doi:10.12068/j.issn.1005－3026.2025.20230250

工业SDN中基于缓存的可靠组播研究

徐久强，邹九龙，徐 冲
（东北大学 计算机科学与工程学院，辽宁 沈阳 110169）

摘 要：为解决基于软件定义网络(software defined network,SDN)的工业物联网组播传输的可靠性问题,研究基于缓存思想的数据重传机制,提出了一种基于缓存的可靠组播方案.设计了基于缓存节点的可靠组播(cache-based reliable multicast,CBRM)框架和相应的可靠组播算法,在SDN中选择合适位置设置缓存节点并由其暂存组播数据,当接收端或中间转发设备发现数据丢失时,通过向上游缓存节点申请丢包重传以保证组播数据的可靠传输.为支持缓存节点数据重传,设计了基于UDP(user datagram protocol)的可靠传输协议URTP(UDP-based reliable transmission protocol).在Mininet仿真环境下进行了验证,验证结果表明CBRM框架和相应的可靠组播算法能在远低于传统重传成本的条件下保证组播传输的可靠性.

关 键 词：SDN;组播树;可靠组播;缓存节点;缓存管理
中图分类号：TP 20 **文献标志码**：A **文章编号**：1005-3026(2025)02-0001-09

Cache-Based Reliable Multicast in Industrial SDN

XU Jiu-qiang, *ZOU Jiu-long*, *XU Chong*
(School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: XU Jiu-qiang, E-mail: xujq@mail.neu.edu.cn)

Abstract: To solve the reliability problem of multicast transmission in industrial Internet of things (IIoT) based on SDN (software defined network), the data retransmission mechanism based on cache is studied, and a cache-based reliable multicast scheme is proposed. A cache-based reliable multicast (CBRM) framework and the corresponding reliable multicast algorithm are designed. Cache nodes for temporarily storing multicast data are deployed in selected locations in SDN. If receiving or forwarding node detects packet loss, it will require its upstream nodes to retransmit the lost packet. To support data retransmission of cache nodes, a UDP (user datagram protocol) -based reliable transmission protocol (URTP) is designed. The results of Mininet simulation show that the CBRM framework and the corresponding reliable multicast algorithm can guarantee the reliability of multicast transmission at a much lower cost than traditional methods.

Key words: SDN; multicast tree; reliable multicast; cache nodes; cache management

随着工业网络应用的不断扩展,工业组播受到学术界和工业界的广泛关注,工业组播在要求稳定、高效传输的同时,对组播的可靠性提出了更高要求.

在传统组播服务中,传输过程中存在数据包重复、丢失、失序等问题,这会影响组播的可靠性并导致传输效率下降[1].软件定义网络(SDN)[2]作为一种新的网络架构通过控制器对整个网络进行集中管理,可以实现拓扑、流量、QoS(quality of service)等的实时监控,可快速应对网络故障和异常情况[3].将SDN与组播相结合,可通过控制器的集中管理和控制网络中的组播流量,在优化网络资源利用的同时提供可靠组播传输[4-5].

现有工业SDN组播研究大多专注于框架、传输时延和可靠性.文献[6]采用多个一对一的TCP(transmission control protocol)连接实现对所

收稿日期：2023-10-05
基金项目：国家重点研发计划项目(2019JSJ2ZDYF01).
作者简介：徐久强(1966—),男,辽宁北镇人,东北大学教授.

有接收节点的组播通信,虽然可靠性得到了保证,但传输过程需要多次握手,且占用较多链路资源,严重影响组播数据传输时间;文献[7]提出的组播通信框架,解决了组播通信的实时性和扩展性问题,但并未考虑其可靠性;文献[8]提出利用强化学习生成最优组播树,但由于SDN的复杂性和基于机器学习方法的局限性,难以保证其准确性;文献[9]提出的基于桶的快速组播路由,提高了事件传递效率,但忽略了可靠性;文献[10]将组播树划分为多个EAM (elastic area multicast)子树,通过在子树中传输恢复数据来降低数据重传成本,但对路由成本和传输效率考虑不足,忽略了对传输时间的影响;文献[11]提出的SVC (scalable video coding)流的QoS感知组播和文献[12]提出的SDM4IIoT共享树,都实现了可扩展性,但未考虑可靠性.

因此兼顾可靠性、实时性和传输效率的SDN可靠组播工作仍需进一步研究.本文专注于解决SDN中可靠组播传输问题,主要研究工作如下:

1) 基于缓存和丢包重传的思想,设计了SDN中CBRM框架,为组播数据传输提供可靠性支持;

2) 研究了基于缓存的可靠组播算法,包括缓存节点的位置选择方案,设计了SDN中的缓存机制,该机制降低了接收节点处理数据的时间复杂度;

