# A Practical Congestion Control Scheme for Named Data Networking

**摘要**

传统的拥塞控制机制是为端到端连接而设计的，不适合命名数据网络（NDN）架构，在该架构中可以通过多个路径从多个源检索内容。为了充分利用 NDN 架构，拥塞控制方案必须考虑网络内缓存、多路径转发和多播数据传输的影响。此外，解决方案不得假设已知的链路带宽或数据包大小，因为这些假设可能不适用于覆盖链路、无线链路或具有不同数据包大小的应用。在本复现的论文中，作者提出了 PCON：一种实用的拥塞控制方案来解决上述问题。PCON 基于 CoDel AQM（通过测量数据包排队时间）检测拥塞，然后通过显式标记某些数据包向客户端发出信号，以便下游路由器可以将流量转移到替代路径，客户端可以降低其兴趣发送速率。通过对该论文的复现表明，PCON 的转发自适应达到了比现有工作更高的总吞吐量，同时保持了类似的 RTT 公平性。此外，PCON 可以适应 IP 隧道和无线链路容量的变化，这是其他逐跳方案不考虑的条件

**关键词：** NDN；拥塞控制；拥塞通知

## 1 引言

命名数据网络 [5] 是一种新兴的网络架构，它将网络通信模型从 IP 的主机到主机数据包传递改变为按名称获取数据：客户端通过向网络发送兴趣包来拉取数据包。这种新模型支持多播数据传输、机会性网络内缓存和多路径转发 [5] [20]。然而，这也使网络拥塞控制变得复杂，因为现有的解决方案无法直接应用。传统的类似 TCP 的拥塞控制基于两个端点之间已建立的连接。发送方通过测量往返时间（RTT）或数据包丢失来检测拥塞，然后相应地调整其发送速率。然而，在 NDN 中，端到端连接的概念并不适用。可以沿着通往这些存储库的路径从不同的存储库或不同的高速缓存检索相同内容的数据块。由于这些不同的内容源会导致不同的检索延迟，并且客户端无法区分它们，因此传统的基于 RTT 的超时成为不可靠的拥塞指标。NDN 的状态转发平面使路由器能够通过丢弃兴趣包或将其转移到替代路径来控制每一跳的拥塞。然而，这个方向上的大多数现有解决方案（HBH 兴趣整形）[3] [18] [16] [6] [22] [21] [15] [7] 假设已知或可预测的链路带宽和数据块大小，这限制了它们在这些假设的场景中的有效性不成立。例如，最大的实验性 NDN 部署 NDN Testbed [10] 在 UDP 隧道上运行，其中底层带宽未知且不断变化；视频点播提供商可以通过动态调整视频质量（因此数据块大小）来响应拥塞。本文提出了 PCON：一种实用的 NDN 拥塞控制方案，不假设已知的链路带宽或数据块大小。PCON 路由器通过使用从 CoDel [12] [13] 扩展的主动队列管理 (AQM) 方案来检测其本地链

路上的拥塞。当检测到拥塞时，路由器通过显式标记数据包向客户端和下游路由器发出此状态信号。然后，下游路由器通过将后续兴趣包部分转移到替代路径和/或将信号传递到更下游来做出反应；客户端的反应是降低发送速率。由于网络内路由器参与避免拥塞，因此我们认为 PCON 是一种“逐跳”拥塞控制方案。然而，它与“逐跳兴趣整形”有根本的不同，因为 PCON 路由器不会根据相应数据块将占用多少链路容量的预测来拒绝兴趣。相反，它们监视传出链路上的队列，这允许及早发出拥塞信号并隐式考虑可用带宽。我们在 ndnSIM [9] 中实现 PCON，并根据文献中基于延迟的 [4] 和 HBH 兴趣整形方案 [18] 评估其性能。我们证明了 PCON 在 RTT、动态数据块大小、IP 隧道和无线链路不同的情况下可以防止拥塞。此外，PCON 的转发自适应（基于显式拥塞标记）实现了比基于均衡待处理兴趣的现有解决方案更高的吞吐量。PCON 能够适应 IP 隧道和无线链路不断变化的可用带宽，当前 HBH 兴趣整形提案未考虑这种情况。

