leqslant \pi \tag{18}$$

与数据链路层中常见的帧排序方法——滑动窗口法相比，该方法通过两大事件对帧的顺序进行处理，保证了帧的有序性。其次，该方法提前在前面节点处理了帧顺序，可以有效解决缓存区溢出的问题。当使用滑动窗口法时，如果序列号最小的帧一直没有到达，而序列号大的帧提前到达，若窗口大小不够大，则很有可能会造成序列号大的帧被丢弃。

4 性能分析与评价

本节介绍了本文提出的基于相交节点处帧保序缓存的FRER机制（FRER-FPCIN）、在接收端利用保序缓存算法进行帧排序以及在接收端使用滑动窗口法的帧排序机制在不同网络负载下的性能评价和对比分析。性能评价指标包括帧丢失率、平均时延和平均时延抖动。

4.1 仿真评价指标

1) 帧丢失率

帧丢失率指数据传输过程中，丢失的帧数量占发送的帧总数量的比率。帧丢失率越低，表示数据传输的可靠性越高，因此本文将帧丢失率作为仿真

评价指标之一。帧丢失率的计算式为

$$\eta = \frac{N_{\text{loss}}}{N_{\text{total}}} \tag{19}$$

其中，N_{loss}表示丢失的帧数量，N_{total}表示发送的帧总数量。

2) 平均时延

帧传输的平均时延（即端到端时延）包括排队时延、发送时延、传播时延和处理时延4个部分。排队时延是帧在队列中等待处理的时间，受网络拥塞影响而增大。发送时延取决于帧大小和发送设备带宽。传播时延与链路长度和材质有关。处理时延是帧到达路由器或交换机时进行检查和处理的时间。帧的平均时延计算式为

$$T_{\text{delay}} = T_{\text{send}} + T_{\text{spread}} + T_{\text{process}} + T_{\text{queue}} \tag{20}$$

其中，T_{send}表示发送时延，T_{spread}表示传播时延，T_{process}表示处理时延，T_{queue}表示排队时延。

3) 平均时延抖动

帧的平均时延抖动J可表示为

$$J = \max\left\{T_{\text{delay}}\right\} - \min\left\{T_{\text{delay}}\right\} \tag{21}$$

4.2　仿真参数设置

如图4所示，仿真包含13个节点、26条链路的网络拓扑。其中节点1为源节点，节点9为目的节点。另外，对于其余的实验参数，由于本文实验的仿真与Yao等[16]进行的仿真均以负载的变化作为变量，且仿真有相似之处，因此本文实验参考Yao等[16]对链路带宽、帧大小、帧到达过程等实验参数的选取。具体地，以1 Gbit/s作为链路带宽，网络负载取值范围设置为[0.1, 1.0]，帧大小设置为服从均匀分布，取值范围为64~500 B，帧个数设置为5×10^5个，帧到达过程设置为服从泊松分布，参数设置为500 frame/s，帧间隔时间设置为服从负指数分布，参数设置为10 μs，具体如表1所示，其中滑动窗口法的窗口大小设置为10 KB。

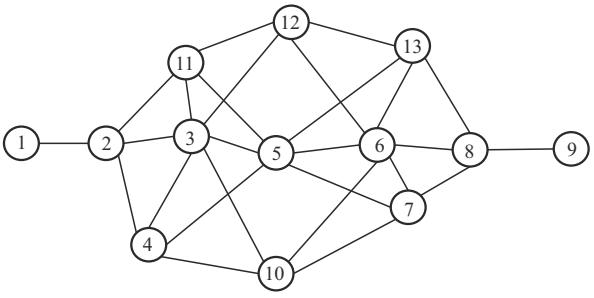

图4　仿真网络拓扑

表1　　仿真参数

参数	值
链路带宽/(Gbit·s^{-1})	1
网络负载	[0.1,1.0]
链路数量/条	26
节点数量/个	13
帧数量/个	5×10^5
帧大小/B	64~500（均匀分布）
帧到达过程/(frame·s^{-1})	500（泊松分布）
帧间隔时间/μs	10（负指数分布）