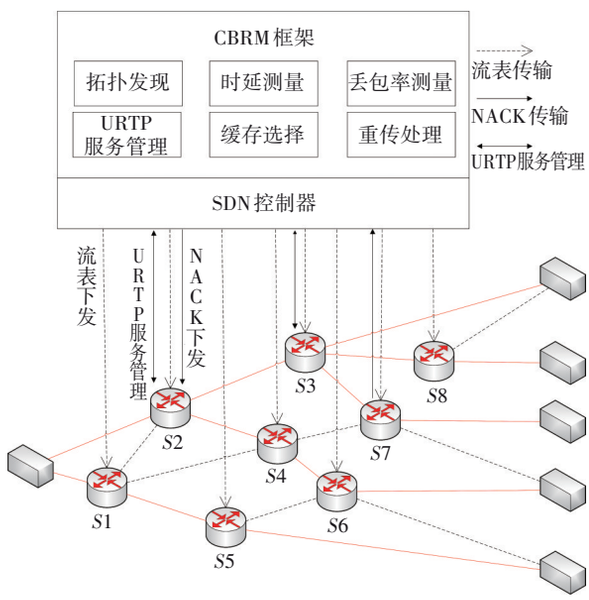
3) 设计了基于UDP的可靠传输协议URTP,该协议保证了组播通信的可靠性;

4) 在Mininet仿真环境下进行了验证,实验结果表明CBRM框架及基于缓存的可靠组播算法能以较低的代价保证组播数据的可靠性.

1 CBRM框架

图1为CBRM框架的结构示意图. CBRM框架包含资源测量和可靠性保障两部分.框架的资源测量部分由网络拓扑发现模块、链路时延测量模块和链路丢包率测量模块组成.其中拓扑发现模块通过控制器向各台交换机下发LLDP(link layer discovery protocol)[13]数据包,获得交换机与交换机、交换机与终端间的连接情况;时延测量模块与丢包率测量模块通过控制器发送时延测量包和丢包率测量包进行链路资源状态获取.

框架的可靠性保障部分由URTP服务管理模块、缓存选择模块和重传处理模块组成,用于保障组播数据传输的可靠性.URTP服务管理模块通过识别服务标识字段为组播提供可靠性传输服务;缓存选择模块根据网络拓扑和链路信息构造拓扑矩阵,以此为基础生成1棵在发送者与接收者间传输组播数据的组播树(图1中的实线);重传处理模块负责处理节点发送的重传请求.



注:NACK为否定确认(negative acknowledgement)

图1 CBRM框架

Fig. 1 CBRM architecture

在缓存选择模块中,使用缓存节点选择算法选择缓存节点,使用组播构建算法保证组播传输的实时性.

2 SDN中基于缓存的可靠组播算法

本节基于CBRM框架的组播树[7] T_{M},提出缓存节点选择算法、组播树构建算法和URTP.

由组播源S途经中间节点与缓存节点,到达接收者的组播树可以表示为

$$T_{\mathrm{M}}=(S,M,C,R). \tag{1}$$

式中:S代表传输组播数据的组播源;$R=\{r_1,\ldots,r_n\}$代表组播中的接收节点集合;$C=\{c_1,\ldots,c_n\}$代表组播拓扑中缓存节点的集合;$M=\{m_1,\ldots,m_n\}$代表组播拓扑中的中间节点集合,且$C\subset M$.

组播树的路由成本$c(T_{\mathrm{M}})$如式(2)所示,路由成本等于组播树中所有链路的延时h的总和,

$$c(T_{\mathrm{M}})=\sum_{l_{i,j}\in T_{\mathrm{M}}}h(l_{i,j}). \tag{2}$$

式中:$l_{i,j}$为交换机v_i和交换机v_j间的链路.

所有节点的重传成本 γ 为

$$\gamma(T_{\mathrm{M}})=\sum_{v\in C}h(p_v). \tag{3}$$

式中：v 为 $C\cup R$ 中的任意节点；p_v 为 v 的上游缓存节点到 v 的重传路径.

CBRM 框架构造的组播树 T_{M} 要保证组播源 S 到接收节点 r 有且只有 1 条路径，组播树构建应满足

$$\sum_{v\in N_s}\pi_{s,r}(s,v)=1,\ \sum_{u\in N_r}\pi_{s,r}(u,r)=1,\ \forall r\in R. \tag{4}$$

式中：N_s，N_r 分别为源节点 s 和接收节点 r 直连的邻居节点集合；$\pi_{s,r}(s,v)$ 表示源节点 s 与节点 v 间的连接是否在源节点 s 到接收节点 r 的路径上.

路径上的中间节点 v 应满足出度与入度为 1，即

$$\sum_{u\in N_v}\pi_{s,r}(u,v)=\sum_{u\in N_v}\pi_{s,r}(v,u),\ \forall r\in R. \tag{5}$$

假设组播拓扑的中间节点 i 与中间节点 j 间链路的丢包率为 $\beta_{i,j}$. 选择缓存节点后，组播中源节点到目的节点处的丢包率为

$$\beta_{s,d}=1-(1-\beta_{s,c_1})\cdot(1-\beta_{c_n,d})\cdot\prod_{i=1}^{n-1}(1-\beta_{i,\ i+1}). \tag{6}$$

式中：β_{s,c_1}，$\beta_{c_n,d}$，$\beta_{i,i+1}$ 分别为源节点 s 到下游缓存节点 c_1、缓存节点 c_n 到目的节点 d、相连缓存节点的丢包率.

