摘要:数据中心网络传输旨在实现低延时和无损行为。其中，RTT 作为数据包往返时间，被证明和交换机队列长度有着很大的关联程度，是网络终端节点能够获取的为数不多的拥塞信号。目前随着网络硬件的进步，RTT 的测量已经能够达到微秒级的精度。本论文设计了一种神经网络拥塞控制算法，充分利用了数据包 RTT 时延信息，对发送速率进行调整，保持低延时和高吞吐量。并在基于 NS3 搭建的仿真平台上部署该算法进行验证。设计容易发生拥塞的 INCAST 通信模式，通过实验表明，它提供了很好的拥塞控制效果，模拟通信平均时延控制在 10 微妙以下，在大规模数据注入时计时预测并缓解拥塞。同时该算法不依赖于网络硬件的修改，便于在数据中心等大规模网络环境内部署。
关键词：神经网络、拥塞控制、数据中心网络、INCAST、RTT

Abstract: Data center network transmissions are designed for low latency and lossless behavior. Among them, RTT, as the round-trip time of packets, proves to have a great correlation with the length of the switch queue, and is one of the few congestion signals that network endpoints can obtain. At present, with the advancement of network hardware, RTT measurement has been able to achieve microsecond accuracy. In this paper, a neural network congestion control algorithm is designed, which makes full use of the RTT delay information of packets, adjusts the transmission rate, and maintains low latency and high throughput. The algorithm is deployed on a simulation platform based on NS3 for verification. The INCAST communication mode prone to congestion is designed, and experiments show that it provides a good congestion control effect, and the average delay of analog communication is controlled below 10 subtlety, which predicts and alleviates congestion when large-scale data injection is timed. At the same time, the algorithm does not depend on the modification of network hardware, which is convenient for deployment in large-scale network environments such as data centers.
Keywords: neural networks, congestion control, data center networks, INCAST, RTT

目录

绪论

1.1 研究背景

根据 2021 年颁布的《数据中心智能无损网络白皮书》，数据中心网络（DCN, Data center network）支持很多类型的业务，包括大数据机器学习、视频、语音等；随着人工智能在数据中心的发展和应用，数据中心东西向传输的业务数据呈指数级的增长，海量数据的传送使网络承受更大的传输压力，RDMA（新远程直接内存访问，Remote Direct Memory Access）协议的使用，使得数据发送的瓶颈逐渐从终端转移到网络设备上，低时延和无损行为成为了数据中心网络新的必要需求。

云时代的数据中心专注于应用转型和服务的快速部署。在 AI 时代，数据中心提供了实现数字化生活所需的信息和算法。高速存储和人工智能分布式计算的结合，将大数据转化为快速数据，供人、机、物访问。高性能、大规模、无丢包的数据中心网络对数字转换的顺利进行至关重要。

人工智能、网络性能等高性能应用的关键指标包括吞吐量、时延和拥塞。吞吐量是指快速传输大量数据的网络总容量。时延是指跨数据中心网络事务的总延迟。当流量超过网络容量时，会发生拥塞。丢包是严重影响吞吐量和时延的因素。为应对这种变化趋势，数据中心使用了 RDMA 协议消除了数据副本，释放了 CPU 资源，能够完成路径选择和取出顺序计算，但是 RDMA 效率的提高给网络带来了更大的压力，将瓶颈转移到数据中心网络基础设施上，这使得网络比以往更容易发生拥塞，而传统拥塞控制算法已经无法适应目前的数据中心数据传输，因此针对数据中心网络设计新的拥塞控制算法和机制就非常重要。

1.2 研究现状及趋势

数据中心无损网络的拥塞控制，目前已提出了多种解决思路和方法，大体方向一共有三种，分别是仅通过端节点的拥塞控制算法、需要网络中交换机或控制器配合的流调度和负载均衡；通过端节点的拥塞控制算法又包含了基于丢包的 TCP Cubic 算法、基于 ECN 丢包的 DCQCN[2]和 DCTCP[3]算法、基于 RTT 信息的 TIMELY[4]算法等，流调度包括了基于截止时间的 D2TCP[5]、基于流大小的 pFabric[6]等，负载均衡包括了基础简单算法 ECMP 和复杂算法 Hedera[7].大部分拥塞控制算法由于需要特殊的硬件或者报文支持，如 HPCC[8]算法需要 INT 报文字段，阿里提出的 uFAB[9]需要能够发送特殊探针的边界路由器。

拥塞控制研究发展有以下几个方向，一是基于 SDN 的中心化控制，集中式控制可以对网络全局信息进行掌控，更能精确的规划控制流量，但同时面临着可部署规模的问题；二是人工智能与流量控制技术结合的研究方向，人工智能具有强大的自适应力和自学能力，能为拥塞控制提供一套有效的决策工具[10]，例如华为在 2021 年提出的 ACC[11]中提出了强化学习的拥塞控制算法等。

1.3 论文研究意义及创新

由于数据中心网络的数据传送的突发性、以及流种类的复杂性，目前的拥塞控制算法和机制主要有两个特点（缺点）：一是具有针对性，对于特定情况或特定流的拥塞处理较好，但是对于网络中的突发流量导致的拥塞情况效果较差；二是大部分算法都需要专门的硬件配合来获取所需的信息，但数据中心网络规模非常大，大部分未实现商用。

本论文基于数据中心提出了一种新的拥塞控制算法，结合了 AI 人工智能，设计了一个新的 LSTM+PIDNN 神经网络框架。解决了上述两个缺点，首先神经网络具有很强的学习能力，能够根据网络状态变化调整参数，解决了大部分算法的“针对性”；其次神经网络的实现以及相应数据（RTT）的获取不需要通过专门的硬件实现，而只需要相应的软件修改；因此更容易大规模部署；

1.4 论文结构安排

本文的章节安排如下：

第一章阐述了数据中心网络的背景及拥塞控制算法概况。对当前数据中心中拥塞控制算法进行了总结了分析，然后介绍了本论文的算法设计思想。

第二章讲述了数据中心网络的拓扑及通信模式的基本概念，以及不同拥塞控制算法的基本概念。包括对于 CLOS 拓扑和 INCAST 通信模式，以及 DCTCP、TIMELY、HCPP、ECMP 等。

第三章介绍了本论文设计的基于神经网络的数据中心拥塞控制算法，包括了模块划分，模型设计搭建及分析验证，以及数据集设计和网络训练方式。

第四章介绍了算法仿真效果。包括网络拓扑搭建及模型训练，效果分析。并对算法效果和其他拥塞控制算法进行比较，主要指标包括平均 RTT、发送速率等。

第五章对全文进行了总结，展望后续可能的工作。

2、数据中心拥塞控制基础

2.1 数据中心网络模型基础

2.1.1 数据中心网络拓扑

新一代数据中心网络架构大多数都为 CLOS 网络架构，CLOS 是一个对称的“接入-核心-接入”网络架构，具有以下特点

- 多级交换，典型为三级交换架构。
- 在每一级的每个单元都与下一级的设备全连接。
- 到指定目的地，在第 1 级交换单元存在多条路由，而后续交换单元都只存在唯一的一条路由。
- 严格意义上的无阻塞。
- 支持递归，可无限扩展。

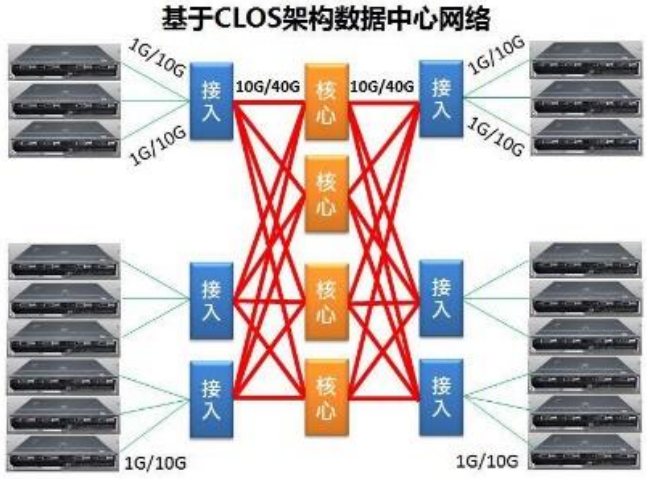

图 1、数据中心 CLOS 架构拓扑图

目前，基于 CLOS 架构搭建的数据中心网络拓扑有 Fat-Tree 和 Spine-Leaf

2.1.2 数据中心网络协议

高带宽、低延迟是目前数据中心应用的基本需求。RDMA 通过 Memory Region 机制使得网卡能够直接读写用户态的内存数据，避免了数据拷贝和上下文切换；并将网络协议栈从软件实现 offload 到网卡硬件实现，极大降低了 CPU 开销。随着 RoCEv2（RDMA over Converged Ethernet v2）技术的成熟，RDMA 可以部署在数据中心已有的网络设施上，RDMA 成为数据中心高速网络通信的主流方案。

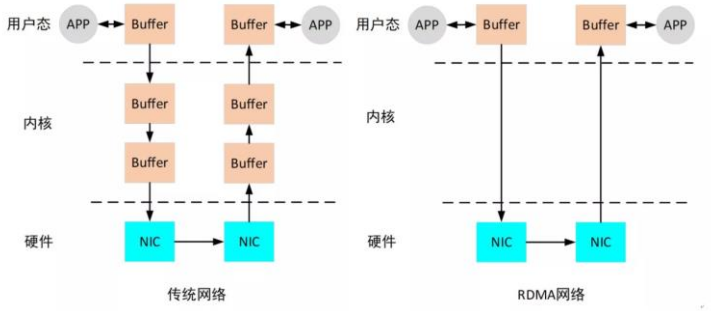

图 2、传统数据发送与 RDMA 协议流程对比

使用 RDMA 协议有以下优点：

- 零拷贝(Zero-copy)
- 内核旁路(Kernel bypass)

- 不需要 CPU 干预(No CPU involvement 消息基于事务(Message based transactions)
- 支持分散/聚合条目(Scatter/gather entries support)

2.1.3 数据中心网络对于流发送的需求

数据中心网络希望能够实现高吞吐、低时延发包[13]：

高吞吐：

数据中心中，有关储存应用的数据流数目相对较少，但是它却贡献了最多的字节数，一般成这样的流为长流或者大象流（Elephant flow），对于长流来说，他的吞吐时延需求不是很敏感，但需要维持较高的吞吐率，以满足应用的需求，为了实现高吞吐率，需要有效的、充分利用网络中的硬件转化能力，不能存在瓶颈环节。

低时延：

有关分布式计算或者 Web 服务请求之类的流量，它们的数目在数据中心中占比非常高，但是每条流的长度却很短，只包含一些通知信息，一般称这样的流为短流或老鼠流（Mouse flow），对于短流，通常希望快速返回结果，因此要求延迟最小化，这是数据排队往往成为瓶颈，不合理的传输策略往往是造成延时增加的重要原因。

数据中心网络 RDMA 协议的使用使得终端吞吐量大幅增加，但由于 RDMA 的重传机制，丢包对 RDMA 造成大幅性能下降，因此 RDMA 要求 L2 层或者 L3 层网络是无损的。

