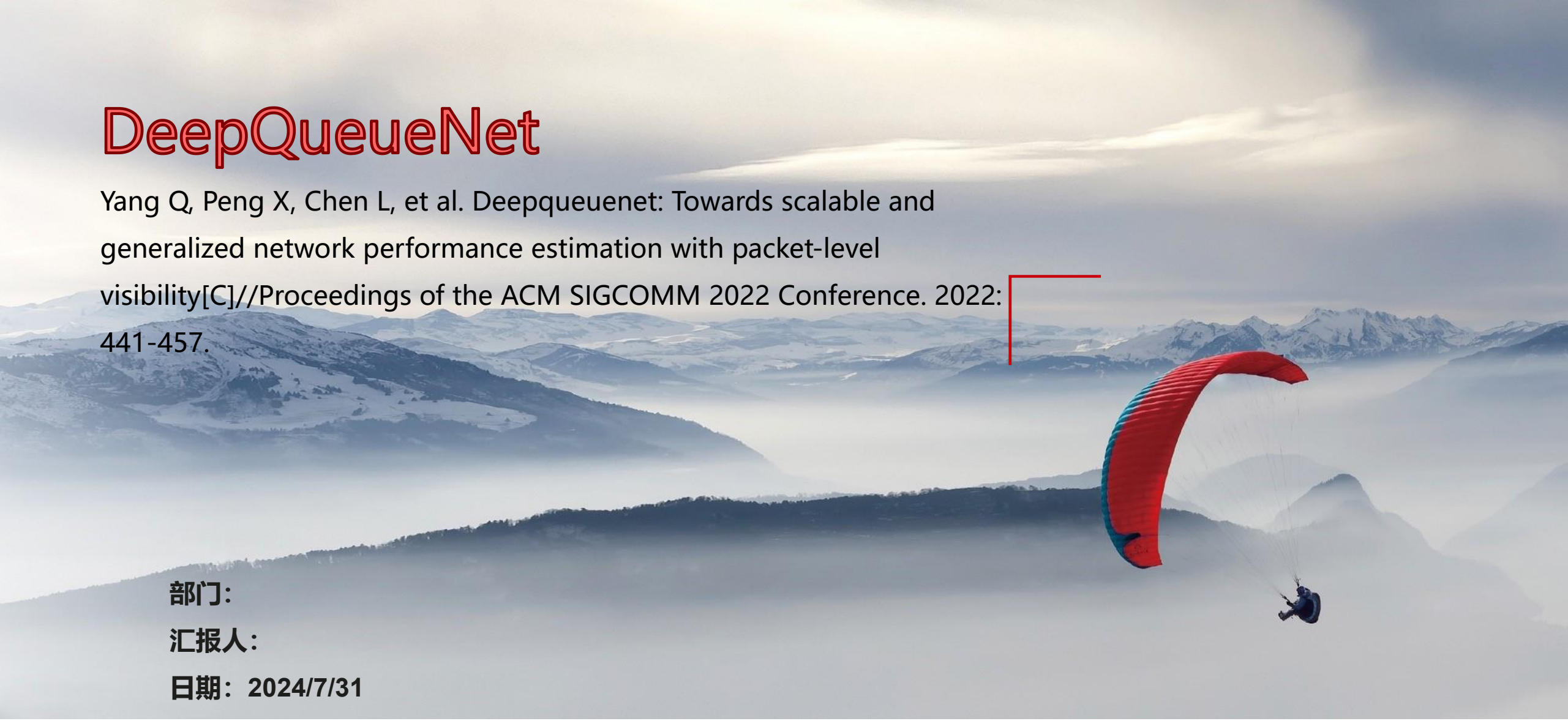
DeepQueueNet

Yang Q, Peng X, Chen L, et al. Deepqueuenet: Towards scalable and generalized network performance estimation with packet-level visibility[C]//Proceedings of the ACM SIGCOMM 2022 Conference. 2022: 441-457.



部门：

汇报人：

日期：2024/7/31

HUAWEI

背景

- 网络模拟器是网络运营商的关键工具，用于辅助如容量规划、拓扑设计和参数调优等重要任务。

- 传统的基于离散事件模拟（DES）的网络模拟器在现代大规模网络面前面临性能瓶颈，主要是因为离散事件之间的依赖性导致即使分布式和并行化也无法显著提高性能。

- 使用机器学习和连续模拟技术构建的端到端性能估计器（EPEs），虽然实现了可扩展性，但在包级别的可见性和通用性方面存在问题，如无法回答具体设备或流量的问题，也不能适应网络配置的变化。