在组播树进行缓存节点选择过程中，要满足在一定缓存节点数量下，组播数据经过 u 传到 v，当 u 为缓存节点时，节点 v 处的恢复成本应至少为 $c_{u,v}\cdot\varepsilon_{u,v}$；当 u 不是缓存节点时，节点 v 的恢复成本应至少为 $\gamma_u+c_{u,v}\cdot\varepsilon_{u,v}$，即

$$\sum_{v\in V}\rho_v\leqslant\gamma, \tag{7}$$

$$c_{u,v}\cdot\varepsilon_{u,v}-(2-\varepsilon_{u,v}-\rho_u)\cdot c_{\mathrm{T}}\leqslant\gamma_v, \tag{8}$$

$$\gamma_u+c_{u,v}\cdot\varepsilon_{u,v}-(1-\varepsilon_{u,v}+\rho_u)\cdot c_{\mathrm{T}}\leqslant\gamma_v. \tag{9}$$

式中：c_{T} 为组播树的路由总成本；$c_{u,v}$ 为节点 u 到 v 的传输成本；$\varepsilon_{u,v}$ 表示 u 与 v 直连边是否在组播树 T_{M} 中；ρ_u 和 ρ_v 分别表示 u 和 v 是否为缓存节点；γ_u 和 γ_v 分别为 u 和 v 的重传成本.

此外，还要保证组播传输成本和数据重传成本最小，即

$$\min\left\{\sum_{\mathrm{edge}_{u,v}\in E}c_{u,v}\cdot\varepsilon_{u,v}+\sum_{r\in R}\gamma_r+\sum_{v\in C}\rho_v\cdot\gamma_v\right\}. \tag{10}$$

式中：E 为 T_{M} 的边的集合；第一项为 T_{M} 的传输成本；第二项为接收节点的重传成本；第三项为缓存节点的重传成本.

上述问题可以转化为生成最小组播成本和最小重传成本的组播树问题. 针对最小组播成本问题使用最短路径算法可将重构后的组播树路由成本降到最小. 为解决重传成本问题，首先使用最小重传成本求解（minimum retransmission cost solution，MRCS）算法，该算法利用式（11）作为状态转移方程：

$$\gamma_x^k\left(\bigcup_{i=1}^{j}T_v^i\right)=\min_{\substack{x^*\in[0,x]\\k^*\in[1,k]}}\left\{\gamma_{x-x^*}^{k-k^*}\left(\bigcup_{i=1}^{j-1}T_v^i\right)+\gamma_{x^*}^{k^*}(T_v^j)\right\}. \tag{11}$$

式中：$\gamma_x^k(T_v^i)$ 表示以 v 为根节点且经过子节点 i 的子树 T_v^i 的总重传成本；k 为子树节点个数；x 为子树缓存节点个数.

该算法将求解组播树最小重传成本问题分解为每个子树最小重传成本的子问题，保留并重复利用子问题的解，从而得到最小重传成本. 算法具体如下：

算法 1　最小重传成本求解（MRCS）算法

输入：组播树 T_{M}，源节点 s，目的节点 D，缓存节点数量 N，自底向上遍历顺序 Order.

输出：最小重传成本 γ_{m}，缓存节点位置 P_{c}，分支记录 R_{e}，源节点管理的节点数量 K.

```
初始化 R_y, R_n, R_e, K, DP
for Node in Order do
    遍历以 Node 为根的子树的分支
    设分支中缓存节点的数量 x 为[0, N]
    记录以 Node 为根的子树的 R_y, R_n 和 P_c
    为状态转移数组 DP 赋予初始值
    for i = 2 to ChildNum do
        for x = 0 to R do
            for k = 2 to T^i_Node 的目的节点 do
                根据状态转移方程更新 DP
                R_e 记录使当前分支 DP 最小时，分支上最多能选择的缓存节点数和该分支源节点所管理的节点数
            end for
        end for
    end for
    for x = 0 to N do
        for k = 1 to T^i_Node 的目的节点 do
            用当前 DP 值更新 R_n 数组
            用 K 记录使当前 R_n 最小时源节点所管理的节点数量，对子树源节点 R_y 和 R_n 进行赋值
        end for
    end for
```

21 更新非目的节点的重传成本

22 **end for**

23 用γ_m记录求解的最小重传成本

24 **return** γ_m, P_c, R_e, K

算法1首先获取每个子树上各个分支的重传成本，R_y记录头节点是缓存节点的最小重传成本，R_n记录头节点不是缓存节点的最小重传成本，P_c记录缓存节点位置，并对状态转移数组DP赋予初始值(第2行至第6行).然后根据式(11)进行状态转移，在状态转移的过程中对R_e进行记录(第7行至第14行).根据子树中的每个分支最小重传成本计算当前子树的最小重传成本，对子树源节点的R_n和R_y进行赋值，用于下一轮的重传成本求解(第15行至第20行).完成自底向上的遍历后，便获得了组播树T_M的最小重传成本γ_m，最后返回γ_m, P_c, R_e和K.