4.3　帧丢失率仿真结果与分析

图5显示了接收端滑动窗口帧排序机制、接收端帧保序缓存帧排序机制和相交节点处帧保序缓存的FRER机制（FRER-FPCIN）3种机制的帧丢失率随网络负载变化的情况。随着负载的增加，3种机制的帧丢失率都在升高，且在负载为0.6之后增长幅度加大。接收端滑动窗口帧排序机制在接收到提前到达的帧时，会将其暂存于缓存区，直到序列号较小的帧到达再释放，此机制在网络拥塞和负载较大时，容易导致大量帧丢失，帧丢失率最高。在接收端使用保序缓存算法帧排序，通过定时器T改善了网络拥塞时帧丢失的情况。基于相交节点处帧保序缓存的FRER机制在相交节点提前处理帧顺序，影响下一个节点的帧到达曲线，减少保序缓存区排队等待的帧，降低了帧丢失率。随着负载的增加，3种机制的帧丢失率都在上升，但任意负载下FRER-FPCIN机制的帧丢失率都比其余2种机制低，当负载为1.0时，该机制相对于接收端滑动窗口帧排序和接收端帧保序缓存帧排序机制，其帧丢失率分别降低了3.2%和1.4%。这表明负载越大，所提机制相较于其他机制性能提升越大。

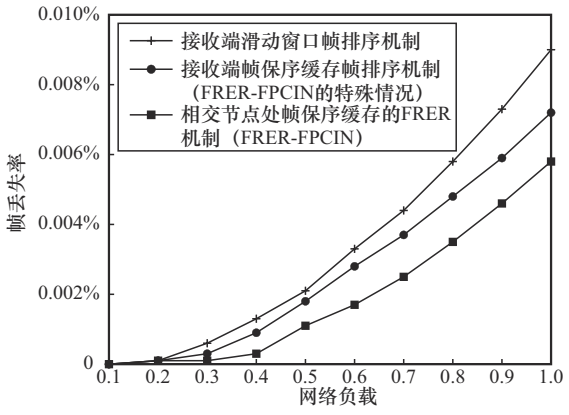

图5　不同网络负载下的帧丢失率

4.4　平均时延仿真结果与分析

图6显示了接收端滑动窗口帧排序机制、接收端帧保序缓存帧排序机制和相交节点处帧保序缓存的FRER机制（FRER-FPCIN）3种机制的平均时延随网络负载变化的情况。随着负载的增加，3种重排序机制的平均时延都在升高。接收端滑动窗口帧排序机制的平均时延最高，因为在接收端需要对所有的乱序帧进行处理，并且该机制对提前到达的大序列号帧需要在比其序列号小的帧到达后才会释放，因此在负载增大时会导致网络拥塞和排队时延增加。在接收端使用保序缓存算法并设置定时器T，平均时延稍好，因为定时器T的存在会改善因等待更小序列号的帧而导致的时延增加。相交节点处帧保序缓存机制提前处理乱序帧，避免重排序时间偏移扩增，降低了平均时延。如前所述，随着负载的增加，3种机制的平均时延都在上升，但是相较接收端滑动窗口帧排序和接收端帧保序缓存帧排序机制，FRER-FPCIN机制的平均时延无论在负载大或者负载小时都低。而且，负载越大，FRER-FPCIN机制的平均时延相比其余2种机制也低得更多。可以看出，在负载达到最大时，FRER-FPCIN机制的平均时延相对于其余2种机制分别降低了16.77%和9.15%。