## 2 相关工作

在过去几年中，出现了许多关于 NDN 拥塞控制的提案。Van Jacobson 的原始论文指出“每一跳都保持流量平衡” [5]，但将拥塞控制的细节留给了未来的工作。其他研究人员很快认识到，NDN 的缓存和多路径转发使得通过 RTT 测量或数据包丢失进行的传统拥塞检测不够充分，并建议以截断的数据包 [14] 或 NACK [20] 的形式使用显式拥塞通知。卡罗菲利奥等人。[4] 提出了一种基于延迟的拥塞控制方案，该方案根据过去 RTT 测量的样本触发兴趣窗口减小。为了克服混合来自不同数据包源的 RTT 测量的问题，他们引入了路由标签的概念，即每个数据包中的标签，指示数据包源自哪个节点以及它采用的路径。然而，路由标签的实用性和可扩展性受到了质疑 [11]，因为潜在路由的数量随着网络中节点的数量呈指数增长。此外，即使有了路由标签，客户端仍然不知道下一个数据包将从哪里来，这使得设置正确的超时值变得困难。作为替代方案，[17] [2] 提出了根据先前返回的数据包的标记来预测下一个数据包的来源的机制。然而，这些方案在实际网络（每个缓存的内容可能会快速变化）中的可靠性仍然是一个悬而未决的问题。NDN 拥塞控制的另一种方法是逐跳 (HBH) 兴趣整形 [3] [18] [16] [6] [22] [21] [15] [7]，基于路由器可以控制返回数据量的假设通过控制相反方向的利息发送率来实现其链接。这些方案需要明确的可用链路带宽的知识，这限制了它们对无线网络和 IP 隧道的有效性。

## 3 本文方法

### 3.1 本文方法概述

遵循这些考虑，我们设计 PCON 的原则如下：1）采用 AQM 进行早期拥塞检测，2）显式拥塞信令，3）利用多路径转发，4）特别考虑 IP 覆盖和无线链路。如图1所示，PCON 由五个组件组成：

- 拥塞检测：每个节点通过监视其传出队列来检测本地拥塞。
- 拥塞信令：检测到拥塞后，节点标记数据包以通知下游路由器和客户端。
- 客户端发送速率调整：最终客户端通过调整其利息发送速率来对拥塞信号做出反应。

- 多路径转发：下游路由器通过调整其流量分配比来对拥塞信号做出反应。
- 本地链路丢失检测：在无线和 IP 覆盖链路上，我们在本地检测数据包丢失并使用 NACK 发回信号。

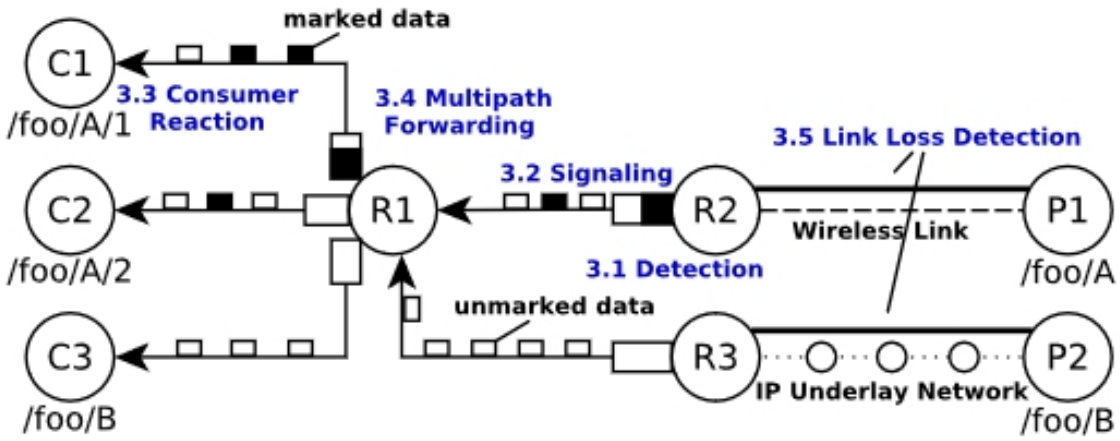


图 1. Pcon 系统架构

### 3.2 拥塞检测

正如 Nichols 和 Jacobson 所指出的 [12] [13]，检测拥塞的最可靠位置是发生拥塞的链路。因此，我们通过监视每个路由器的传出队列来检测拥塞，而不是尝试通过监视数据包丢失或往返时间来推断客户端的拥塞（如前所述，这些技术在 NDN 中不如在 IP 网络中可靠）。为了检测路由器队列上的拥塞，我们使用经过充分研究的 AQM 机制 CoDel [12] [13]，因为它允许具有大缓冲区的路由器通过在缓冲区溢出之前检测拥塞来吸收突发流量并保持队列较小。CoDel 测量每个数据包在其传出链路上的排队延迟（“停留时间”）。如果一段时间内的最短停留时间（默认：100ms）超过阈值（默认：5ms），则认为该链路拥塞。CoDel 使用排队延迟而不是队列大小符合我们未知链路容量的设计假设：排队延迟隐式考虑链路带宽（对于固定队列大小，较高的带宽会导致较低的排队延迟）。