缓存节点数量及位置确定是实现重传机制的重要基础.将算法1返回的P_c, R_e, K作为输入，使用缓存节点获取算法(getting cache algorithm，GCA)对缓存节点的分布情况进行解析，并通过递归方法获取组播树中的缓存节点集合.

算法2 缓存节点获取算法(GCA)

输入：组播树T_M，源节点s，缓存节点数量N，缓存验证记录P_c，分支管理记录R_e，子树管理记录K，缓存节点集合CacheSet

输出：缓存节点集合CacheSet

1 **if** $s \in D$ **then**

2 当访问的节点是目的节点时，递归结束

3 **end if**

4 $n \leftarrow K[s][N]$

5 **if** $n == 1$ **then**

6 **if** $P_c[s][0][N][n] == 1$ **then**

7 CacheSet.insert($T_M[s][0]$)

8 $N \leftarrow N - 1$

9 **end if**

10 GCA (T_M, $T_M[s][0]$, N, P_c, R_e, K, CacheSet)

11 **else if** $n > 1$ **then**

12 ChildNum $\leftarrow T_M[s]$.size

13 $X \leftarrow N$

14 **for** i = ChildNum **to** 1 **do**

15 $x^* \leftarrow$ Record$[s][i][X][n]$.first

16 $k^* \leftarrow$ Record$[s][i][X][n]$.second

17 GCA (T_M, $T_M[s][i]$, x^*, P_c, R_e, K, CacheSet)

18 $X \leftarrow X - x^*$

19 $n \leftarrow n - k^*$

20 **end for**

21 **end if**

算法2首先设置了递归的终点，当遍历到接收节点时本轮递归结束(第1行至第3行)；然后根据K获取源节点在选择N个缓存节点时管理的节点数量(第4行)，当n=1时，表明当前节点只有1个子节点，通过P_c判断当前节点管理n个节点且最多选择N个缓存节点时子节点是否被选为缓存节点(第6行)；如果其子节点被选为了缓存节点则将其加入到缓存节点集合CacheSet中(第7行)；否则对当前节点的子节点进行递归调用.当n>1时，表明当前节点有多个子节点，通过R_e获取每个分支能选择的缓存节点个数x^*和管理节点个数k^*，更新以当前节点为根的子树所拥有的缓存节点数和管理节点数(第11行至第19行)，然后对其所有子节点进行递归调用(第17行).当递归完成时，组播树T_M的具有最小重传成本的缓存节点集被记录在CacheSet中.

为了让缓存节点发挥作用，需要设计相应的可靠性传输协议.本文设计了基于UDP的URTP，其数据包格式见图2. 新添加的字段属性含义如表1所示.

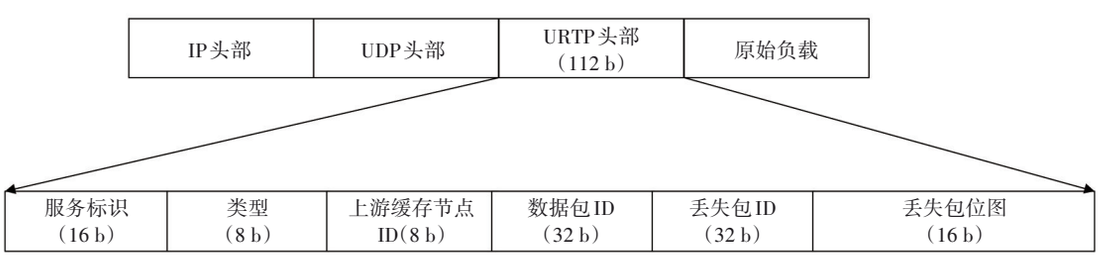


图2 URTP数据包格式

Fig. 2 URTP message format

表1　URTP协议字段属性

Table 1　URTP protocol field properties

字段属性	位数/b	字段描述
服务标识	16	判别URTP数据包是否需要提供可靠通信服务
类型	8	URTP数据的类型,包括:初始、重传、丢失等
上游缓存节点ID	8	上游缓存节点的datapath ID信息
数据包ID	32	记录数据包发送的序号
丢失包ID	32	记录丢失数据包的序号
丢失包位图	16	记录丢失包ID后续16个数据包的丢失情况

控制器通过解析服务标识字段判断是否提供可靠组播通信,类型字段为0~15,代表不同URTP数据类型,根据不同类型执行不同的操作.控制器根据上游缓存节点ID,将NACK数据包转发给对应的缓存节点.根据丢失包ID确定丢失的数据包.根据丢失包位图了解丢失数据包后续16个数据包的丢失情况.

3　实验与评估

为了验证CBRM框架中功能模块的性能以及组播数据传输的可靠性,在Linux环境下进行实验仿真,通过Ryu控制器实现SDN控制平面,用OpenVSwitch交换机搭建SDN数据平面,基于Mininet对网络环境进行搭建,使用Wireshark进行网络数据包抓取.

3.1　三种队列设计

为了平衡数据包在缓存节点中所占空间、保存时间和重传时间,设计了缓存队列、丢失队列和正序队列.