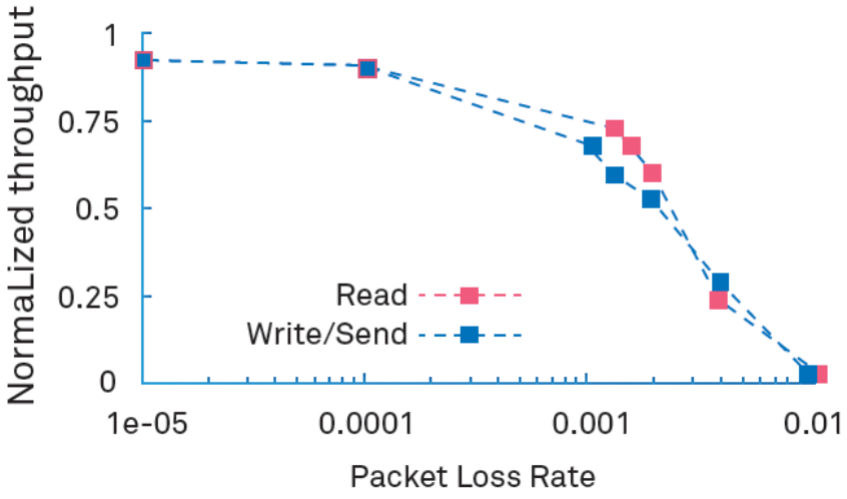

图 3、丢包率对 RDMA 吞吐量的影响

由上图可知，RDMA 吞吐量对于丢包率的增长是十分敏感的，当网络丢包率到 1%,时，RDMA 的读写速率下降到几乎为 1 零，只有当丢包率保持在一个非常低的比率时，才能够完整发挥 RDMA 的性能；

为了实现不丢包，数据中心网络 RoCEv2 引入了基于优先级的流量控制 PFC 协议，基于优先级的流量控制（PFC）功能是当接收设备输入缓存区的占用超过设 定阈值时，暂停上游发送设备，防止因缓存区溢出造成丢包。虽然提供了

RoCE 所必需的无损环境，但 PFC 的大规模使用也存在一些问题，包括 PFC 死锁的可能性。

因此，单是上述的 PFC 协议是无法避免网络拥塞的，数据中心网络还需在网络上的各个节点采取不同的拥塞控制协议或者机制，多者之间联合调控。

2.2 拥塞控制算法及机制

根据算法执行的位置进行分类，拥塞控制算法主要分为在终端节点控制的拥塞控制协议和在交换机等交换设备上实现的流量工程；两者虽然实现在 CDN 网络中不同的位置节点，但是都是为了合理调度网络资源以减少网络拥塞情况的发生。下面将具体进行介绍。

2.2.1 速率控制

在终端节点是拥塞控制主要就是根据获取的网络信息来控制 TCP 层发送速率，网络信息大多数从接收到的数据包中某个标志字段分析得出，大致分为三类：基于丢包率（TCP cubic）、基于 ECN 标志字段(DCTCP,HULL,DCQCN)和 RTT(TIMELY)。

2.2.2 流量工程

流量工程主要是实现在网络中交换机等交换设备上，主要分为流量调度和负载均衡两种；流调度主要是交换机队列数据包调度，通过某种机制或算法控制流的发送数量或顺序，有基于截止时间的 D2TCP、基于流大小的 pFabric 和基于链路价格的 FCP 等；而负载均衡则是控制数据流或数据包走不同的网络链路，避免大量数据报文占用同一条链路资源，使得不同物理路径之间负载平衡，例如 ECMP 算法；

除了速率控制和流量工程之外，有些拥塞控制算法结合了二者；例如阿里巴巴提出 uFAB 算法，需要终端和边缘交换机联合调控，边缘交换机发送探测数据包收集网络信息，后再将统计信息发送给连接的终端节点，终端节点根据网络状态控制流发送速率；同时交换机还可以根据网络状态控制流发送路径。

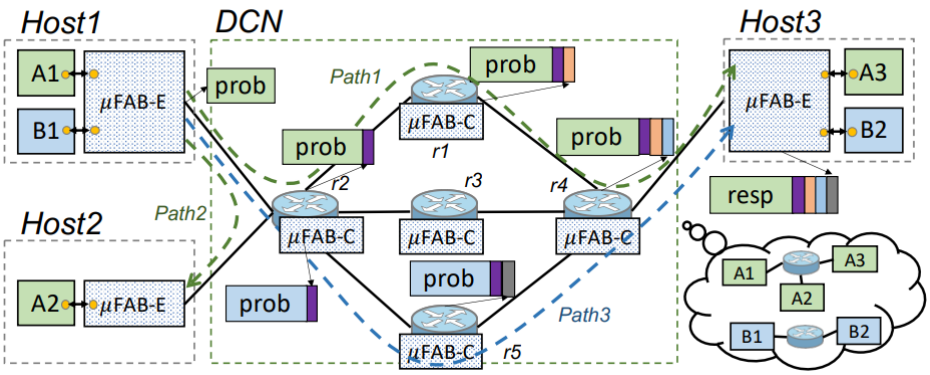

图 4、uFAB 拥塞控制算法网络框架

2.3INCAST 通信模式

Incast(又称 TCP Incast)是 many-to-one 的通信模式。这种情况发生在当一个父服务器向一组节点（服务器集群或储存集群）发起一个请求时，集群中的节点都会同时收到该请求，并且几乎同时做出相应，导致很多节点同时向一台机器（父服务器）发送 TCP 数据流，从而产生一个“微突发流”，Incast 经常会导致交换机中的缓冲区溢出，从而发生流量崩溃[13]。

2.4 拥塞控制算法总结

目前数据中心拥塞控制算法种类繁多；其中大部分对于特定拥塞场景或者对于特定东西向流（视频流、语音流、AI 数据流等）有着很好的控制效果，但是不能够作为处理拥塞的普适算法；其中一些算法需要对网络协议中报文字段进行增加或者对设备硬件进行修改，这些算法在拥塞处理上结果很理想，能够达到很好的预期，但是硬件的更改很难大规模部署，停留在测试阶段。很多拥塞算法参数固定，对网络状态变化适应能力差，速率控制比较机械，不够灵活，不能够适应智能化网络发展趋势。

3.基于神经网络的拥塞控制算法研究

3.1 算法设计思路及模块划分

基于神经网络的拥塞控制算法是主要使用神经网络来控制终端节点数据发送速率以达到控制拥塞的效果，根据发送终端收集到的网络拥塞信息，作为算法的输入，再经算法计算得出结果，输出作为速率调节因子。

使用神经网络实现拥塞控制算法主要为了利用神经网络的预测和学习功能。数据中心流量大，若不进行预测，当检测拥塞发生，再到调用控制算法进行调节时，很有可能已经错过了拥塞控制的最佳时机；即传统终端拥塞控制策略具有一定的滞后性，一是检测拥塞到拥塞信号传递到发送终端需要一定的时间，二是拥塞控制算法开始调节到作用到网络交换机拥塞点需要一定的传播时间；这两者是造成滞后性的两个方面，尤其是第一部分更为重要，往往越拥塞，拥塞信号传递给终端的时间就越长；因此，预知拥塞的发生，从而提前做出反应对于拥塞控制很有必要。整体算法设计思路为预测加控制。

传统拥塞控制算法都没有对网络的将来状态进行预测，一是考虑到数据中心网络状态复杂、包括有网络拓扑、发送速率、丢包率、ECN 标记等，二是预测需要比较复杂的数学模型进行学习，而传统算法绝大很多参数已经固定，无法准确预测，预测的准确度又对控制效果起着决定性作用。神经网络通过训练能够提高预测的准确度。对于一中复杂的网络状态，论文[12]中指出将队列长度当作网络拥塞级别，统计数据包 RTT 和拥塞级别之间的相关系数高达 0.9998，即基于延时反馈网络拥塞就有很高的准确性。综合考虑到终端节点获取的网络信息有限和算法复杂程度，将一维 RTT 信息作为拥塞反馈信息，具有以下好处：

1. RTT 延时反馈是隐式反馈，不需要任何内部网络支持。
2. 它可以采用比 ECN 或丢包率更大的值范围，提供更细粒度的反馈形式。

但由于数据中心网络中 RTT 短时间内变化剧烈，有着很高频率的抖动，所以不能够对原始 RTT 进行预测，数据预处理模块算法对原始数据进行一定的处理，使得由 RTT 数据变为能够训练和测试网络的数据集。

当实现预测之后，还需设计一个速率控制，终端对于拥塞的控制主要就是通过调节发送速率来实现的，这里使用神经网络搭建 PID 速率控制器，利用神经网络的学习功能去优化控制参数，尽可能实现更好的控制效果；相较于其他控制算法，使用 PID 算法对于 RTT 变化反应灵敏，且不需要很多参数，同时也能够控制发送速率使得真实 RTT 向目标收敛。

基于上述分析，神经网络拥塞控制算法从功能划分上主要分为四个模块：

- 网络信息采集模块
- 数据预处理模块
- RTT 时间序列预测模块
- PIDNN 速率控制模块

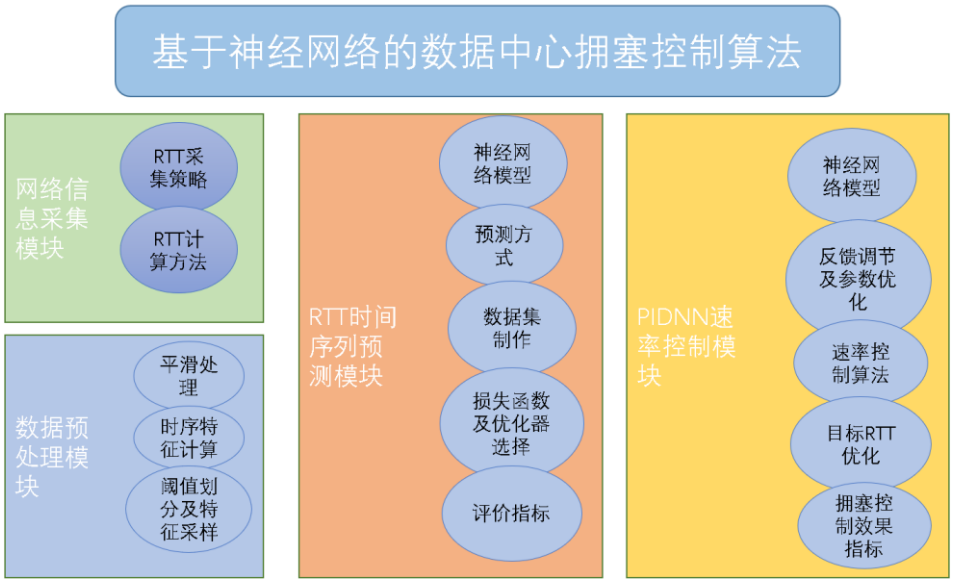

图 5、神经网络拥塞控制算法框架

每个模块实现各自功能，按照顺序相互配合，算法按照“网络信息采集模块->数据预处理模->RTT 时间序列预测模块->PIDNN 速率控制模块”的流程运行，后两个模块需要其提前将网络参数进行训练才能够使用。

3.2 网络信息收集模块设计

DCN 网络数据量庞大，发送连续数据包间隔时间极短，因此对每个数据包 RTT 都执行算法是不现实的，会占用大量的 CPU 计算资源，因此设计了 RTT 的采集策略，降低了算法执行的频率。

RTT 采集策略：每一个 RTT 完成时间内只统计第一个数据包，不同 RTT 完成时间不互相重叠。

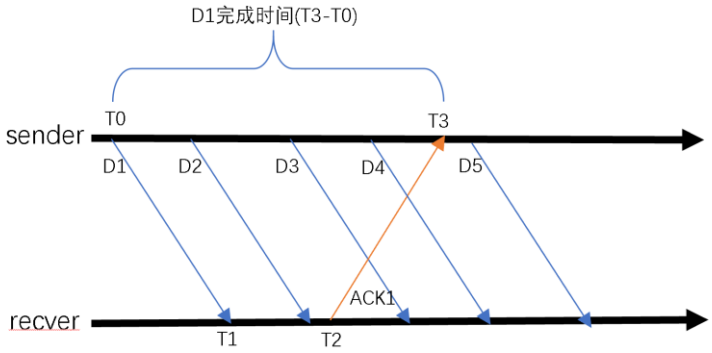

图 6、数据报文完成时间示意图

例如图 6 所示，发送方一共连续发送了 5 个数据包（D1 D2 D3 D4 D5）并收到了数据包 D1 的确认 ACK1，完成时间定义为数据包从发送到收到 ACK 确认之间的时间，即 T0 到 T3 的时间，而这个时间段内发送方还发送了 D2 D3 D4 三个数据包，根据统计策略，在这个时间段内，只会统计 D1 的往返延时 RTT1，而不会统计 D2 D3 D4 这三个数据包的往返延时，T3 之后则会统计 D5 的往返延