PCON 监视下游（数据）和上游（兴趣）方向的拥塞。当兴趣大于数据包、在非对称链路上或在上行方向存在交叉流量时，可能会导致拥塞。在检测到兴趣包方向的拥塞后（如图 1 中的 R1-R2），R1 将标记该兴趣包的 PIT 条目，并考虑相应的数据进行拥塞信令。

除了监视传出队列之外，我们还使用两种替代方法来检测拥塞。第一种用于无法访问路由器队列状态的场景，例如 UDP 隧道。这里我们通过监控本地链路的数据包丢失来推断拥塞程度（参见第 3.5 节）。第二个解决了另一个可以被视为“拥塞”的问题：PIT 变得太大并有可能溢出路由器内存（类似于 TCP 发送方溢出接收方缓冲区）。就像由于链路带宽不足导致的拥塞一样，PIT 过度增长的状态可以向下游发出信号，以减少传入兴趣包的数量。然而，在本文中，我们关注与带宽相关的拥塞，并将过度增长的 PIT 的检测留给未来的工作。

### 3.3 拥塞信号

路由器检测到其传出链路之一发生拥塞后，会向客户端和路径上的所有路由器发出此状态信号。仅在下游方向需要信令，因为只有下游路由器才能减少信令路由器将接收的流量。因

此，我们通过标记 NDN 数据包而不是兴趣包来表示拥塞。

我们发出类似于 CoDel 使用 ECN 标记的拥塞信号：第一个数据包在进入拥塞状态时被标记，后面的数据包以递增的“标记间隔”进行标记（类似于 CoDel 的丢弃间隔）；标记间隔从 1.1 * CoDel 间隔（110 ms）开始 [13]，并与标记数量的平方根成反比减少，以实现客户端发送速率的线性下降。在拥塞的上游链路上，我们使用相同的标记间隔在 PIT 条目中设置标志，然后标记关联的数据包。

由于 NDN 内置的多播数据传输，一个标记的数据包可能会分支到多个客户端，这都会降低它们的速率。这正是期望的行为，因为如果任何客户端未能放慢速度，拥塞的链路根本不会看到速率降低（它只会看到更少的 PIT 聚合）。

图1显示了信号机制的一个示例。客户端 C3 从 P2 检索数据（在前缀 /foo/B 下）。由于路径上没有拥塞，C3 只接收未标记的数据包。C2 从 P1 (/foo/A/2) 检索数据并从 R2 接收拥塞标记，因为 R2 和 R1 之间的链路拥塞。C1 还从 P1 检索数据，但使用不同的前缀 (/foo/A/1)。除了从 R2 接收拥塞标记之外，C1 还从 R1 接收拥塞标记，因为链路 R1-C1 也拥塞。这使得 C1 可以调整到 R1 处的瓶颈，而 C2 可以调整到 R2 处的瓶颈。

### 3.4 客户端发送速率调整

一旦拥堵标志到达客户端手中，客户端就必须决定如何调整其费率。我们使用拥塞窗口（指定传输中的兴趣包的最大数量），该窗口在未标记的数据包上增加，在标记的数据包、NACK 和超时上减少。数据包充当兴趣包的选择性确认，这意味着每个数据包都会确认一个相应的兴趣包。选择性 ACK 需要在我们的设计中进行一些思考，因为 NACK/超时经常突发发生（在缓冲区溢出时）。如果我们对突发中的每个数据包丢失执行乘法减少，则在单个拥塞事件之后拥塞窗口会大幅减少（通常回到 1）。我们通过应用“基于 TCP SACK 的保守丢失恢复算法”[1]（参见算法1）来防止这种过度调整：客户端在每个 RTT 中最多执行一次窗口减少。

**Algorithm 1** NDN Conservative Window Adaptation

```
function OnData(dataPkt,seq)
if dataPKt.isMarked() then
  windowDecrease();
else
  windowIncrease();
end if
end function =0
```