缓存节点的数据保存时间为

$$t_{\mathrm{cur}}=\tau\cdot\frac{\sum\limits_{i\in C}t_i\cdot\left|R_i\right|\cdot\max\limits_{r_j\in R}\{\beta_{i,r_j}\}}{\sum\limits_{i\in C}\left|R_i\right|\cdot\max\limits_{r_j\in R}\{\beta_{i,r_j}\}}.\tag{12}$$

式中:β_{i,r_j}为节点i到缓存节点r_j链路上的丢包率;τ是数据保存因子;R_i代表缓存节点i下游接收节点的集合.

根据式(12)可对缓存容量理想值进行估计,但实际情况中缓存容量不容易估计,进而导致丢包和缓存溢出,因此,设计了如图3所示的缓存队列结构.

采用环形队列实现缓存,在实际工作中可以根据数据传输情况实现灵活调度.当出现缓存溢出时,可以根据环形队列的结构特点占用最早到达缓存队列的数据所占用的缓存空间,从而最小化数据溢出对整体性能的影响.

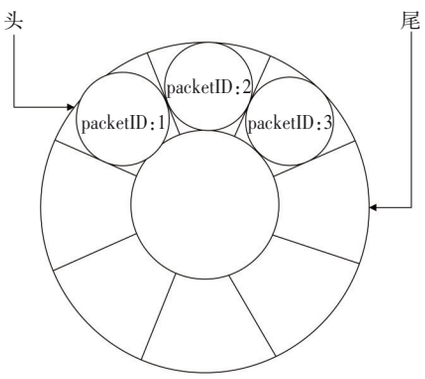


图3　缓存队列

Fig. 3　Cache queue

为了更好地记录传输过程中丢失的数据包,设计了丢失队列,丢失队列的存储结构见图4.图中maxID为链表中最大的packetID.丢失队列支持插入和删除操作,分别对应算法3和算法4.

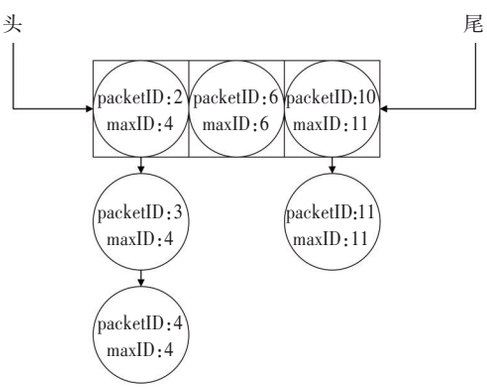


图4　丢失队列结构

Fig. 4　Lost queue structure

算法3　丢失队列插入操作

1　**for** *i*=head **to** tail **do**

2　**if** cur_maxID≥*i*.maxID **and** *i*.next！=null **and** cur_max<*i*.next.maxID **then**

3　　将packetID按顺序插入*i*节点所在链表,更新该链表所有节点的maxID

4　**else if** *i*= =tail **or** *i*= =head **then**

5　　将packetID按顺序插入*i*节点所在链表,更新该链表所有节点的maxID

```
    end if
end for
```

算法 4 丢失队列删除操作

```
if (cur_maxID, packetID) ∈ LostQueue
    if node in LinkedList 中间
        LinkedList 分离
    end if
    移除并更新链表 maxID
end if
```

因为延迟或者重传等原因,可能导致缓存队列乱序,这会增加重传时查找数据包的时间复杂度,因此设计了正序队列结构,使用插入排序提高缓存节点收到重传请求时的查找效率. 当 packetID> nextSendID时,从正序队列中查找并重传.

3.2 缓存节点选择算法评估

为了验证缓存选择算法的性能,定义了 5 种网络拓扑,通过最短路径算法处理后,其拓扑结构如图 5 所示. 基于源节点、随机缓存节点、缓存选择算法的缓存节点 3 种情况分别对图 5 的重传组播拓扑进行实验,得到重传成本如图 6 所示.