时，同时不会统计从收到 D5 确认报文 ACK5 时间段内发送的其他报文 RTT.采取这样的策略主要考虑以下几点优点：

- 由于 DCN 网络发送速率大，报文数量多，速率控制次数和频率是由统计的 RTT 次数和频率决定的，每统计一次 RTT，就需要根据算法进行一次速率调节。因此如果每一个报文都去进行统计，则速率越大，算法运行频率越快，这对于终端计算负载很大。因此采取此策略降低算法运行频率，以缓解计算压力。
- 降低了对于算法计算时间限制，假设运行一次拥塞控制算法所需的时间为 T, 如果采取逐包预测，则该时间 T 则需要小于连续接收到的两个 ACK 时间间隔Δt，否则上一数据包算法还没有计算完，下一个数据包的 RTT 已经能够计算出来，预测模块便失去了作用。如果采取每一个 RTT 完成时间内只统计第一个数据包，只需要满足时间 T 小于第一个数据包的完成时间$T3-T1$；实际情况中$T3-T1$远大于Δt；这放宽了对与算法时间上的限制，因此可以采用更为复杂的算法去实现控制。
- 降低了算法运行频率同时也降低了发送速率变化频率，这使得发送窗口大小更加稳定。，网络稳定时，前后连续几个报文的 RTT 几乎相等，该策略降低了相同数据的重复运算的次数。
- 避免了不同 RTT 之间统计数据的交叉储存。

但该采集策略也有以下缺点：

- 采集频率不固定，受到数据包完成时间的影响，完成时间越大，采集频率越低，完成时间越小，采集频率越高；而完成时间受网络状态影响最大，因此需要尽可能保证网络的稳定性，使得 RTT 在一个较低的范围波动。
- 当网络拥塞时，RTT 时延迅速增加，完成时间也会大幅增加，这样会导致采集频率降低，速率控制频率降低；该情况可以通过拥塞预测解决，因此需要很低的预测偏差（MAPE 在 10%以下）。

RTT 计算方法：

根据图 6 中统计数据可知，

$$RTT = (T3 - T0) - (T2 - T1) \tag{1}$$

根据数据数据包发送时间统计 T0，在仿真平台中，由于发送的并非真实数据包，因此接收端在收到发送方数据包时不需处理直接返回相应的 ACK 报文，因此$(T2-T1) \approx 0$，发送方根据报文序列号进行判断，当收到了第一个数据包的 ACK 回应时，统计出 T3， $RTT \approx (T3-T0)$得出采集 RTT 值；采集完一个值后，下一个发送的数据包则作为下一个被采集的报文，更新 T0，以此流程反复，直到整个流发送完毕。

3.3 数据预处理模块设计

3.3.1 算法设计

数据预处理对于后续的预测准确度影响很大，选择一个合适的处理算法，能够避免网络训练发生过拟合和欠拟合的情况。

数据预处理模块包括了单节点数据包时延平滑处理、时序特征计算和采样；

平滑处理：

数据中心 RTT 抖动频率很大，一般是以微秒（us）为单位，不适合神经网络训练，往往会造成欠拟合的情况，因此显示对原始数据进行平滑处理，处理公式如下：

$$S_t = \begin{cases} R_t, t = 0 \\ \sigma * R_t + (1-\sigma) * S_{t-1}, t \geq 1 \end{cases} \quad (2)$$

R_t是信息采集模块采集的第t 个 RTT, S_t是处理后的平滑数据，σ是平滑因子，一般取 0.2；

这里的 t 并非真正意义上的时间，而是时间序列的顺序，即$t \in N$;

根据从仿真平台采集到的 RTT 数据进行平滑处理，效果如下：

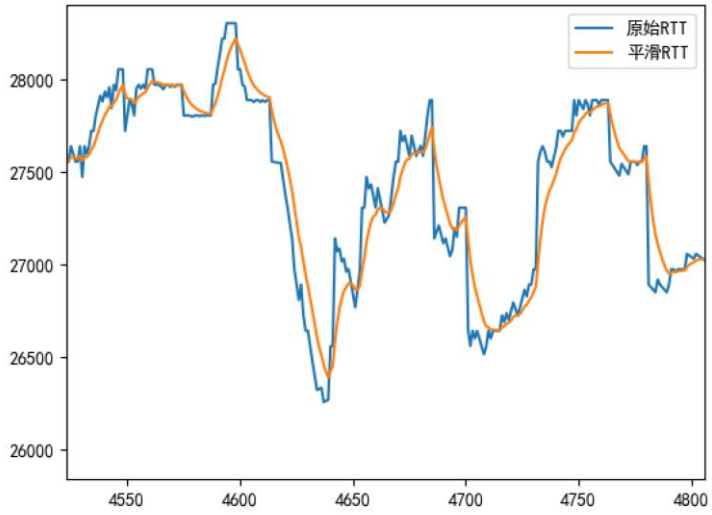

图 7,RTT 平滑处理对比

时序特征计算：

由于平滑因子 σ 较低，平滑效果很好，但是处理前后 RTT 损失了很多时序特征，如果不考虑损失的时序特征，会导致神经网络预测偏差较大。因此根据平滑前后值，设计了一种算法，提取其特征值，计算方法如下：

$$\begin{cases} K_t = (R_t - S_t)/S_t \\ L_t = (R_{t+1} - S_t)/S_t \end{cases} \quad (3)$$

K_t即为采集模块采集的第t个 RTT 平滑处理损失的幅度特征, K_t保留了平滑处理前后损失的时序抖动，因此通过平滑值及其特征就能够得出真实值；而L_t为采集模块采集的第$t+1$个 RTT 与对应 t 时刻平滑值的变化幅度特征, L_t是对未来变化的一种预测特征，即在真实过程中L_t的大小是未知的，通过对L_t的预测，再根据当前时刻的平滑值，得到对未来 RTT 真实值的一个预测。在实际 RTT 时间序列预测过程中，就是通过连续多个已知 K_t对L_t进行预测，从而实现对下一时刻 RTT 的预测。

阈值划分及特征采样：

并不是越多的数据用于训练预测效果越好，观察多个拥塞控制算法采集后的变化幅度特征K_t发现，大部分的变化幅度特征值的大小都聚集到一个很小范围，造成这种情况的原因是不同的拥塞控制算法对于 RTT 控制各有特点（例如 HCPP

算法目的是为了尽量让交换机队列保持空状态，因此 RTT 大多数情况很低，变化幅度特征值大多数情况趋近 0，而 TIMELY 算法根据前后的 RTT 变化梯度进行速率控制，这样就会导致 RTT 有很强的抖动，因此变化幅度特征值的绝对值大多数时间保持在一个很高的值），但无论哪个拥塞控制算法都会使变化幅度特征值的大小分布不均匀，用这些数据直接训练往往会导致过拟合的结果（即训练集预测指标很高，而测试集指标远远低于训练集）。观察多个算法(HPCC,TIMELY,DCTCP 以及未优化参数的 PIDNN)运行过程中特征值K_t的大小并统计，CDF（累计分布函数 cumulative distribution function）统计结果如下：

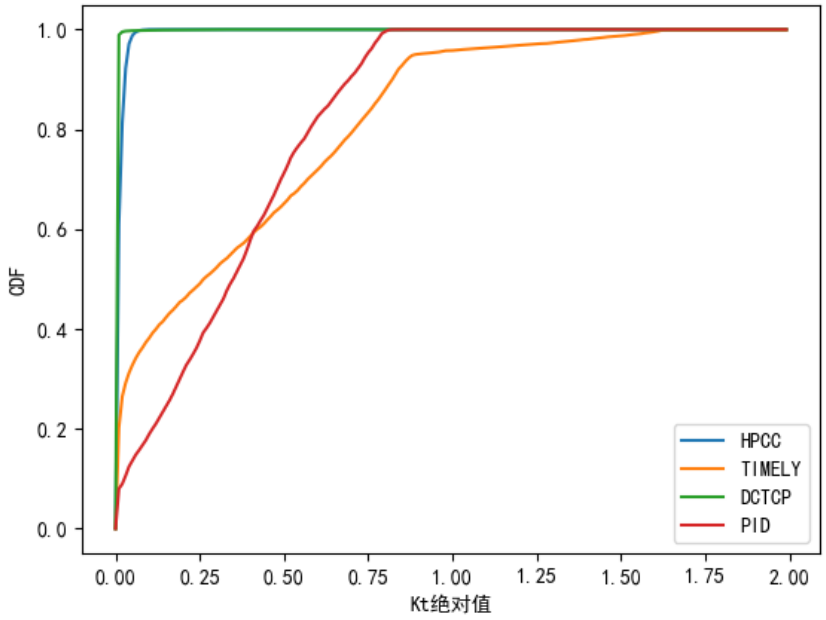

图 8、多个算法特征值K_t累积分布函数图

CDF 值上升越快说明拥塞控制过程中 RTT 变化较稳定，抖动百分比小。上图中 HPCC 算法和 DCTCP 算法 CDF 值上升较快，但是 HPCC 算法因为控制准确 RTT 稳定值较低，DCTCP 整体 RTT 非常高，因此同等变化下抖动百分比小。TIMELY 算法和 PID 算法都是基于 RTT 的单一控制算法，PID 虽然控制比 TIMELY 精确度高，且 PID 参数未经过训练，因此两者 CDF 控制效果比较相似，抖动大，CDF 上升慢，比较平稳。

从数据集制作角度上分析，CDF 上升越平稳可制作数据集规模越大：

观察统计结果可知;HPCC 算法 90%的特征绝对值和 DCTCP 算法 87%都处在一个较低的范围（0.，0.04），而 TIMELY 算法特征绝对值主要分布在（0.08，1）范围内，但在（1，2）范围内仍有分布；未优化参数的 PIDNN 分布相对均匀。如果将所有的数据都作为数据集训练网络，则大量的低绝对值K_t会造成网络的过拟合；为了避免该情况的发生，根据特征值绝对值的大小划分为四个范围，为[0,0.02）、[0.02，0.08）、[0.08，0.15）、[0.15，inf）,并根据从每个范围中采样出近似数量的数据作为最终用于训练和测试神经网络的数据集，最终选取 TIMELY 算法和 PID 算法采集制作数据集。

3.4RTT 预测模块设计

3.4.1 神经网络模型设计

神经网络模型选择：

预测模块受到以下两方面的约束：

- 数据中心数据包完成时间较短，因此预测模块需要很快的前向传播时间，在下一个 RTT 采集之前预测出结果，网络模型不能够过于复杂或庞大。
- 速率控制要求预测模块预测出 RTT 变化趋势，因此需要较低的预测偏差，这对拥塞控制的有着决定性影响，因此当网络模型过于简单造成准确度不足时，预测反而对拥塞控制有着负作用。

通过比较 LSTM、GRU、Simple RNN 三种神经网络：

- Simple RNN 模型最为简单，预测效果最差，多次训练后 LOSS 值很大
- GRU 模型较为简单，同等规模预测效果较好，但效果不如 LSTM，没有达到预期
- LSTM 模型复杂，预测效果最好