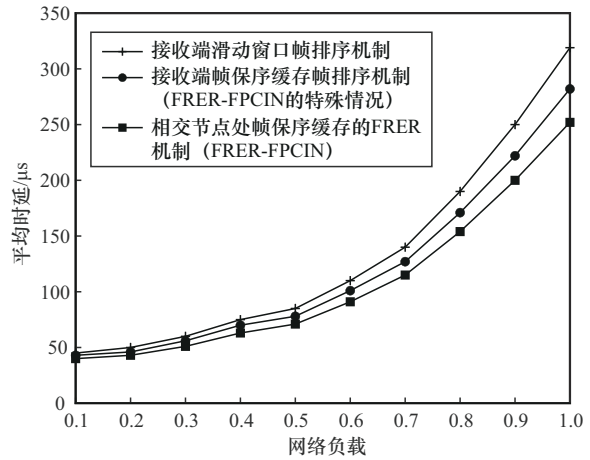

图6　不同网络负载下的平均时延

4.5　平均时延抖动仿真结果与分析

图7显示了接收端滑动窗口帧排序机制、接收端帧保序缓存帧排序机制和相交节点处帧保序缓存的FRER机制（FRER-FPCIN）3种机制的平均时延抖动随网络负载变化的情况。随着负载的增加，3种机制的时延抖动均增大。接收端滑动窗口帧排序机制的时延抖动最大，因为其对提前到达的大序号帧，需要在比其序列号小的帧到达后才能释放，所以增加了缓存区的负载，使缓存区更容易出现溢出。同时，乱序帧重排序时间偏移增加了缓存区排队时间，导致更大的时延抖动。在接收端使用保序缓存算法减少了缓存区溢出，通过改善时延抖动下界降低了时延抖动。相交节点处帧保序缓存的FRER机制提前处理帧顺序，减少了后续节点重排序时间偏移，通过改善时延抖动上界降低了时延抖动。如前所述，3种机制的平均时延抖动都在随着负载的增加而增大，但是相较接收端滑动窗口帧排序和接收端帧保序缓存帧排序机制，FRER-FPCIN机制的平均时延抖动无论在负载大或者负载小时都低。而且，负载越大，FRER-FPCIN机制的平均时延抖动相比其余2种机制也低得更多。在负载达到最大时，平均时延抖动相对于其余2种机制分别降低了19.44%和10.12%。

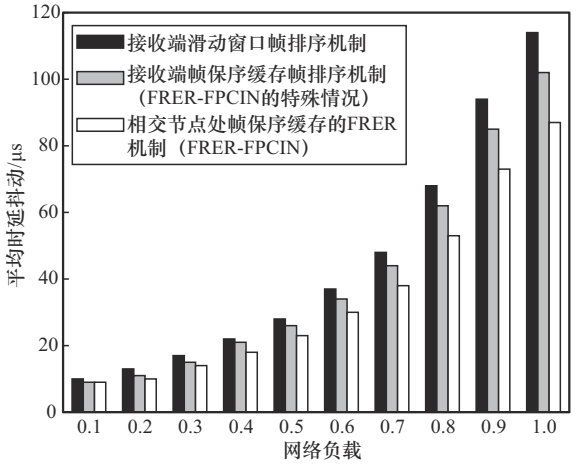

图7　不同网络负载下的平均时延抖动

5　结束语

本文提出的FRER-FPCIN通过在流传输路径对中的相交节点处设置保序缓存区提前进行重排序，减少后续相交节点以及接收端处的帧重排序时间偏移。同时保序缓存区中帧的调度规则用保序缓存算法实现，分为帧到达事件触发与定时器超时事件触发。最后为了避免保序缓存区中因缓存区大小B或定时器超时值T设置过小导致的缓存区溢出，利用帧重排序时间偏移和帧重排序字节偏移推导了这2个参数的范围。仿真结果表明，与接收端滑动窗口帧排序机制相比，所提机制能明显降低帧丢失率，并

且由于提前处理了帧的乱序问题，因此降低了后续相交节点上帧的重排序时间偏移，避免了帧重排序时间偏移的扩增，在时延和时延抖动上也有更好的表现。

然而该机制也存在如下挑战：随着网络规模增大和拓扑的复杂化，在相交节点处设置保序缓存区会增加部署成本。同时保序缓存区位置与大小的固定设置难以满足动态网络拓扑和网络流量的需求。因此，后续工作将研究如何使用机器学习等方法选择保序缓存区节点、确定缓存区大小和定时器超时值取值，以增强保序缓存机制的适应性和可扩展性。

参考文献：

[1] YAN L L, GE J C, PAN Z X, et al. Time-sensitive network control and management system for heterogeneous networks convergence[C]//Proceedings of the 2023 3rd International Conference on Intelligent Communications and Computing (ICC). Piscataway: IEEE Press, 2023: 60-69.