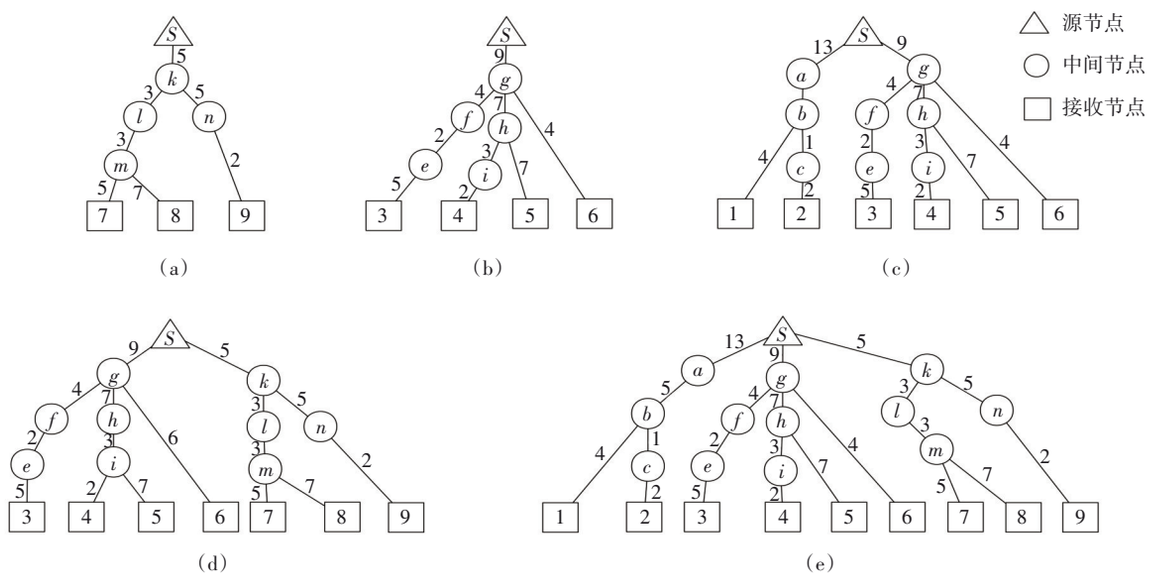


图 5 最小路由树拓扑

Fig. 5 Minimum routing tree topology

(a)—最小路由树 1;(b)—最小路由树 2;(c)—最小路由树 3;(d)—最小路由树 4;(e)—最小路由树 5.

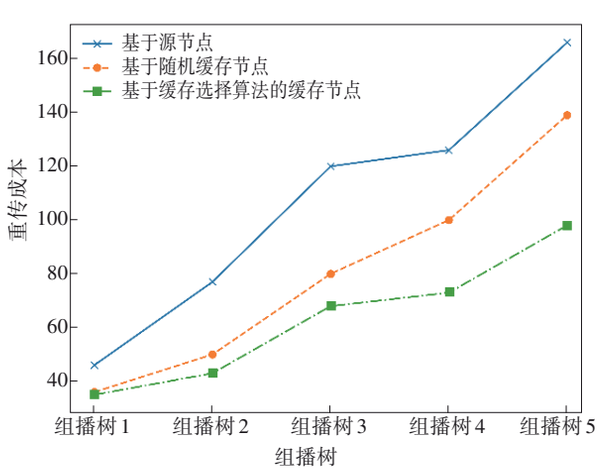


图 6 组播重传成本对比

Fig. 6 Comparison of multicast retransmission costs

由图 6 可知,基于源节点的重传组播由于不存在缓存节点,任何 1 个节点丢失的包都需要向源节点请求重发,无疑增加了重传的路径跳数,重传成本自然很高. 而基于随机缓存节点由于缺乏全局考量,重传成本相对较高. 缓存节点的存在对重传成本的影响十分明显,基于缓存节点的重传成本明显小于基于源节点的重传成本,组播拓扑的规模越大缓存节点的效果越明显,证明了网络拓扑中缓存节点的存在能够有效地降低组播树的重传成本. 同时可以发现,相对于随机选择的缓存节点,基于缓存选择算法选择的缓存节点,位置分布更加合理,能够更有效地降低组播拓扑的重传成本.

3.3 不同数量缓存节点测试对比

为了验证 CBRM 框架中缓存节点的作用,同时判断不同数量缓存节点对整体性能的影响,在图 7 中(*hi* 为主机,*Si* 为交换机)通过指定不同缓存节点数量 *N*,探究其对缓存节点选择算法的影响,并验证能够获得最小重传成本的缓存节点数

量与重传成本的关系.

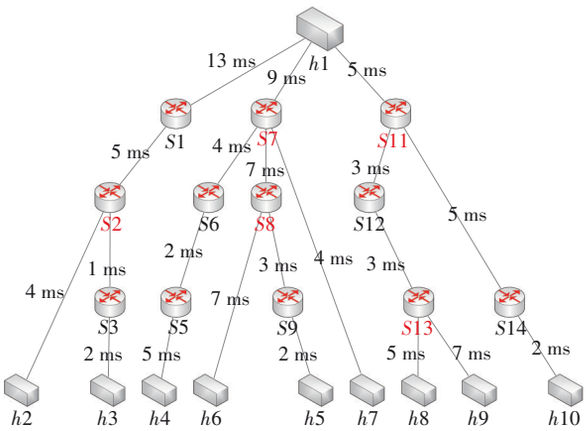


图 7　带有缓存节点的组播树

Fig. 7　Multicast tree with cache nodes

当 N 在[0,8]取值时，选择合适位置作为缓存节点对应的最小重传成本如表 2 所示.