最终实现的预测神经网络总共有两层隐藏层

第一层为三 LSTM 神经元（根据 LSTM 传播特性可知，LSTM 前向传播时间与该层神经元数量相关，考虑到 LSTM 内部复杂性、约束限制和前后特征相关性，只使用三个神经元），隐藏层维度是 16；输入为三个连续的K_t。

第二层为 linear 层，负责将多维转化为一维时序特征L_t，并作为预测输出。

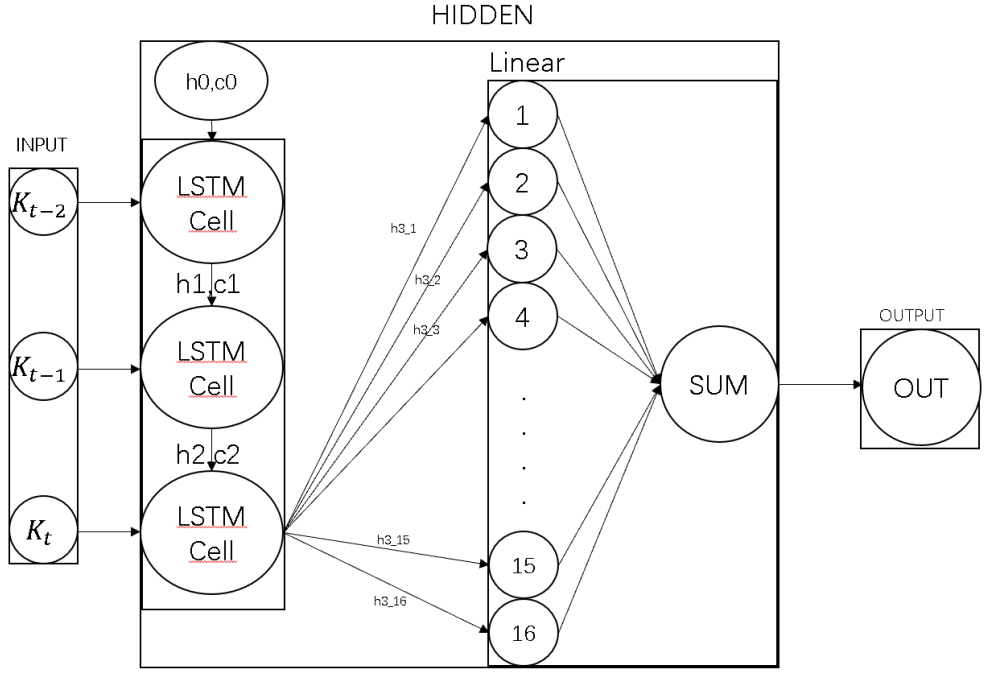

图 9、时间序列预测网络结构

每个 LSTM 神经元都有遗忘门、输入门和输出们；神经元的输入包括时间序列x_t,上一个神经元输出结果h_{t-1}和上一个神经元状态C_{t-1}；

遗忘门该门会读取h_{t-1}和x_t，输出一个在 0 到 1 之间的数值给每个在细胞状态C_{t-1}中的数字。1 表示“完全保留”，0 表示“完全舍弃”;，计算公式如下,其中f_t为遗忘门输出，：

$$f_t = \sigma\left(W_f * [h_{t-1}, x_t] + b_f\right) \tag{4}$$

输入门负责下一步决定让多少新的信息加入到 cell 状态中来。实现这个需要包括两个步骤：首先，一个叫做“input gate layer ”的 sigmoid 层决定哪些信息需要更新；一个 tanh 层生成一个向量，也就是备选的用来更新的内容，C_t 。再下一步，把这两部分联合起来，对 cell 的状态进行一个更新，现在是更新旧细胞状态的时间了，C_{t-1}更新为C_t。前面的步骤已经决定了将会做什么，现在就是实际去完成。把旧状态与f_t相乘，丢弃确定需要丢弃的信息。接着加上$i_t * C_t$。这就是新的候选值，根据决定更新每个状态的程度进行变化，计算公式如下：

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{5}$$

$$C_t = \tanh(W_C * [h_{t-1}, x_t] + b_c) \tag{6}$$

输出门需要确定输出什么值。首先，运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终仅仅会输出确定输出的那部分,计算公式如下：

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = o_t * \tanh(C_t) \tag{8}$$

参数说明：

LSTM 层是由多个 LSTM 神经元串连起来的，共用一套权值参数和偏置参数，参数传递流程如图 9 所示，进行运算的神经元记为新神经元，其上一个记为旧神经元。

C_{t-1}和C_t分别是旧神经元和新神经元的细胞元状态，h_{t-1}和h_t是旧神经元和新神经元的输出。x_t是新神经元的输入。f_t、i_t、C_t、o_t、h_t是各个门的输出结果。

上述公式中，W_f、W_i、W_c、W_o是神经元中的权值参数，b_f、b_i、b_c、b_o是神经元中的偏置参数，预测神经网络的训练目的就是对这两种参数的优化更新。

σ和$tanh$是两个激活函数，前者保证输出值在 0 到 1 之间，后者保证输出值在-1 到 1 之间，两者都是用来加入非线性因素的，提高神经网络对模型的表达能力，具体计算公式如下：

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{10}$$

LSTM 神经元结构图如下：

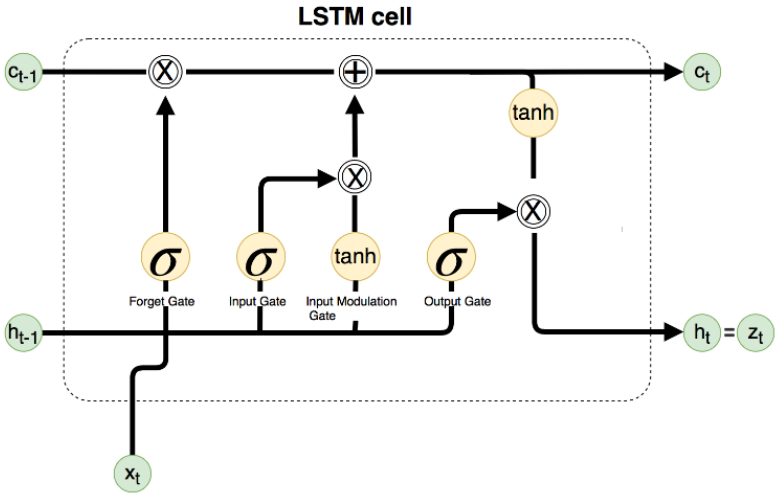

图 10、LSTMCell 内部结构

3.4.2 数据集、损失函数、学习率等参数设计

结合数据预处理和预测神经网络，设计的参数如下：

数据集：

从数据预处理模块和 RTT 时间序列预测模块，用到的网络信息只有采集到的数据包往返时延，使用不同拥塞控制算法，数据发送速率不同，所造成的数据包 RTT 也不同，但是从神经网络的角度分析，它对与 RTT 的预测不考虑用了什么拥塞控制算法，只关心网络的模型、网络的输入输出和最终的预测效果，RTT 数据的来源算法是不重要的，只需保证此 RTT 是真实仿真环境下的统计输出。采集出的 RTT 数据来自于了论文后续优化 PIDNN 参数时的仿真数据和使用 TIMELY 算法的仿真数据，总共有 19058 对数据。

Input:输入为三个连续的特征值[K_{t-2}、K_{t-1}、K_t]

Label:标签设置为当前时刻的预测特征L_t

Train:每轮从数据集中随机选择 800 对数据进行训练

Test: 每轮从数据集中随机选择 200 对数据进行测试

Epoch:由于数量限制，设置训练 19 轮

损失函数：

$$loss(x,y) = \frac{1}{n}\sum_{i=1}^{n}|yi - f(xi)| \tag{11}$$

学习率：0.001

根据预测神经网络的 t 时刻输出 out 和公式 3 推导，t 时刻 RTT 的预测结果如下:

$$RTTpred(t) = (1 + out(i)) * S_t \tag{12}$$

3.4.3 算法分析验证

在 19 轮数据集训练过程中，训练集损失 loss 变化如下：

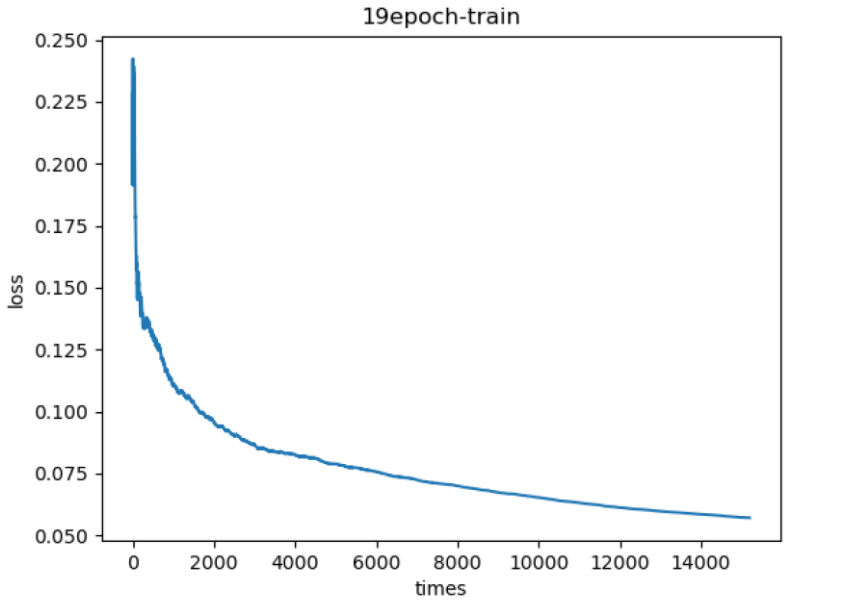

图 11、训练集 loss 变化

上图为数据集训练过程中整体 loss 变化图（做了一个时间平均处理），横坐标为训练次数，纵坐标为训练 loss,观察上图可知，随着训练次数的增加，训练 loss 逐渐减小，且下降速度由快到慢，后趋于平稳，说明训练模型参数逐渐向最优值收敛。

使用 MAPE(Mean Absolute Percentage Error,平均绝对百分比误差)评价指标评估预测效果,n 为预测步数：

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|RTTpred(i) - RTTtrue(i)|}{RTTtrue(i)} = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|out(i) - L_i|}{(1 + L_i)} \quad (13)$$

MAPE 值越小，说明预测结果和真实结果越接近，预测越准确，每一轮预测统计 MAPE 值如下：

表 1、MAPE 统计表

MAPE/epoch	1	2	3	4	5	6	7	8	9
训练集	0.113	0.089	0.083	0.064	0.061	0.050	0.052	0.055	0.051
测试集	0.081	0.079	0.059	0.083	0.058	0.067	0.053	0.045	0.044

MAPE/epoch	10	11	12	13	14	15	16	17	18	19
训练集	0.046	0.044	0.042	0.039	0.040	0.042	0.040	0.040	0.036	0.035
测试集	0.042	0.045	0.039	0.044	0.037	0.043	0.035	0.038	0.042	0.036

观察上表可以看出，随着训练轮式的增加，整体 MAPE 指标在逐渐降低，说明预测神经网络预测效果越来越好，同时可以看到初始几轮的 MAPE 值“较低”，但真实效果却并非“较好”，原因如下：

预测神经网络并非对未来时刻 RTT 的直接预测，而是对特征值的预测再通过公式 12 计算出对 RTT 的预测值，根据公式 13 可以看出，其 MAPE 指标在计算过程中经过了一定的转化，分母变$1 + L_i$：

1：当L_i较小时，MAPE 值接近两者之差。

2：当L_i较大时，MAPE 值接近两者百分比差距。

因此初始训练时，预测值和真实值百分比差距较大，因为是从K_t 在[0,0.02）、[0.02，0.08）、[0.08，0.15）、[0.15，inf）四个范围采样，L_k与K_t接近，因此除了最后一个范围，绝对差距较小，这对于 MAPE 值具有一定程度的降低。从数值上第 1 轮的 MAPE 是第 19 轮的 MAPE 指标的 3 倍左右，但实际预测效果差距非常大。

3.4.4 滑动预测机制

无论是在仿真平台还是真实网络环境中，采集 RTT 总是符合时间上的先后顺序，即预测是根据 RTT 时间序列的预测，这就引发一个思考：

当前根据相应的数据预测出下一时刻 RTT 后，什么时刻再次进行预测。即前后两次预测的时间间隔。是固定时间还是根据采集 RTT.