我们不需要将“保守丢失适应算法”应用于标记的数据包，因为 CoDel 标记已经避免了标记突发。此外，CoDel 标记会适应拥塞程度（当拥塞持续存在时，标记会变得更加频繁），而丢失自适应算法则不会（它始终允许每个 RTT 最多减少一个窗口）；因此禁用它可以让客户端对高度拥塞的链路做出更快的反应。我们仍然对 IP 底层网络中的超时（由缓冲区溢出导致）和缓冲区溢出导致的 NACK 使用保守的丢失自适应。

通过保守的损耗自适应和 CoDel 标记，PCON 可以实现许多经典的基于损耗的 TCP 算法，如 Taheo、Reno、New Reno、HTCP、HSTCP、BIC 和 CUBIC。与传统 TCP 的唯一区别在于，窗口减小不仅由超时触发，还由数据包标记（类似于 TCP ECN 标记）和 NACK 触

发。在尝试了 TCP Reno 后，我们切换到 TCP BIC [19]，因为（类似于 TCP CUBIC）它在高 BDP 网络上表现更好。

当路由器缓冲区足够大时，PCON 会消除传统的数据包丢失来源：1) 路由器缓冲区溢出（它通过显式发出拥塞信号来防止它们），2) 通过 AQM 方案丢弃（它改为标记数据包），以及 3) 丢弃“链接”，如无线或 UDP 隧道（它通过网络内重传来恢复丢失或发送显式 NACK。因此，数据包丢失的唯一原因应该是故障（主机或链路故障）或配置错误。这为缓存和多路径引起的高 RTT 方差提供了解决方案：我们可以使用更高的 RTO 来避免误报超时，并且仍然实现快速重传，因为可以避免或明确发出丢包信号。

# 4 复现细节

## 4.1 与已有开源代码对比

Pcon 在 github 上开源了其代码，其代码可按功能分解为如下模块：

- 数据包标签（包含拥塞标签，NACK 标签）：CongestionTag.h, NS3CCTag.h
- CODEL AQM（包含队列检测，排队时延计算）：Codel-queue2.h
- 转发策略辅助模块（主要用于从 CODEL 模块中获取拥塞情况）：str-helper.h
- Pcon 转发策略模块（根据辅助模块获取的拥塞情况对转发接口与数据包进行拥塞标记）：pcon-strategy.h
- 客户端模块（主要根据获得的数据包的拥塞标记情况进行拥塞窗口调整）：ndn-consumer-pcon.h

在 Pcon 开源代码的基础上，我参考了 FleCom [8] 的设计思路（未开源），引入了拥塞概率 $p$ 与调节拥塞窗口大小的因子 $C_{cwnd}$，对 Pcon 拥塞窗口调整算法进行了改进。首先是关于拥塞概率的计算，拥塞概率 $p$ 由当前排队时延 $d_{cur}$、参考排队时延 $d^{ref}$（默认为 5ms）和上一周期的排队时延 $d^{cur}_{k-T}$ 相结合得到：

$$p_k = p_{k-T} + \alpha \cdot (d^{cur}_k - d^{ref}) + \beta \cdot (d^{cur}_k - d^{cur}_{k-T}) \tag{1}$$

其中参数 $\alpha$ 和 $\beta$ 为比例因子，单位均为 Hz。参数 $\alpha$ 决定了当前时延与目标值的偏差如何影响拥塞概率；$\beta$ 根据延迟趋势是上升还是下降来施加额外调整的量 [14]。请注意，$p$ 的范围被强制限制为 0 到 1。如果 $p \leq 0$，则 $p$ 设置为 0。如果 $p > 1$，则 $p$ 设置为 1。

为了缓解拥塞，本地概率 $p_k$ 必须小于或等于 0。因此，当前节点的期望排队延迟设置为 $d^{cur}_{k+T}$ ，计算公式为

$$0 = p_k + \alpha \cdot (d^{cur}_{k+T} - d^{ref}) + \beta \cdot (d^{cur}_{k+T} - d^{cur}_k) \tag{2}$$

$$d^{cur}_{k+T} = \frac{\alpha \cdot d^{ref} + \beta \cdot d^{cur}_k - p_k}{\alpha + \beta} \tag{3}$$