表 2　不同 N 下的重传成本

Table 2　Retransmission costs under different N conditions

N	最小重传成本	缓存节点集合
0	166	{}
1	139	{$S7$}
2	121	{$S2$,$S7$}
3	110	{$S2$,$S7$,$S13$}
4	103	{$S2$,$S7$,$S8$,$S13$}
5	98	{$S2$,$S7$,$S8$,$S11$,$S13$}
6	98	{$S2$,$S6$,$S7$,$S8$,$S11$,$S13$}
7	98	{$S2$,$S6$,$S7$,$S8$,$S11$,$S12$,$S13$}
8	98	{$S1$,$S6$,$S7$,$S8$,$S11$,$S12$,$S13$,$S14$}

组播重传成本对比结果见图 8，基于缓存节点的重传成本明显低于基于源节点的重传成本，且当 N 为 5 时，重传成本达到最小值，在此基础上增大 N 不能进一步降低重传成本，最终选择{$S2$,$S6$,$S7$,$S8$,$S11$,$S13$}作为缓存节点集合.

3.4　不同保存因子测试对比

为找到缓存节点保存时间和存储容量最优化对应的保存因子 τ，利用缓存队列特性，通过 NACK 查找率对缓存的溢出情况进行记录. 图 9 是不同 τ 下 NACK 查找率，对缓存节点类型为 5 的 URTP 数量进行统计，发现 τ 越大，缓存节点保存时间越长，缓存容量需要设置越大. 当 τ 从 0.7 增大到 1.2 时，重传失败数据包数量明显降低；从 1.2 增大到 1.4 时，改善效果不明显. 在权衡存储空间和丢包重传效率下，在连续发送 1 000 个 300 B 大小数据包的实验环境下，选择 1.2 作为 τ 值，缓存容量为 20 kB.

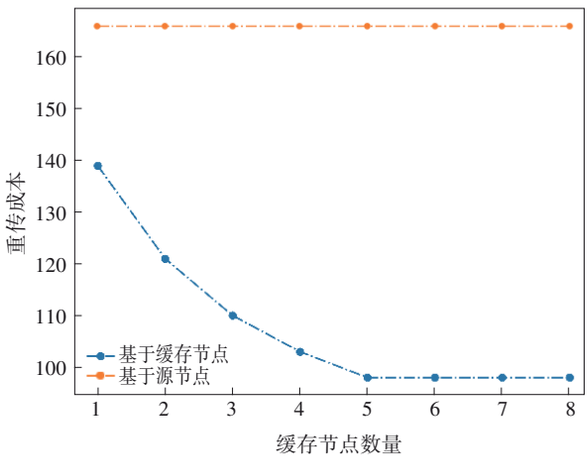


图 8　参数 N 对重传成本的影响

Fig. 8　Influence of parameter N on retransmission cost

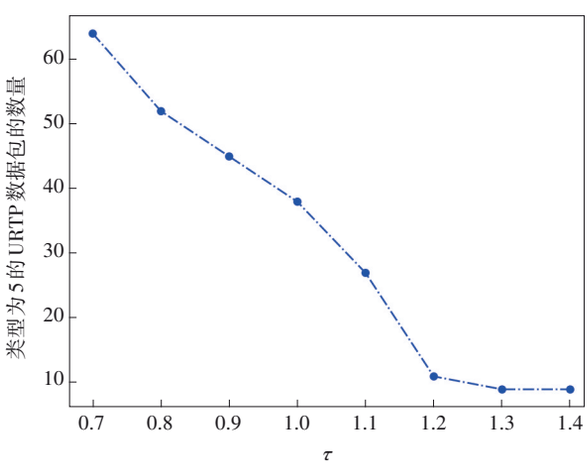


图 9　τ 对 NACK 查找率的影响

Fig. 9　The impact of τ on NACK search rate

3.5　缓存容量验证

基于缓存的组播可靠性 r_e 可以以节点 v 丢包时从 front(v)恢复的概率来度量，可表示为缓存容量 c_c、数据包丢失率 f_r 和组播传输率 t_r 以及重传策略 r_s 的非线性函数，如式(13)所示：

$$r_e = f(c_c, f_r, t_r, r_s). \tag{13}$$

其数量关系为

$$r_e \propto \frac{c_c \cdot (r_s)^{\alpha}}{t_r \cdot f_r}. \tag{14}$$

式中 α 为影响因子，随全局网络状况不同而改变.

由于故障发生的概率是随机的，为方便讨论，这里假设 f_r 服从均匀分布，即等间隔内丢包数相同. 以数据包大小为 576 B 的视频流数据包为例，分别在 2，4，8，16 Mb/s 组播传输速率和 0.01 至 0.15 的丢包率下对缓存容量及可靠性进行验

证,实验结果如图10所示.结果表明,随缓存容量的增加可靠性逐步提高;在可靠性要求确定的条件下,单位时间组播数据包数量越多要求缓存容量越大,数据包丢失率越高要求缓存容量越大;在f_r服从均匀分布且丢包率很低时,较小的缓存容量就能得到较高的可靠性.