参考其他拥塞控制算法，每采集一次数据就运行一次拥塞控制算法。因此为了后续算法比较的公平性，设计相同的算法执行机制，即预测根据采集 RTT 的序列，每采集到一个 RTT 便执行一次算法（每次执行算法需要一次预测）。

每次预测使用本时刻及前两个时刻的特征值，从 RTT 时间序列上看，需要本时刻 RTT、前两个时刻 RTT、以及前三个时刻的 RTT 平滑值。若需要实现上文的算法执行机制，则需进行滑动预测。本时刻数据预处理输入：$[S_{t-3}, RTT_{t-2}, RTT_{t-1}, RTT_t]$，下一时刻输入：$[S_{t-2}, RTT_{t-1}, RTT_t, RTT_{t+1}]$，每次向前滑动一个窗口。

3.5PID 速率控制模块设计

在一些自动化设备控制中，由于被控变量和控制变量之间没有准确的数学映射关系，因此常常采用 PID 控制算法，根据经验修改算法参数，使得该控制算法通常能够发挥很好的控制效果；而且算法结构简单、稳定性好、工作可靠、调整方便。

本论文将 PID 算法中各个控制模块作为神经网络的神经元，将该算法实现为一种神经网络，通过反向传播学习的方法替代人工经验调整参数，使其能够适应网络状态变化，在预测模块的配合下，能够对未来拥塞情况或趋势进行提前的预知，提前调节速率进行防范，从而达到良好的拥塞控制效果。

3.5.1 网络模型搭建

速率控制模块使用 PID 算法进行速率调节，预测模块的 RTT 预测值作为该模块的输入，输出为速率调节因子。

PID 控制算法总共包括了三个小模块，差分模块、积分模块和为分模块；

差分模块：最重要的调节因子，根据实际输入和目标值之差的特定比例作为模块输出，是该时刻的目标调节因子。

积分模块：对实际输入和目标值之差进行一定时间的累计求和，由于实际仿真过程中算法运算次数巨大，为了避免积分模块短时间内的爆炸式增长，因此修改为长时间平均,作为长时间调节因子

差分模块：实际输入和目标值之差进行时间求导，作为梯度调节因子。

PID 输入为预测和目标的差值：

$$e(t) = (rtt_{pred}(t+1) - rtt_{target})/rtt_{target} \tag{14}$$

PID 输出为三个调节因子和系数的乘积和：

$$PID\big(e(t)\big) = \partial(t) = K_p(t) * out_p\big(e(t)\big) + K_i(t) * out_i\big(e(t)\big) + K_d(t) * out_d\big(e(t)\big) \tag{15}$$

PID 调节因此计算如下：

$$out_p\big(e(t)\big) = e(t) \tag{16}$$

$$out_i\big(e(t)\big) = \frac{1}{T}\int e(t)dt \tag{17}$$

$$out_d\big(e(t)\big) = de(t) = e(t) - e(t-1) \tag{18}$$

根据 PID 输出，控制发送速率：

$$rate(t) = rate(t-1) * \big(1 + \partial(t)\big) \tag{19}$$

为了防止速率的剧烈变化，设置 PID 控制阈值：

$$\partial(t) = \begin{cases} 0.5, \partial(t) > 0.5 \\ \partial(t), -0.6 \le \partial(t) \le 0.5 \\ -0.6, \partial(t) < -0.6 \end{cases} \tag{20}$$

为了后续的参数优化，将 PID 控制算法搭建为神经网络。

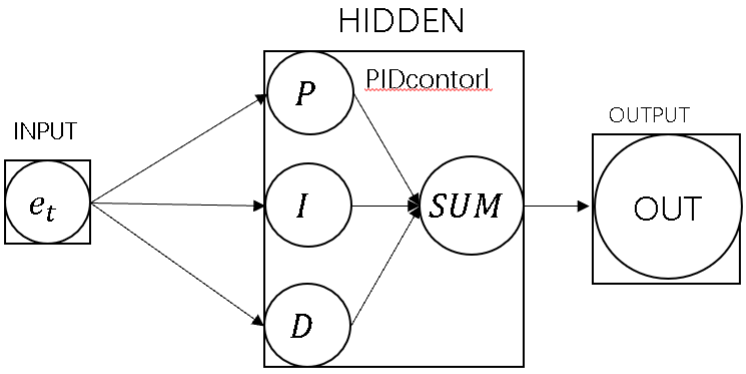

图 12、PIDNN 结构

PID 算法控制简单，但是对于其参数 KP、KI、KD、RTTtarget 决定了算法输出，从而影响拥塞控制的好坏，传统 PID 算法根据经验调节 KP、KI、KD 参数大小，为适应网络变化，将 PID 搭建为神经网络，对 KP、KI、KD 参数进行自动调节。RTTtarget 为我们控制的拥塞目标，在仿真环境中一般为 5 微秒常数。

3.5.2 参数优化设计

参数优化阶段 KP、KI、KD 是随时间变化的变量，通常记为 W(t) = [KP(t)、KI(t)、KD(t)]，之后的使用阶段是优化后的固定常数。

根据误差公式：

$$J(t) = \frac{1}{2}(rtt(t+1) - rtt_{target})^2 \tag{21}$$

通过反向求导对参数 W(t)进行优化，由公式[15,19,21]可知，根据误差对三个控制参数求导如下：

$$\frac{dJ(t)}{dw(t)} = \frac{dJ(t)}{drtt(t+1)} * \frac{drtt(t+1)}{drate(t)} * \frac{drate(t)}{dw(t)} \tag{22}$$

其中，根据公式 21 可得到：

$$\frac{dJ}{drtt(t+1)} = rtt(t+1) - rtt_{target} \tag{23}$$

$$\frac{drate(t)}{dw(t)} = \left[rate(t-1) * out_p(t), rate(t-1) * out_i(t), rate(t-1) * out_d(t)\right] \tag{24}$$

但是$\frac{drtt(t+1)}{drate(t)}$无法精确求出，由$\frac{rtt(t+1)-rtt(t)}{rate(t)-rate(t-1)} = \frac{rtt(t+1)-rtt(t)}{rate(t-1)*\partial(t)}$代替，而仿真环境中，由于 rtt 和 rate 抖动较大，因此为了参数的快速收敛，将该值替换为一个接近 1 的常数β

参数更新：

$$w(t+1) = w(t) - \mu(t) * \frac{dJ(t)}{dw(t)} \tag{25}$$

其中$\mu(t)$为参数更新的学习率，可以设置为一个很小的常数；当参数收敛困难时，也可以设置为一个随时间减小的变量;本论文中为了加快参数的收敛，学习率设计如下：

$$\mu(t) = 0.01 * \frac{1}{\sqrt{t+1}} \tag{26}$$

当训练时间较短时，参数更新快，而随着时间增加，参数逐渐趋于稳定，不再更新。

观察网络环境可知，当 t 较小时，网络数据包发送处于起步状态，网络状态波动较大，此时 RTT 震动幅度加大，即参数$\frac{dJ(t)}{dw(t)}$震动幅度较大，而且此时刻学习率较大，这种情况容易导致参数发生突变，这对于 PID 参数的后续的学习更新会造成严重的错误，可能导致后续参数不断负优化，为了避免这种情况发生，设置求导梯度范围为在[-0.1，0.1]之间.

3.5.3 参数训练

PID 速率控制模块训练主要基于在线的方式，因为每次反向传播中的误差计算需要下一时刻的 RTT，而下一时刻的 RTT 又受到速率控制模块的影响，因此不能够通过其他的拥塞控制算法（TIMELY）获取的下一时刻 RTT 数据进行训练，只能使用 PID 算法；训练时使用在线的方式，在仿真平台中部署还未训练的 PIDNN，然后使用该速率控制模块调节发送速率，并根据该算法调节下的下一时刻 RTT 通过反向传播参数梯度求导的方式，使用公式 25 对参数进行优化，多轮的训练后 PIDNN 参数收敛。

PID 参数的初始化对于参数后续优化影响很大，根据经验判断参数的大致范围，以$rtt_{target} = 5us$为列,设置初始值$[k_p(0)\ k_i(0)\ k_d(0)] = [-0.2\ -0.05\ 0.1]$,优化过程中参数变化如下：

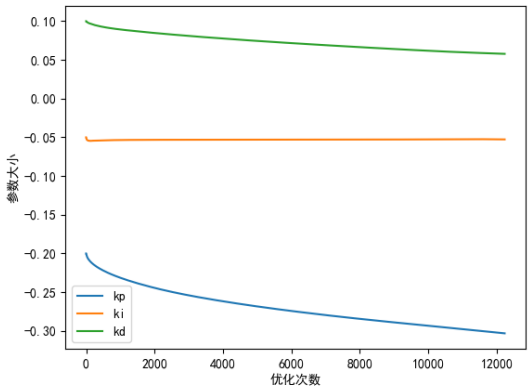

图 13，PID 算法参数优化曲线

根据上图可知，随着学习次数的增加，三个参数逐渐趋于稳定，最终优化结果为$[k_p\ k_i\ k_d] = [-0.358 - 0.060\ \ 0.040]$；

3.5.4 目标 RTT 调节机制

数据中心网络状态变化快，设置固定目标 RTT 值无法适应状态变化，为此添加一个目标 RTT 调节机制。当某个流发送数据包往返时延无法达到目标 RTT 设定值时，该机制会根据之前采集的 RTT 平均值对 RTTtarget 大小进行调整。该机制主要包括三个功能：

目标 RTT 调节判断器

判断器根据一个流连续多个采集数据包 RTT 是否能够达到目标要求；调用一个计数器对连续多个采集 RTT 都大于或小于目标 RTT 进行计数，若发现 RTT 在目标 RTT 上下震荡时，计数器清零；若连续 RTT 在目标 RTT 一侧时，计数器不断累计；当累计到一定数量时触发 RTT 目标调节器。逻辑如下：

```
Init count = 0//初始化计数器为 0
Set RTTch_num 6//设置触发数量
```

```
UPDATA 目标偏差值
b = (rtt - rtt_target)>0?1:-1//观察采集 RTT 在目标上侧 1 还是下侧-1
if count*b>=0 //说明采集 RTT 保持在目标一侧
   count++;
else count=0;
if count>RTTch_num//触发目标调节器
   RTT 目标调节
   RESET count
```

表 2、RTT 目标调节逻辑

采集 RTT 目标偏差统计器

RTT 平均统计器统计该流所有采集 RTT 与目标 RTT 偏差的均值。计算如下：

$$\Delta rtt = \frac{1}{T}\sum_{i=1}^{T}\left(RTT_i - RTT_{target}\right) \tag{27}$$