注意，$d_{k+T}^{cur}$ 必须是 $d_{k+T}^{cur}$ 、$d_k^{cur}$ 和 $d^{ref}$ 中的最小值，以保证稳定的局部非拥塞状态。最终调整后的窗口大小定义为 $cwnd_{final}$，当前窗口大小定义为 $cwnd_{now}$，则 $C_{cwnd}$ 可以通过以下公式获得:

$$cwnd_{final} = cwnd_{now} + \frac{d_{k+T}^{cur} - d_k^{cur}}{d_k^{cur}} \cdot 1 \cdot cwnd_{now} \tag{4}$$

$$C_{cwnd} = \frac{d_{k+T}^{cur} - d_k^{cur}}{d_k^{cur}} = \frac{\alpha \cdot (d^{ref} - d_k^{cur}) - p_k}{(\alpha + \beta) \cdot d_k^{cur}} \tag{5}$$

以上计算被我加入到 Codel-queue2.cpp 文件中, 并在 pcon-strategy.cpp 中的 beforesatisfyInterest() 函数中将结果 $C_{cwnd}$ 加入到数据包中。最后在 ndn-consumer-pcon.cpp 的 Ondata () 函数中根据数据包的拥塞标记与 $C_{cwnd}$ 的值执行我新设计的拥塞窗口调整算法，如下所示:

**Algorithm 2** My Conservative Window Adaptation

**function** OnData(dataPkt,seq)
**if** dataPKt.isMarked() and !NACK and !TimeOut **then**
  **if** dataPKt.$C_{cwnd} < 0$ **then**
    $Cwnd = Cwnd - Ccwnd$;;
  **end if**
**end if**
**if** NACK or TimeOut **then**
  windowDecrease();
**else**
  windowIncrese();
**end if**
**end function** =0

下面是两个关键的代码改动部分:

```
  }
  else {
    if (CoDelTimeAfter(now, (m_firstAboveTime + Time2CoDel(m_interval)))) {
      /* Queue has been over limit for longer than interval!
       * Set class state to mark next packet:

      m_sojourn = delta;
      m_overTargetForInterval = true;*/
      m_sojourn = delta;
      m_overTargetForInterval = true;
      probability = probability+0.125*(delta.GetSeconds()-m_target.GetSeconds
())+1.25*(delta.GetSeconds()-sojournTime_before);
      if(probability<=0)  probability = 0;
      if(probability>=1)  probability = 1;
       Cwnd = (0.125*(m_target.GetSeconds()-delta.GetSeconds())-probability)/
(1.375*delta.GetSeconds());
      sojournTime_before = delta.GetSeconds();
    }
  }
}
```

图 2. codel-queue2.cpp 改动部分

```
ConsumerCC::bicDecrease(bool resetToInitial)
{
  double Cwnd = CoDelQueue2::Cwnd;
  std::cout<<"get in decrease and cwnd = "<<Cwnd<<std::endl;
  // BIC Decrease
  if (congstate && Nackstate != NACK_TYPE_LL_MARK) {
    congstate = false;
    if(Cwnd < 0) {
        m_cwnd = m_cwnd +Cwnd;
    }
  }
  else if (m_cwnd >= LOW_WINDOW) {
        auto prev_max = bic_max_win;
        bic_max_win = m_cwnd;
        m_cwnd = m_cwnd * m_beta;
        bic_min_win = m_cwnd;
        if (prev_max > bic_max_win) //Fast. Conv.
          bic_max_win = (bic_max_win + bic_min_win) / 2;
        bic_target_win = (bic_max_win + bic_min_win) / 2;
      }
      else {
            // Normal TCP
          m_sstresh = m_cwnd * m_beta;
          m_cwnd = m_sstresh;
      }
```

图 3. 拥塞窗口调整算法改动部分

## 4.2 实验环境搭建

本次复现使用 ubuntu16.04 + ndnSIM 2.1 [9] 评估 PCON 和我的设计，以证明决策的有效性。

ndnsim 是一个基于 ns-3 仿真环境建立的用于 NDN 仿真的测试环境，该环境集成了相当多 NDN 的基本功能，如 PIT,FIB, 兴趣包, 数据包以及基本的网络拓扑结构和转发策略：