三种类型的网络仿真系统

包级DES，流级连续模拟器，基于DNN的EPEs

包级DES

- 提到了几种常用的DES软件，包括NS2/3、OMNeT++。
- 解释性：包级别的DES是最具解释性的模拟类型，因为它在模拟器中的实体与实际网络中的实体之间建立了直接的一一对应关系，使得用户能够深入理解模拟结果。
- 灵活性：包级别DES允许指定任何流量管理(TM)机制，因为它按顺序处理每个数据包。
- 可扩展性问题：随着模拟规模的增大，包级别DES面临严重的可扩展性问题。尽管开发了并行和分布式DES (PDDES) 来提高模拟效率，但由于进程间的消息传递和同步开销，PDDES不一定能提供速度提升，在某些情况下甚至可能比顺序模拟慢。PDDES的性能和可扩展性受到细粒度通信的限制，尤其是在通信成本较高的执行环境中。

流级连续模拟器

- 控制理论估算器用于设计传输层流量控制协议，要求用户通过偏微分方程（PDE）来明确指定系统的状态和动力学。这类方法的实用性有限，因为它们需要专业知识来定义所有流量的动力学，而且只能提供稳态解，无法反映实际的流量和网络设备的流量管理机制。

- 网络微积分（NC）通过使用广义递增函数，提供了一个分析网络中最坏情况下的延迟和缓冲区大小界限的理论框架，但它只能给出较宽松的界限，无法提供平均或任何百分位的估计。

- 排队理论估算器利用排队模型来评估网络性能，可以非常准确地近似延迟和丢包率，但时间复杂度随着队列数量和缓冲区大小呈指数增长。

端到端性能估计器

- 将先进的深度神经网络 (DNN)架构应用于端性能估计器 (EPE)，例如：

  SimNet 使用卷积神经网络 (CNN) 预测计算机体系结构模拟的结果。

  RouteNet 利用图神经网络 (GNN) 来预测网络性能指标，如延迟、抖动和丢包率

- 优势：DNN 可以利用高效并行推理基础设施，实现高可扩展性。

- 不足之处
  - 包级别可见性：现有 EPE 缺乏对特定设备或流量细节的洞察力，且当需要关注新的性能指标时，必须重新训练模型。
  - 通用性问题：

    拓扑变化：当网络拓扑改变时，EPE 需要重新训练。

    流量模式变化：流量模式的变化会导致现有 EPE 的准确性下降。

    流量管理机制：大多数 EPE 假设 FIFO 队列策略，不能很好地适应更复杂的调度机制。

端到端性能估计器

MimicNet 方法

- 结合 DES 和 EPE：MimicNet使用DES对数据中心网络的一个子集（集群）进行包级别的精确模拟，包括详细的队列和传输层动态。然后使用从这个可观测子集中收集的数据来训练“模仿者”，这些模仿者是网络内部和跨集群行为的近似器。最后，他们将可观测的集群与模仿集群组合起来形成全规模的数据中心网络模拟。

- 性能与准确性：这种方法显著减少了模拟时间，但牺牲了一定的准确性。

- 局限性：MimicNet 主要针对 FatTree 拓扑，对于其他网络拓扑的支持有限。

目标

DeepQueueNet旨在结合以下三类网络模拟器的优点：

- 可见性和通用性类似于离散事件模拟（DES）；
- 准确性类似于排队理论模型；
- 可扩展性类似于基于深度神经网络（DNN）的端到端性能估计器（EPEs）。
- 同时，DeepQueueNet试图避免这三类模拟器的缺点。