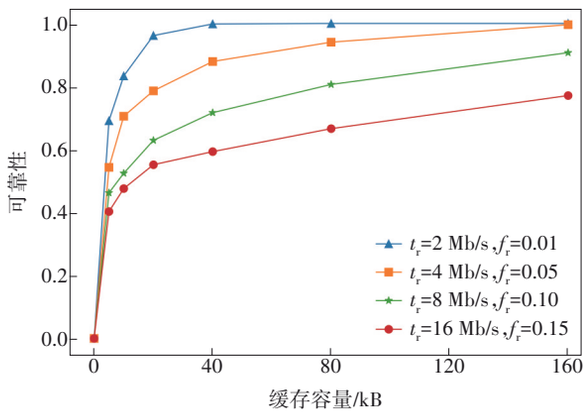


图10 缓存容量对可靠性的影响

Fig. 10 Impact of cache size on reliability

在出错数据包分布不均匀的极端情况下,缓存容量可靠性实验表明,若想满足同等期望的可靠性,缓存容量需扩展为正常情况下的2.5倍.

3.6 可靠性验证

将最小路由树(minimum routing tree,MRT)、基于边缘节点的最小路由树(EB-MRT)、基于缓存节点的最小路由树(CB-MRT)作为实验对象,图11显示通过缓存节点重传的EB-MRT和CB-MRT丢包数量有所下降,对比MRT,EB-MRT和CB-MRT丢包数量,可知缓存节点可以有效降低丢包率,通过对比EB-MRT和CB-MRT接收端的丢包情况证明了缓存选择算法的合理性.

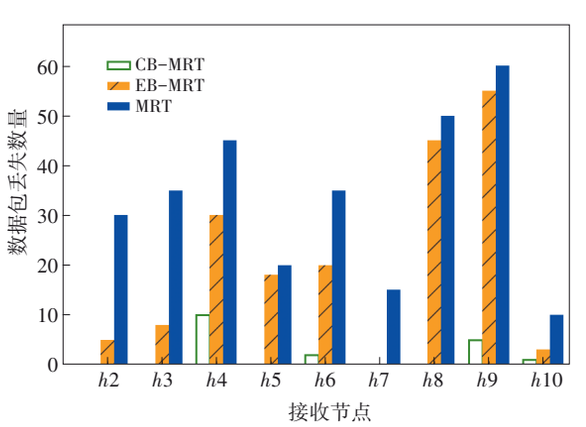


图11 接收端的丢包情况

Fig. 11 Packet loss situation of receive nodes

4 结 语

结合缓存技术与重传技术,本文提出了基于缓存节点的可靠组播(CBRM)框架;设计了最小重传成本求解算法和缓存节点获取算法,并给出了伪代码;对OpenVSwitch交换机进行功能扩展,构造了缓存队列、丢失队列和正序队列,实现了数据重排序和数据去重的功能;通过实验验证了CBRM框架、最小重传成本求解算法、缓存节点选择算法能够在保证组播可靠性的前提下通过选择合理的缓存节点集合,有效降低组播重传开销.

参考文献:

[1] Levine B N, Garcia-Luna-Aceves J J. A comparison of reliable multicast protocols[J]. *Multimedia Systems*, 1998, 6(5): 334-348.

[2] Mckeown N. Software-defined networking[J]. *INFOCOM Keynote Talk*, 2009, 17(2): 30-32.

[3] Kreutz D, Ramos F M V, Verissimo P E, et al. Software-defined networking: a comprehensive survey [J]. *Proceedings of the IEEE*, 2014, 103(1): 14-76.

[4] Zou J F, Shou G C, Guo Z G, et al. Design and implementation of secure multicast based on SDN[C]//The 5th IEEE International Conference on Broadband Network & Multimedia Technology. Guilin, 2013: 124-128.

[5] Chiti F, Fantacci R, Pierucci L. Energy efficient communications for reliable IoT multicast 5G/satellite services[J]. *Future Internet*, 2019, 11(8): 164.

[6] Mahajan K, Sharma D, Mann V. Athena: reliable multicast for group communication in SDN-based data centers[C]// The 9th International Conference on Communication Systems and Networks (COMSNETS). Bengaluru, 2017: 174-181.

[7] 徐久强,路佳熹,李鹤群,等. 基于SDN的工业物联网组播通信框架研究[J]. 东北大学学报(自然科学版),2023,44(2): 192-198.
(Xu Jiu-qiang, Lu Jia-xi, Li He-qun, et al. SDN-based multicast communication framework for industrial Internet of things[J]. *Journal of Northeastern University (Natural Science)*, 2023, 44(2): 192-198.)

[8] Chae J H, Kim N, Lee B D. The analysis of multicast tree construction scheme using reinforcement learning in SDN [J]. *The Journal of Korean Institute of Communications and Information Sciences*, 2020, 45(5): 820-827.

[9] Shi Y L, Wong J, Jacobsen H A, et al. Topic-oriented bucket-based fast multicast routing in SDN-like publish/subscribe middleware[J]. *IEEE Access*, 2020, 8: 89741-89756.

[10] Zhang X C, Yang M H, Wang L, et al. An OpenFlow-enabled elastic loss recovery solution for reliable multicast [J]. *IEEE Systems Journal*, 2016, 12(2): 1945-1956.

[11] Cui J, Kong L B, Zhong H, et al. Scalable QoS-aware multicast for SVC streams in software-defined networks [C]//2021 IEEE Symposium on Computers and Communications (ISCC). Athens, 2021: 1-7.

(下转第17页)