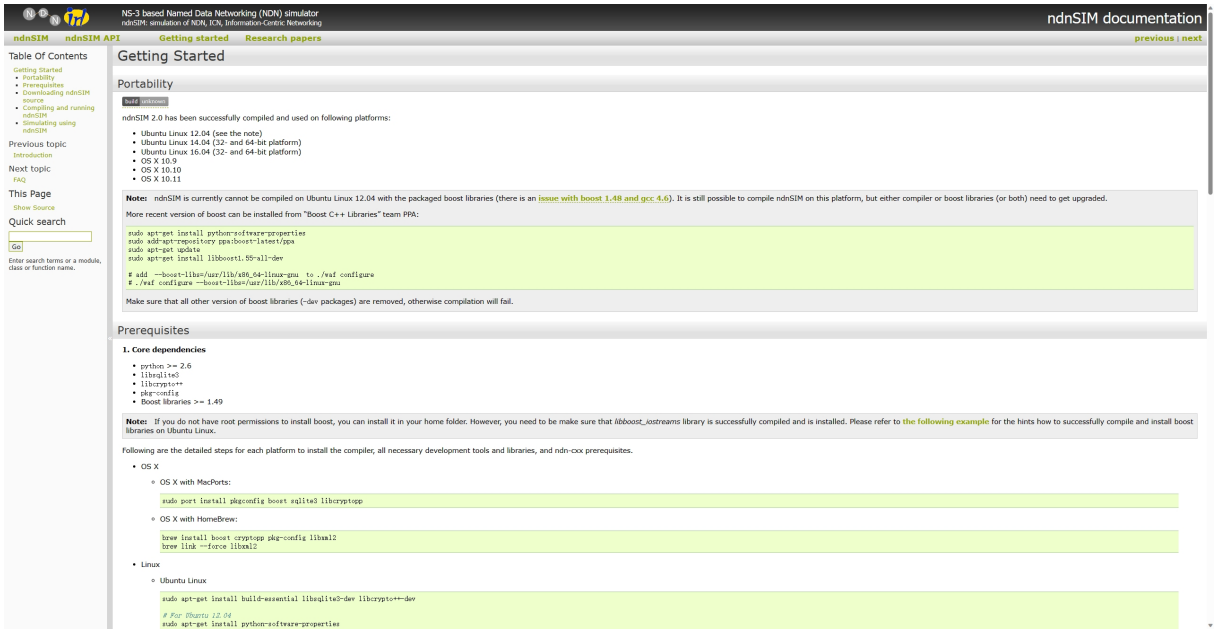


图 4. ndnsim2.1 下载教程

## 4.3 使用说明

首先在 ubuntu16.04 系统上根据 ndnsim2.1 版官方教程下载依赖的环境配置，之后将本人的 git 代码下载至本地后跳转至 ns3 文件夹, 并依次执行如下命令：

make configure

make

上述为编译命令，执行成功后会在 ns3 文件夹下生成 build 文件夹。之后可以执行以下代码查看 Pcon 的运行结果（run 命令前是由两个短横线组成请注意）：

./waf –run pcon_scen1

./waf –run pcon_scen2

./waf –run pcon_scen3

./waf –run pcon_scen4.1

./waf –run pcon_scen4.2

结果在 ns3 下的 result 文件夹中可以查看，若想可视化结果需编写 R 语言程序对 txt 文件进行可视化。本次对比我的拥塞窗口调整算法与 Pcon 的算法使用的场景是 pcon_scen1。

## 4.4 创新点

我认为，排队时间超过阈值只代表着一定概率发生拥塞，而不是实际发生拥塞，因此简单地根据排队时间记录拥塞标记的真假，之后客户端检测到拥塞标记后直接进行 TCP 拥塞

控制算法是不合适的。这将过于激进地减小拥塞窗口的大小，从而导致发送速率不必要的降低。

以上述观点为前提，我的算法将拥塞标记与 NACK 和超时区分开来。只有当传回的数据包只有拥塞标记时（没有 NACK 或超时发生），算法才会按照我们设计的方式减小拥塞窗口（如算法2所示），否则它也会像 Pcon 一样执行 TCP BIC 算法。

## 5 实验结果分析

对 Pcon 的复现以及自己的改进与 Pcon 的对比均在如图??的网络拓扑结构下执行，因为在该场景下不会发生 NACK 与超时情况，可以方便的对比我的改进与 Pcon 原来的拥塞窗口控制算法的性能表现。