RTT 目标调节器

当触发该调节器时，会调用 RTT 平均统计器结果对 RTT 目标值进行修改,修改之后会重置计数器。具体修改如下所示：

$$rtt_{target} = rtt_{target} + \Delta rtt \tag{28}$$

在文章后续的仿真验证中，对于 INCAST 通信模式下，没有其他流加入，网络状态稳定，因此不加入该调节机制；当加入 30%网络负载随机流时，引入该机制以适应网络状态变化。

3.6 本章小结

本章将设计的基于神经的数据中心网络拥塞控制算法进行模块划分，介绍各模块的意义、功能、算法实现；其中包括了基于神经网络实现的 RTT 数据包往返时延预测模块以及 PIDNN 速率控制模块，使用这两个模块需要提前对网络参数进行训练优化，并根据网络特征采取不同的优化方法；最终完成各个模块的搭建。

最终实现的算法流程如下：

表 3、基于神经网络的数据中心拥塞控制算法

输入：RTT_t

输出：$RATE_t$

确定控制目标：rtt_{target}

更新：$[S_{t-4}, RTT_{t-3}, RTT_{t-2}, RTT_{t-1}] \rightarrow [S_{t-3}, RTT_{t-2}, RTT_{t-1}, RTT_t]$

根据公式 2-3 计算出：

$$[S_{t-2}, S_{t-1}, S_t, k_{t-2}, k_{t-1}, k_t]$$

k_{t-2}, k_{t-1}, k_t三个特征值输入预测神经网络得到输出：out_t

计算 RTT 预测值：

$$RTTpred(t) = (1 + out(i)) * S_t$$

PIDNN 计算：

$$\partial(t) = PIDNN(RTTpred(t) - rtt_{target})$$

IF $\partial(t)$>0.6

$\partial(t) = 0.6$

ELSE IF $\partial(t) < -0.6$

$\partial(t) = -0.6$

$$RATE_t = RATE_{t-1} * (1 + \partial(t))$$

更新：$RATE_{t-1}$->$RATE_t$

下一步需要将各模块先后部署到 NS3 仿真平台进行功能等相关测试，验证通过后合并所有模块实现整体算法部署到 NS3 仿真平台进行算法验证。

4、神经网络模型仿真及算法比较

4.1 NS3 仿真平台环境搭建及设计

4. 1. 1 环境配置

NS3 仿真平台基于 LINUX 系统中的 C++实现，系统复杂，要求特定的环境变量，在实现算法仿真前需要将相应的编译环境及环境变量完成配置（仿真平台环境配置不包括神经网络训练所使用到的环境）。

具体配置如下：

虚拟机平台：VMware Workstation 16 pro

镜像：ubuntu-22.04-desktop-amd64.iso

C++版本：gcc version 4.8.5

Python 版本：Python 2.7.18

仿真平台：基于 NS3 3.17 的 HPCC: High Precision Congestion Control (SIGCOMM' 2019)

仿真代码编写软件：Visual Studio Code 1.74.3

4.1.2INCAST 通信网络拓扑仿真设计

为了尽可能地模拟数据中心拥塞场景，在设计仿真拓扑时选择了最容易发生拥塞的 INCAST 通信场景，其中设计 20 个终端节点向一个目的发送大量数据包，每个终端节点连续发送同一条流，并使用一个交换机将该 20 条流进行合并，使用一条链路传递给终端，该通信模式十分考验发送端对发送速率的控制（因为交换机出口对所有流合并处理，发送速率的控制准确度决定了合并处端口的拥塞程度，消除了交换机上相关的负载均衡协议或者机制对模拟的影响），具体网络拓扑和流大小如下图所示：

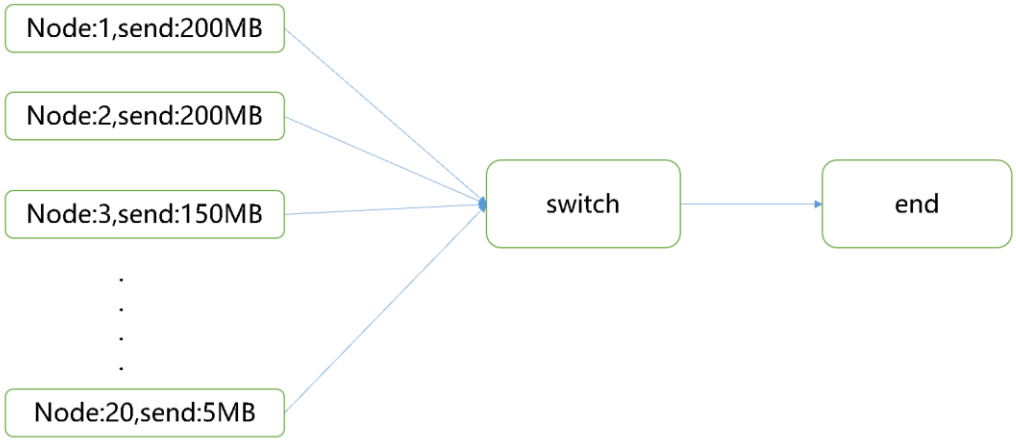

图 14、仿真 INCAST 网络拓扑

总共 20 个节点，每个节点负责发送一条大数据流，其中前三个节点发送流的大小分别是 200MB、200MB、150MB,其余 17 条流的大小都为 5MB，网络发送速率限制为 1Gbps-100Gbps;初始发送速率为 10Gbps.

4.2 算法模块化效果展示

将整体算法分为三个部分分别展示效果，即数据采集、RTT 预处理预测和 PID 速率控制，一方面为了验证各模块在仿真平台上是否满足设计预期功能，另一方面同时观察各自效果，便于模块修正，最后再将三个部分同时部署到仿真平台中。

4.2.1RTT 数据获取及展示

RTT 时间序列预测模块需要根据设计的采集策略获取原始 RTT 值，基于 NS3 平台搭建好仿真模型，先后部署 TIMELY/ DCTCP 拥塞控制算法，并搭建设计的网络拓扑，分别对 20 个节点进行 RTT 采集，由于流的数量较多，这里仅展示节点 1 采集数据。

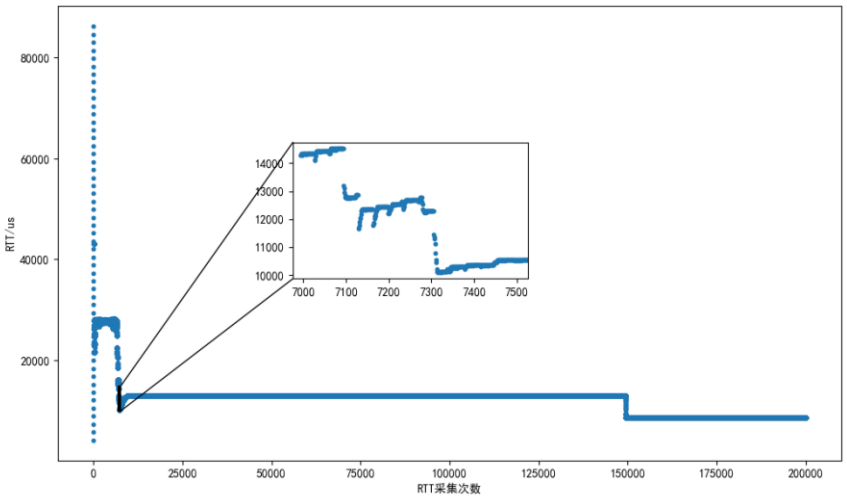

图 15,DCTCP 算法 RTT 采集数据

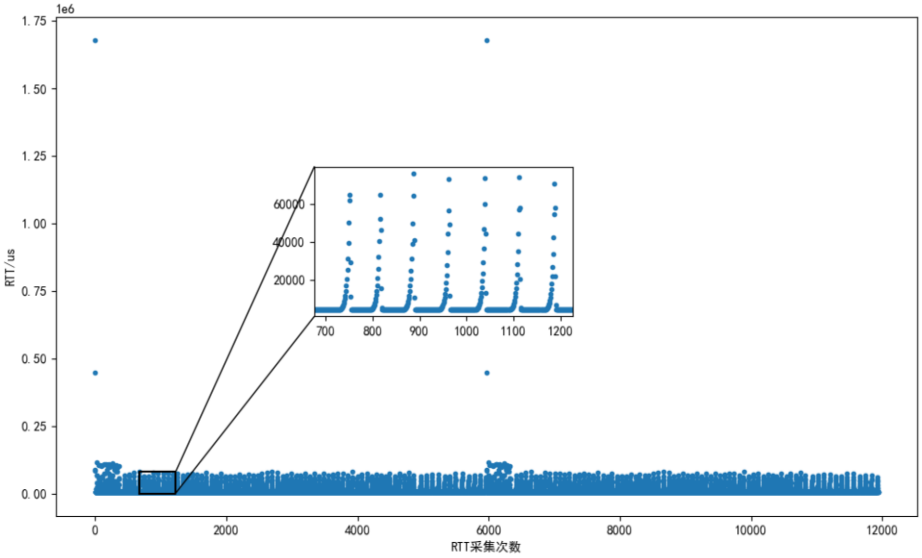

图 16、TIMELY 算法 RTT 采集数据

4. 2. 2RTT 预测模块效果展示

通过 3.4.2 的数据集划分及训练方法，对上述数据集进行训练，根据 19 轮训练后的网络模型，使用 PYTORCH 模型中的 torch.save 函数将训练后网络保存为 pt 文件。基于公式[4-10]在仿真平台使用 C++复现该神经网络算法，并使用 torch.load 读取训练后的参数人工加入到 C++代码中，运行仿真平台测试效果，这里选取其中两个节点 RTT 预测结果，由于数据量过大，这里仅展示部分时刻的 RTT 预测结果，如下图所示：

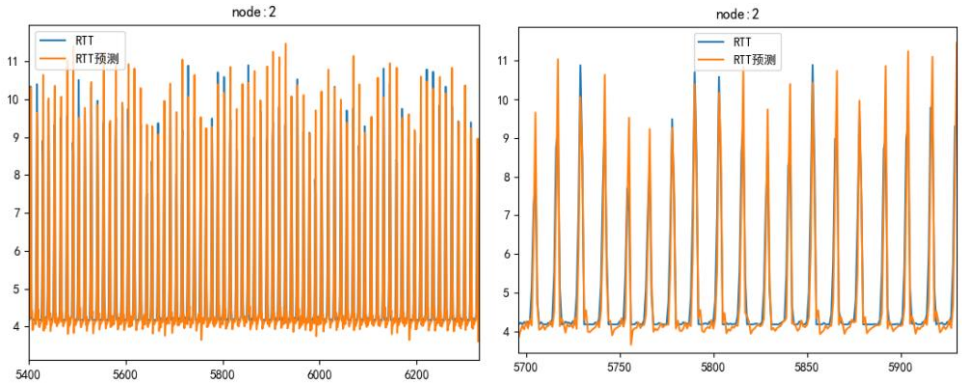

图 17、仿真平台部分 RTT 预测图

通过上图可以观察出，RTT 时间序列预测模块整体预测效果很好，预测曲线几乎与真实曲线重合,预测效果分析：

- 能够预测出 RTT 的迅速增加和降低，即能够预测出网络拥塞程度的变化。
- 在 RTT 峰值和谷值出预测效果值与真实值重合度低，并非预测不准确，是由于速率控制模块控制拥塞导致的，当 RTT 迅速增加时或降低时，预测模块能够准确判断该趋势并给出预测结果，但控制模块为了抑制 RTT 过高增加（拥塞加剧）和 RTT 过低（队列空，链路带宽利用率低，资源浪费）的情况会调整速率，进而影响后几个时刻的 RTT。