[2] XUE J L, SHOU G C, LIU Y Q, et al. Scheduling time-critical traffic with virtual queues in software-defined time-sensitive networking[J]. IEEE Transactions on Network and Service Management, 2024, 21(1): 967-978.

[3] IEEE standard for local and metropolitan area networks--audio video bridging (AVB) systems: IEEE 802.1BA-2021[S]. 2021.

[4] MIN J H, KIM W, PAEK J, et al. Effective routing and scheduling strategies for fault-tolerant time-sensitive networking[J]. IEEE Internet of Things Journal, 2024, 11(6): 11008-11020.

[5] SUN W J, ZOU Y, ZHANG X D, et al. Joint routing and scheduling optimization of in-vehicle time-sensitive networks based on improved grey wolf optimizer[J]. IEEE Internet of Things Journal, 2024, 11(4): 7093-7106.

[6] JOVER M, BARRANCO M, ÁLVAREZ I, et al. Migrating legacy Ethernet-based traffic with spatial redundancy to TSN networks[C]//Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA). Piscataway: IEEE Press, 2022: 1-8.

[7] KEHRER S, KLEINEBERG O, HEFFERNAN D. A comparison of fault-tolerance concepts for IEEE 802.1 time sensitive networks (TSN)[C]//Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA). Piscataway: IEEE Press, 2014: 1-8.

[8] QIAN S F, LUO F, XU J P. An analysis of frame replication and elimination for time-sensitive networking[C]//Proceedings of the 2017 VI International Conference on Network, Communication and Computing. New York: ACM Press, 2017: 166-170.

[9] PRINZ F, SCHOEFFLER M, LECHLER A, et al. End-to-end redundancy between real-time I4.0 components based on time-sensitive networking[C]//Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). Piscataway: IEEE Press, 2018: 1083-1086.

[10] HOFMANN R, NIKOLIĆ B, ERNST R. Challenges and limitations of IEEE 802.1CB-2017[J]. IEEE Embedded Systems Letters, 2020, 12 (4): 105-108.

[11] THOMAS L, MIFDAOUI A, BOUDEC J Y L. Worst-case delay bounds in time-sensitive networks with packet replication and elimination[J]. IEEE/ACM Transactions on Networking, 2022, 30(6): 2701-2715.

[12] MOHAMMADPOUR E, BOUDEC J Y L. On packet reordering in time-sensitive networks[J]. IEEE/ACM Transactions on Networking, 2022, 30(3): 1045-1057.

[13] BENES T, UBIK S, HALAK J. Packet reordering correction for low-latency network applications[C]//Proceedings of the 2022 11th Mediterranean Conference on Embedded Computing (MECO). Piscataway: IEEE Press, 2022: 1-5.

[14] TIAN Z R, YANG F. FOS: a shaping mechanism for frame ordering in time sensitive networking[C]//Proceedings of the 2022 IEEE 8th International Conference on Computer and Communications (ICCC). Piscataway: IEEE Press, 2022: 401-405.

[15] WANG W, GUO W, HU W S. Dynamic wavelength and bandwidth allocation algorithms for mitigating frame reordering in NG-EPON[J]. Journal of Optical Communications and Networking, 2018, 10(3): 220-228.

[16] YAO Z C, CAI Y P, LI T C. Multiple cascaded preconfigured cycles for the FRER mechanism in time-sensitive networking[C]//Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops). Piscataway: IEEE Press, 2021: 1-6.

[17] MORTON A, CIAVATTONE L, RAMACHANDRAN G, et al. Packet reordering metrics[R]. 2006.

[作者简介]

蔡岳平（1980– ），男，江苏丹阳人，博士，重庆大学副教授，主要研究方向为工业互联网、确定性网络、5G/6G网络、数据中心网络、算力网络等。

胡绍柳（1997– ），男，重庆人，重庆大学硕士生，主要研究方向为时间敏感网络。

韩笑（2001– ），男，山西忻州人，重庆大学硕士生，主要研究方向为时间敏感网络。