我们首先评估使用 AQM 方案 (CoDel) 和显式拥塞信号的效果。我们使用具有单个客户端且无缓存线性 3 节点拓扑（图 5）。

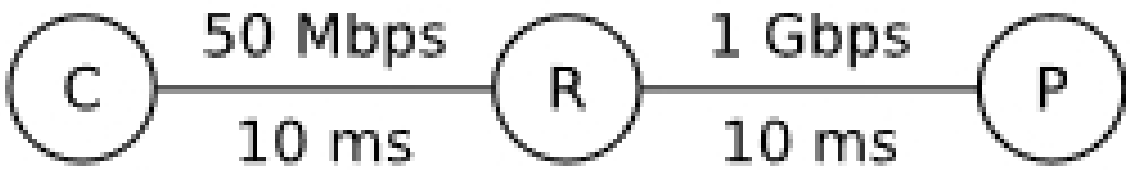


图 5. Topo1

我们比较三种不同的情况：

- FIFO：TCP BIC 的简单实现，具有 FIFO 队列和使用保守丢失自适应的客户端。
- CODEL：路由器使用 CoDel 丢弃队列而不是 drop-tail FIFO；客户端不使用保守的损失适应。
- PCON：作为 CODEL 加上显式拥塞信令。在所有场景中，我们都使用 TCP BIC 窗口自适应和传统的 TCP RTO 估计（由于没有缓存多样性或多路径，因此执行正常）。BIC 使用“慢启动”阶段和预先配置的“最大增量”，指定每个 RTT 的 cwnd 最大增加量（此处：16 个兴趣）。丢失事件后，它使用“二分搜索”快速占用可用带宽，类似于初始慢启动阶段（有关详细信息，请参阅 [19]）。丢失事件由超时（FIFO、CODEL）或标记数据包（PCON）触发。

结果如图6、图7和图8所示。我们发现，即使在这种简单的场景中，使用尾部队列 (FIFO) 的超时也是有问题的：它们总是需要完全填满队列，可能会导致持久队列，并增加重传的延迟。CoDel 和 PCON 的优点是它们在队列达到其限制（此处为 100 个数据包）之前做出反应。将队列大小增加到 1000 个数据包不会对 CoDel/PCON 产生影响，但会严重增加 FIFO 队列的排队延迟。此外，PCON 的显式标记方案实现了与其他方案相同的吞吐量，但避免了数据包丢失，从而避免了重传延迟（延迟图中的尖峰表示重传）。

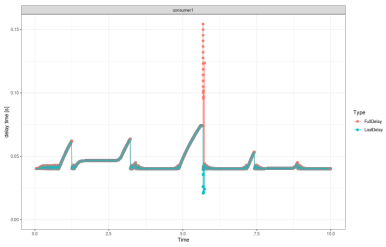

图 6. FIFO

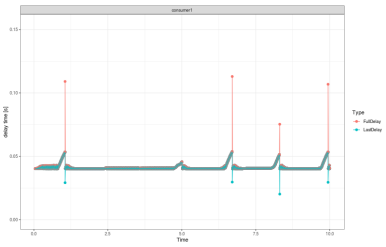

图 7. CODEL

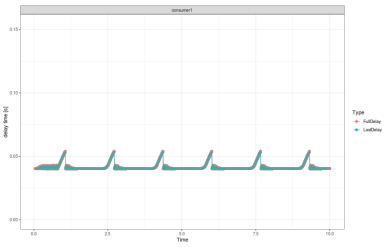

图 8. Pcon

接着是对比改进之后的拥塞调整算法与 Pcon 原本的设计在延迟方面以及拥塞窗口的变化的表现，如图9，图10，图11所示。可以看到，在完全不影响延迟表现的情况下，改进之后的算法相比原来大幅缩减了检测到拥塞标志时减少拥塞窗口大小的幅度。证明了该改进的可行性。

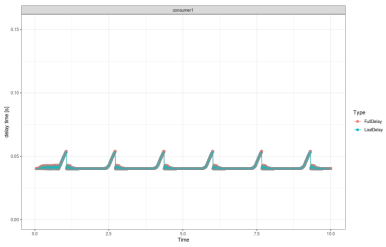

图 9. Pcon 的延迟

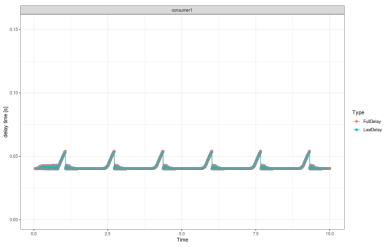

图 10. 改进后的延迟

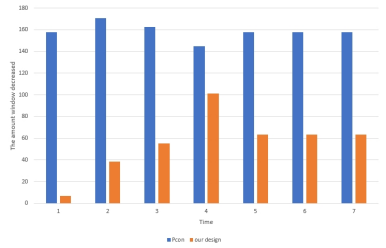

图 11. 拥塞窗口减少幅度

## 6 总结与展望

本文复现了 Pcon 的工作，并在 Topo 1 中验证了其在时延方面相对于 FIFO 和 CODEL 具有绝对的性能优势。同时，我改进了 Pcon 的拥塞调整算法，实验结果表明在只有拥塞标记的情况下，改进的算法可以显著减少拥塞窗口的减少量，同时保证与 Pcon 相同的时延性能。