4.2.3PID 速率控制模块效果展示

为了尽量保持发送链路交换机端口队列接近空队列，设置$rtt_{target} = 5us$,由于是单个模块效果展示,因此在计算上将 RTTpred 替换为 t 时刻 RTT 采集数据，使用上一章已经调整好的参数$[k_p\ k_i\ k_d] = [-0.358 - 0.060\ \ 0.040]$，观察该 PID 控制算法下节点 2 部分采集数据包的 RTT 时延和 RATE 速率变化：

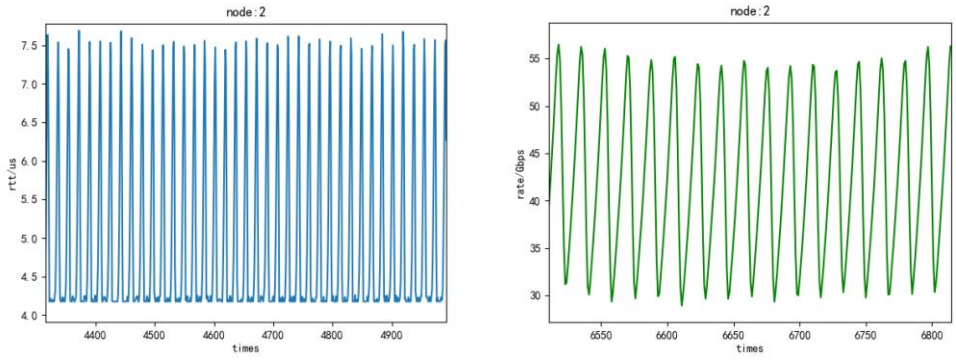

图 18、PIDNN 控制部分时延和速率展示

观察上图发现，节点 2 部分采集数据包的 RTT 时延和发送速率 RATE 速率变化有着相同的规律，即不断上下震动，这是基于 RTT 这一类拥塞控制算法共有的特点（基于 RTT 控制拥塞，当 RTT 增加时，发送速率降低，后 RTT 随着降低，这时算法观测到 RTT 降低又重新增加发送速率，而 RTT 又随之增加，如此

往复，保持在一定的范围内不断震荡）；左图最低值大概在 4us 左右，这是仿真拓扑中固定的链路传播延时，右图最小值为 1Gbps,这是仿真平台控制的最小发送速率。统计所有节点采集数据，其中平均 RTT 为 4.9616 微秒，平均发送速率 14.7977Gbps, 99%RTT 为 7.462 微秒，最大 RTT 为 24.552 微秒。

4.3 算法拥塞控制效果展示

4.3.1 拥塞控制效果指标

拥塞控制效果的指标有以下内容：

- 统计的 RTT 时延和平均发送速率。

  RTT 时延包括了平均 RTT、99%RTT 和最大 RTT。

  由于采集策略，并非对每个数据包都统计了 RTT，且采集点前后数据间隔不同，针对此情况，设计了平均值近似计算方法。

$$RTT_{mean} = \frac{\sum_{i=1}^{N} RTT_i}{N} \tag{29}$$

  发送速率根据最终每个流发送总时间统计，平均速率即为所有流大小除以总用时：

$$RATE_{mean} = \frac{\sum_{i=1}^{20} FLOW\_SIZE_I}{\sum_{i=1}^{N} FCT_I} \tag{30}$$

  99%RTT 是指所有采集 RTT 值进行排序，排在 99%位置的 RTT 数值大小.

  最大 RTT 是指采集点采集到的最大 RTT 值，包括算法启动阶段（如果启动阶段初始发送速率较大时，对于最大 RTT 指标影响较大）：

$$RTT_{max} = \max(RTT_i), i \in N \tag{31}$$

- 基于仿真平台提供的 FCT 分析和跟踪工具。

  平均 FCT 计算：

$$FCT_{mean} = \frac{\sum_{i=1}^{20} FCT_i}{20} \tag{32}$$

  20 个节点同时发送，因此全部发送完毕所需时间:

$$T_{finish} = \max(FCT_i), i \in [1,20] \tag{33}$$

4.3.2 算法整体效果展示

通过上一节可知，各模块在仿真平台上分别部署实现各自功能；该节将各个模块进行合并，同时部署到仿真平台中进行测试，观察算法整体效果以及各模块之间配合情况。

在仿真平台中部署整个基于神经网络的数据中心拥塞控制算法，每当选择不同的rtt_{target}，具体控制效果就不同，观察其对所有发送节点采集报文平均时延及其发送速率的影响,其中平均 RTT 单位为微秒 us,平均发送速率单位为 1Gbps：

表 4、不同rtt_{target}时延及吞吐量展示

rtt_{target}(us)	5	6	7
平均 RTT	5.0302	6.0611	7.1478
平均发送速率	21.1789	15.7269	11.2404
99%RTT	6.276	10.253	12.906
最大 RTT	21.144	23.986	48.699

调整不同rtt_{target}目的是通过增加目标 RTT，观察算法是否会增加吞吐量以提高平均 RTT 向目标 RTT 靠近，但实际情况确实平均 RTT 增加，吞吐量等各项指标都有所下降；该拓扑情况下空队列时延在 4us 到 5us 之间，增加目标 RTT 在算法启动阶段是，会根据 RTT 目标的增加而增快启动速率，但是这会导致后续拥塞情况更加严重，单位速度的增加更大目标 RTT 拥塞增加更严重，这会导致吞吐量以更快的速率下降，最终导致平均发送速率指标更差。

因此，选择一个“适当的rtt_{target}”对于 PID 控制算法的控制效果很重要，这里“适当的rtt_{target}”指的是保持交换机端口队列接近空队列时的数据包 RTT，在 4.1 部分中的仿真拓扑下，“适当的rtt_{target}”为 5us 左右.

FCT 流完成时间分析:

表 5、不同rtt_{target}FCT 指标展示

rtt_{target}(us)	5	6	7
FCT_{mean}(ms)	11.9930	16.1507	22.5970
T_{finish}(ms)	53.7560	53.6880	53.5559

观察上表发现，平均发送速率越大对应平均 FCT 流完成时间越小，但是整体完成时间几乎一样；由于 20 条流是同时开始发送，所以整体完成时间为最大的流完成时间，前三个节点发送流大小远超过后 17 个节点，因此整体完成时间即为前三个节点最大的流完成时间；该拓扑情况下，通过观察发现，发送中间时间时，其余流已经发送完毕，剩余两个大流在继续发送，这时链路吞吐量远超过这两个节点的最大发送速率之和，无论rtt_{target}大小，采集包延时都为空队列延时，因此都在以最大速率发送，这导致最终时延都较为接近。

4.3.2 算法各模块之间配合效果展示

与之前 4.2.3 单一 PID 速率控制模块相比，该节在仿真平台中部署了数据预处理模块和速率控制模块;控制rtt_{target}(us)为 5us，观察这两个模块对于拥塞控制算法效果的影响，效果比较如下：

图 19，PID 控制算法加入预测模块前后对比

加入预测模块四个指标都有不同程度的提升，在加入预测后，平均 RTT 更接近 RTTtarget,但是略有上升，相比之下，平均 RTT 上升 1.38%，吞吐量增加 43.12%，99%RTT 下降 18.9%，最大 RTT 下降 16.12%。最大 RTT 和 99%RTT 都有所降低,这说明加入了 RTT 预测模块后对于拥塞变化反应更快，速率调整更及时；加入了 RTT 预测模块后，当预测 RTT 大于rtt_{target}，速率便会提前下降，对于拥塞具有更快的反应。同时平均发送速率却大幅提高，这说明 RTT 正确预测，使得发送速率更加合理，充分利用带宽的同时能够根据预测值提前调整，且不会加剧拥塞。

4.4 算法比较及分析

4. 4. 1 算法比较

LSTM+PID 初始发送速率为 10Gbps,而其余算法初始发送速率为 100Gbps，基于 4.1 设计的 INCAST 通信模式和流进行网络仿真，最终根据仿真平台统计的 RTT 时延和发送速率如下：

表 6、算法拥塞效果对比

DCN 拥塞控制算法	LSTM+PID ($rtt_{target}=5$)	HPCC	TIMELY	DCTCP
平均 RTT	5.0302	4.3223	11.9797	14.6578
平均发送速率	21.1789	16.3949	15.6302	17.4700
99%RTT	6.276	4.56	102.673	28.054

最大 RTT	21.144	90.48	165.244	86.32

整体比较分析，综合考虑其中算法拥塞控制效果最好的是 HPCC，其次为 LSTM+PID（$rtt_{target} = 5$）；

- HPCC 不仅实现平均 RTT 几乎等于空队列 RTT，还在此情况下保证了一定的发送速率；LSTM+PID（$rtt_{target} = 5$）以 5us 为目标 RTT，目的是在保证较小的拥塞的同时，充分利用带宽，提高吞吐量；这需要目标 RTT 取值较为合适，若过小，则无法充分利用带宽，若过大，则会加剧拥塞情况，反馈自身造成吞吐量下降。
- 比较拥塞控制效果主要参考平均 RTT 和 99%RTT，最大 RTT 受到算法启动阶段的速率影响，LSTM+PID（$rtt_{target} = 5$）以 10Gbps 开始发送，其余三种算法则为 100Gbps,因此前者的最大 RTT 较小，从拥塞控制效过分析，HPCC>LSTM+PID>DCTCP≈TIMELY。
- 上述五种算法中，HPCC 算法通过 INT 协议获取了发送链路上各个交换机端口队列长度等网络信息，这使得该算法拥塞控制最为精确，在 INT 协议的配合下，能够计算出链路中传播的报文数量；这使得在不牺牲发送速率的条件下，保证队列长度长时间接近空队列，充分利用链路带宽。
- 而 LSTM+PID 算法与 TIMELY 算法相似，只通过在发送端统计的 RTT 数据进行速率控制，这对于算法本身要求很高；LSTM+PID 使用神经网络的方法根据已有 RTT 中提取出较多有用信息进行预测和控制，考虑到参数的优化和选择，算法最为复杂，速率控制更为精确；而 TIMELY 算法控制相对模糊，只是根据 RTT 前后变化计算梯度，从而控制速率的增加，而且速率在增加时只有两个增加速率，局限性较大，无法做到快速提高速率；
- TIMELY 效果较差，在 TIMELY 算法中只根据梯度控制速率的加减，受到梯度范围的影响，仿真中 20 微秒为一个梯度，因此时延控制不准确，效果差，在实际应用中需要根据网络调整适应的梯度值。
- 为控制 RTT 速率向rtt_{target}变化，LSTM+PID 算法控制速率变化相对较快，算法输出为一个乘法系数，每次速率直接乘以系数，这与 HPCC 算法类似，但是由于获取的网络数据只有 RTT，因此在速率上不如 HPCC 准确；但是两者比较下，HPCC 需要相关的 INT 协议以及支持 INT 协议的交换机，而且网络数据占据了报文长度，需要其在硬件上配合实现；而 LSTM+PID 无需硬件修改，对于不同种类的交换机都适用，只需在终端节点传输层部署该算法，在软件层面实现。
- 与传统算法不同的是，LSTM+PID 算法无法直接使用，需要对算法参数提前进行训练修正，这使得该算法可以适应不同网络状态及不同种类流。

4.4.2 随机流注入分析

之前仿真结果及分析都是基于 4.1 部分拓扑设计，主要目的是为了观察算法对网络拥塞通信下流的数据包时延 RTT、流完成时间 FCT 及各个节点吞吐量（发送速率）的影响，但是真是数据中心网络环境中不只有 INCAST 通信，更多的是普通的网络流量，这些网络流量发送端和接收端是随机的，为了观察该算法在网络拥塞下（产生 INCAST 流量）对于普通流量的影响。采取论文[8]中的仿真设计：