由于 NDN 多路径传播的特点，在今后的工作中，可以根据拥塞概率进行更细粒度地调整每条路径的传输概率，从而充分利用拥塞标记中包含的拥塞概率信息。同时进一步优化拥塞窗口调整算法，目前在未检测到拥塞标志时对拥塞窗口执行传统的 TCP 增加拥塞窗口大小的算法。但未检测到标志时拥塞概率仍客观存在。可根据前文提到的 $C_{cwnd}$ 细化增加窗口的方式。

## 参考文献


[1] Ethan Blanton. A conservative sack-based loss recovery algorithm for tcp. *Internet draft, draft-allman-tcp-sack-07. txt*, 2001.

[2] Stefan Braun, Massimo Monti, Manolis Sifalakis, and Christian Tschudin. An empirical study of receiver-based aimd flow-control strategies for ccn. In *2013 22nd international conference on computer communication and networks (ICCCN)*, pages 1–8. IEEE, 2013.

[3] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks. *ACM SIGCOMM Computer Communication Review*, 42(4):491–496, 2012.

[4] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Optimal multipath congestion control and request forwarding in information-centric networks: Protocol design and experimentation. *Computer Networks*, 110:104–117, 2016.

[5] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12, 2009.

[6] Kai Lei, Chaojun Hou, Lihua Li, and Kuai Xu. A rcp-based congestion control protocol in named data networking. In *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 538–541. IEEE, 2015.

[7] Chengcheng Li, Tao Huang, Renchao Xie, Hengyang Zhang, Jiang Liu, and Yunjie Liu. A novel multi-path traffic control mechanism in named data networking. In *2015 22nd International Conference on Telecommunications (ICT)*, pages 60–66. IEEE, 2015.

[8] Zhuo Li, Hao Xun, Yang Miao, Weizhe Zhang, Peng Luo, and Kaihua Liu. Flecom: A flexible congestion control protocol in named data networking. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2023.

[9] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim 2.0: A new version of the ndn simulator for ns-3.

[10] Named Data Networking (NDN). Ndn testbed - named data networking (ndn), 2024. [Accessed: 12 January 2024].

[11] Dinh Nguyen, Masaki Fukushima, Kohei Sugiyama, and Atsushi Tagami. Efficient multipath forwarding and congestion control without route-labeling in ccn. In *2015 IEEE international conference on communication workshop (ICCW)*, pages 1533–1538. IEEE, 2015.

[12] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[13] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Jana Iyengar. Controlled delay active queue management. Technical report, 2018.

[14] Sara Oueslati, James Roberts, and Nada Sbihi. Flow-aware traffic control for a content-centric network. In *2012 Proceedings IEEE INFOCOM*, pages 2417–2425. IEEE, 2012.

[15] Heungsoon Park, Hoseok Jang, and Taewook Kwon. Popularity-based congestion control in named data networking. In *2014 sixth international conference on ubiquitous and future networks (ICUFN)*, pages 166–171. IEEE, 2014.

[16] Natalya Rozhnova and Serge Fdida. An extended hop-by-hop interest shaping mechanism for content-centric networking. In *2014 IEEE global communications conference*, pages 1–7. IEEE, 2014.

[17] Lorenzo Saino, Cosmin Cocora, and George Pavlou. Cctcp: A scalable receiver-driven congestion control protocol for content centric networking. In *2013 IEEE international conference on communications (ICC)*, pages 3775–3780. IEEE, 2013.

[18] Yaogong Wang, Natalya Rozhnova, Ashok Narayanan, David Oran, and Injong Rhee. An improved hop-by-hop interest shaper for congestion control in named data networking. *ACM SIGCOMM Computer Communication Review*, 43(4):55–60, 2013.

[19] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524. IEEE, 2004.

[20] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Computer Communications*, 36(7):779–791, 2013.

[21] Feixiong Zhang, Yanyong Zhang, Alex Reznik, Hang Liu, Chen Qian, and Chenren Xu. A transport protocol for content-centric networking with explicit congestion control. In *2014 23rd international conference on computer communication and networks (ICCCN)*, pages 1–8. IEEE, 2014.

[22] Jianer Zhou, Qinghua Wu, Zhenyu Li, Mohamed Ali Kaafar, and Gaogang Xie. A proactive transport mechanism with explicit congestion notification for ndn. In *2015 IEEE international conference on communications (ICC)*, pages 5242–5247. IEEE, 2015.