共 320 个节点，随机时间（范围为 2-2.01s）发送不定大小的普通流，发送方和接收方随机，普通流平均占 30%网络总吞吐量，随机发生 INCAST 通信，共 60 个随机节点同时向一个节点发送一条大小为 500KB 的流，INCAST 流量占据网络总吞吐量的 2%。

320 个节点，每个节点带宽 100Gbps，采取 FAT 网络拓扑结构。

使用仿真平台模拟 Facebook 数据中心网络流量特点，按照 30%平均负载生成 300B-10MB 不定大小流约 10w 条。

SIZE			100B	200B	300B	350B	400B	500B	600B	700B
小于 SIZE 的流数量占比（%）			1	2	5	15	20	30	40	50
1K	2K	7K	30K	50K	80K	120K	300K	1M	2M	10M
60	67	70	72	82	87	90	95	97.5	99	100

表 7、facebook 流量分布统计

INCAST 发生次数 T =320* 100Gbps * 0.02 *(2.01-2) / （500KB*60） = 26.67，使用泊松到达方法的投射在 2-2.01s 内，最终实际产生 34 次。

观察 INCAST 发生对不同大小流 95%FCT slowdown 影响：

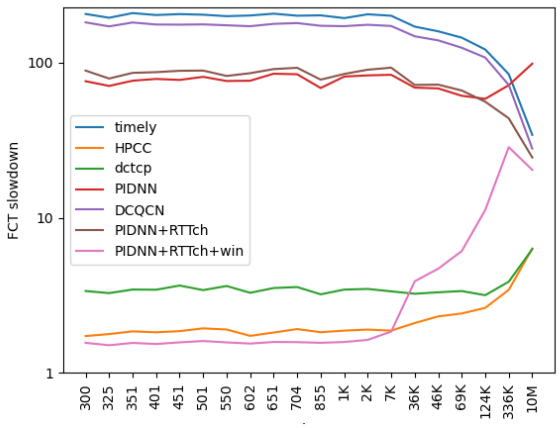

图 20、不同算法对不同大小普通流 95%FCT slowdown 影响

FCT slowdown 为实际流完成时间和网络中只发送该流所需的完成时间之比。

总共测试了 5 种算法，观察发现 PIDNN 算法在流较小时性能很好，相比于 timely 和 DCQCN 同等大小流都能够实现更短的 FCT，但是当流超过 336k 后性能不如 timely 和 DCQCN;观察得知，这是因为 RTTtarget 固定导致的，当流较大时，网络负载加大，平均 RTT 较高，而很低的 RTTtarget 会抑制发送速率的提升，造成吞吐量不足，FCT 升高；因此将 PIDNN 算法加入了 3.5.4 小节设计的 RTT 目标调节机制，作为一种算法优化方式，即 PIDNN+RTTch，当流长时间采集的

数据包 RTT 都无法到达目的 RTT 时，对其进行调整至采集的平均 RTT，最终以损失一些小流的 FCT 为代价，很大程度的降低了大流 FCT slowdown,如上图所示。

PIDNN+RTTch+WIN 是论文[8]中对于没有窗口限制的 RDMA 拥塞控制算法的改进方式，通过增加窗口限制来加强拥塞控制效果，将其应用到 PIDNN 发现其对于小流提高很大，当流小于 7k 时，算法效果甚至优于 HPCC 和 DCQCN,90%FCT 对应的 slowdown 在 1.5 以下，几乎实现了在 INCAST 发生时保持空对列。但是当流的大小超过 7K 后性能就急速下降。图 20 中具体 95%SFCT slowdown 数值如下：

DCN 拥塞控制算法/流大小	PIDNN	PIDNN+RTTch	PIDNN+RTTch+WIN	HPCC	TIMELY	DCTCP	DCQCN
300B	76.064	89.064	1.560	1.725	206.612	3.369	182.315
325B	70.922	79.151	1.505	1.777	195.293	3.265	171.817
351B	76.638	86.132	1.558	1.850	209.355	3.443	181.984
401B	78.706	87.052	1.553	1.842	203.374	3.433	176.648
451B	77.562	88.801	1.571	1.856	206.179	3.654	176.273
501B	81.183	89.110	1.603	1.934	204.360	3.415	176.991
550B	76.312	82.280	1.570	1.901	199.939	3.628	174.634
602B	76.632	85.695	1.543	1.730	201.974	3.280	172.234
651B	85.039	91.107	1.580	1.818	207.490	3.521	178.379
704B	84.354	92.864	1.576	1.914	201.339	3.573	180.477
855B	68.754	77.882	1.561	1.826	202.429	3.207	173.332
1K	81.509	84.753	1.579	1.870	194.271	3.440	172,369
2K	82.972	90.244	1.630	1.900	205.625	3.474	175.972
7K	83.692	93.057	1.844	1.870	201.261	3.351	172.856
36K	69.164	71.998	3.892	2.094	171.223	3.234	148.273
46K	68.353	72.304	4.693	2.310	159.647	3.305	139.570
69K	61.281	66.258	6.095	2.412	145.350	3.367	125.073
124K	58.559	56.103	11.222	2.623	121.854	3.165	108.084
336K	71.712	43.848	28.547	3.428	84.506	3.869	71.945
10M	98.729	24.493	20.386	6.338	34.287	6.273	27.963

表 8、多算法 95%FCT slowdown 值

需要一段说明

4.4.3 分析总结

4.4.3.1 算法优势及缺点

优点：

1. 相较于其他基于时延的拥塞控制算法，本算法使用相同的信息实现对网络拥塞更精准的控制，具有更好的拥塞控制效果。

2. 算法实现只需要编写网络代码，不需要修改设备硬件或网络协议,便于大规模部署，节省成本。
3. 能够预测网络拥塞状态变化，提前进行控制。
4. 将网络拥塞与 AI 智能相结合，适应研究趋势，利于算法的进一步优化和拓展，为后续相关研究提供一些方向和思路。

缺点：

1. 算法结构相对复杂，需要搭建神经网络并进行训练。
2. 算法需要一些先验参数，例如 PIDNN 训练前$[k_p\ k_i\ k_d]$初始值，以及 RTTtarget.

4.4.3.2 **算法创新点**

- 该算法将拥塞控制和 AI 结合到一起，与传统的静态控制算法不同，该算法在控制参数上是可调整的，可以根据实际网络状态不同重新优化参数。
- 该算法引入了预测模块，通常网络链路受到不同的流的影响，因此网络状态是快速变化的，很多算法都是通过希望获取更多的网络参数使得拥塞控制的更加精确；而本算法则是复杂的网络拥塞情况抽象为数据包的返延时，通过对下一时刻 RTT 的预测判断网络拥塞的变化；与其他算法相比能够更快的获取拥塞变化的信息，而非更准确的拥塞信息；这使得拥塞控制具有一定的预知性，使用更少的网络信息实现近似的控制效果。

4.5 本章小结

本章首先展示了各个模块在仿真平台的真实效果，在整合各个算法整体部署，统计其方针下的性能指标及拥塞控制效果；与其他的 DCN 拥塞控制算法进行比较分析优略。最后分析了算法优缺点及创新。

5、全文总结及展望

5.1 全文总结

本文以数据中心网络的拥塞控制现状为研究背景，提出了一种新的基于神经网络的拥塞控制算法。算法实现主要关注点在于如何使用神经网络从较少的终端信息获取更精确的速率控制，以达到更好的拥塞控制效果。

首先，本文总结了相关领域的研究现状，陈述了本论文研究的背景和意义。梳理了解决 DCN 网络拥塞的算法种类，主要包括速率控制和流量工程；介绍的 DCN 的基本拥塞控制需求。

其次，本文阐述了算法设计的整体框架，将整体功能分解为多个模块，在网络条件约束下设计各个模块的解决算法。通过模块化的架构图、数学表达式的形式对搭建的算法模型进行详细介绍；再根据神经网络的流程对 RTT 时间序列预测模块和 PID 数据包发送速率控制模块进行了训练并展示效果。

然后，对于算法拥塞控制效果使用基于 NS3 的仿真平台进行网络仿真，搭建 INCAST 通信模式对算法进行测试验证，在仿真结果的基础上同其他 DCN 网络的拥塞控制算法比较，总结了算法的优缺点和创新性。

5.2 后续工作展望

数据中心拥塞控制算法和机制一直在不断发展和进步，但目前为止，已经出现了数十种 DCN 拥塞控制算法，但很少见到基于神经网络方面的研究，因此本文提出的基于神经网络的数据中心拥塞控制算法没有很多已经实现的类似算法进行参考，算法整体流程上为个人单独设计，存在很大的进步空间；且算法的验证目前还是基于他人的 NS3 仿真平台，后续工作可以再真实的网络环境中测试，进一步的验证算法的拥塞控制效果并进一步对算法进行优化。

经整理和总计，后续工作主要设想有：

对算法进行进一步的理论论证和算法优化。在 PIDNN 网络参数优化过程中，为了保证参数的收敛，做了很多方面的限制；未来可能寻求一些更好的 PID 参数优化方法。

真实的网络测试环境。虽然后续依然无法在真实的 DCN 网络中测试，但是可以使用 OVS 等搭建真实的网络环境，脱离对基于 NS3 仿真平台的依赖，也使得测试结果更加真实、更具有说服力。

更多样的网络拓扑。本论文中，针对拥塞设计了 20 固定个节点的 INCEST 拓扑结构，只有一个接收方。虽然在 4.4.2 部分随机流注入分析引入了新的 320 节点 FAT 拓扑，但是在实际数据中心网络中，网络拓扑更加复杂多样，因此后续可能考虑不同的网络拓扑。

致谢

参考文献：

[1]开放数据中心委员会. 数据中心只能无损网络白皮书. [R/OL]. (2021-09-15), ODCC-2021-05001. https://www.ambchina.com/index.php?id=338265

[2] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. SIGCOMM Comput. Commun. Rev. 45, 4 (October 2015), 523–536. https://doi.org/10.1145/2829988.2787484

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In Proceedings of the ACM SIGCOMM 2010 conference (SIGCOMM '10). Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[4]Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. SIGCOMM Comput. Commun. Rev. 45, 4 (October 2015), 537–550. https://doi.org/10.1145/2829988.2787510

[5]Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. 2012. Deadline-aware datacenter tcp (D2TCP). In Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '12). Association for Computing Machinery, New York, NY, USA, 115–126. https://doi.org/10.1145/2342356.2342388

[6]Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. PFabric: minimal near-optimal datacenter transport. In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM '13). Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/2486001.2486031

[7]Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: dynamic flow scheduling for data center networks. In Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10). USENIX Association, USA, 19.

[8] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19). Association for Computing Machinery, New York, NY, USA, 44–58. https://doi.org/10.1145/3341302.3342085

[9] Shuai Wang, Kaihui Gao, Kun Qian, Dan Li, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, Dai Zhang, Binzhang Fu, Frank Kelly, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. 2022. Predictable vFabric on informative data plane. In Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22).

Association for Computing Machinery, New York, NY, USA, 615–632. https://doi.org/10.1145/3544216.3544241

[10] Xiao L , Wang Z , Peng X . Research on congestion control model and algorithm for high-speed network based on genetic neural network and intelligent PID[C]// Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on. IEEE Press, 2009.

[11] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. 2021. ACC: automatic ECN tuning for high-speed datacenter networks. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 384–397. https://doi.org/10.1145/3452296.3472927

[12]C. Lee, C. Park, K. Jang, S. Moon and D. Han, "DX: Latency-Based Congestion Control for Datacenters," in IEEE/ACM Transactions on Networking, vol. 25, no. 1, pp. 335-348, Feb. 2017, doi: 10.1109/TNET.2016.2587286.

[13]杜鑫乐,徐恪,李彤,郑凯,付松涛,沈蒙.数据中心网络的流量控制:研究现状与趋势[J].计算机学报,2021,44(07):1287-1309.