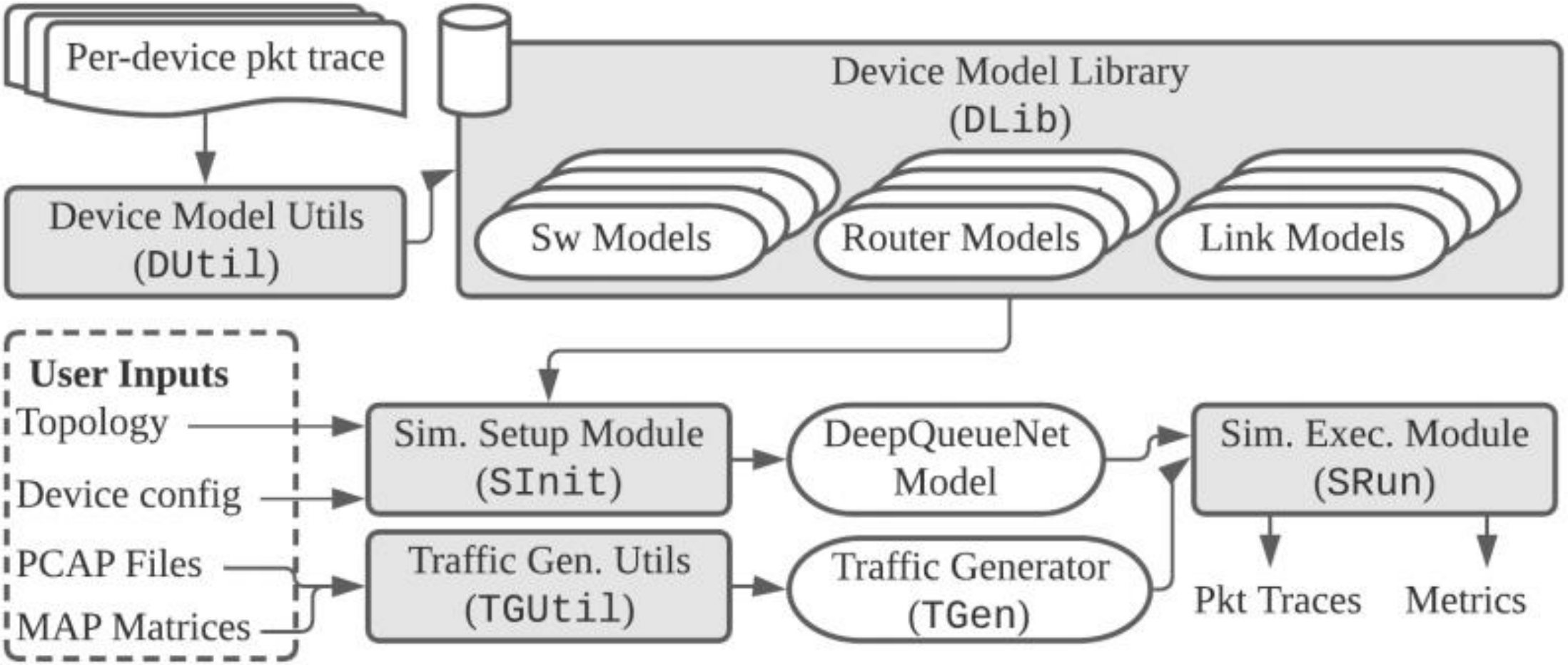
DeepQueueNet的限制和应用

- DeepQueueNet 主要关注网络达到稳定状态时的性能分布，并通常忽略网络的瞬态状态。这加快了网络的模拟速度。（忽略了一些高层应用、传输协议和网络规模动态流量转向之间的状态相关交互）

- DeepQueueNet主要关注的是网络层的信息（即数据包所经过的路径、数据包的大小、数据包的间隔到达时间以及数据包到达和离开路径上每个设备的时间）

- 适用场景：DeepQueueNet 适合于当网络达到稳定状态时性能分布的评估，例如容量规划和拓扑设计。

系统设计——系统组件


Per-device pkt trace
Device Model Utils
(DUtil)
Device Model Library
(DLib)
Sw Models
Router Models
Link Models
User Inputs
Topology
Device config
PCAP Files
MAP Matrices
Sim. Setup Module
(SInit)
Traffic Gen. Utils
(TGUtil)
DeepQueueNet
Model
Traffic Generator
(TGen)
Sim. Exec. Module
(SRun)
Pkt Traces
Metrics

系统设计——假设

- 路由表和优先级/权重分配：假设在模拟开始阶段已经给定了路由表以及用于数据包调度的优先级/权重的分配，并且在整个模拟过程中保持稳定。

- 关注点：DeepQueueNet专注于大规模网络的性能模拟，其基础建立在排队理论之上，重点关注系统达到稳定状态时的性能分布，通常忽略系统的瞬态行为。

- 批处理：DeepQueueNet以批处理方式处理数据包以实现高可扩展性，这也意味着它不适合需要详细交互和状态维护的协议模拟，例如传输层协议或路由协议。

系统设计——数据包流建模

数据包包表示：特征向量
- 唯一标识符 (PID)
- 流标识符 (FID)
- 数据包长度 (LEN)
- 传输协议 (TRP)

$$p=<\text{pid},\text{fid},\text{len},\text{trp}>$$

数据包流：数据包流是一系列按时间顺序排列的数据包到达记录。对于具有 n 个数据包的流，可以用一个有序对列表表示，其中包含了数据包及其到达时间。

$$\tau=[(p_0,t_0),(p_1,t_1),(p_2,t_2),\cdots,(p_n,t_n)],where\ t_i\leq t_{i+1},\forall i\geq 0$$

端口模型：网络设备（如路由器）通常有多个端口，这些端口被分为逻辑入口端口和逻辑出口端口。所有进入设备的数据包流组成 `Tin`，所有离开的数据包流组成 `Tout`。

$$T_{in}=[\tau_{0,in};\tau_{1,in};\cdots;\tau_{k-1,in}]$$

$$T_{out}=[\tau_{0,out};\tau_{1,out};\cdots;\tau_{k-1,out}]$$

系统设计——设备建模

链路模型

链路是最简单的设备类型，只有一个输入端口和一个输出端口。链路的延迟取决于链路长度 l、链路上的传播速度 c 和带宽 C。链路设备的操作是为所有数据包在入站时间序列中添加延迟，延迟基于每个数据包的长度和链路的参数。

$$\begin{aligned}\tau_{\text{out}} &= \tau_{\text{in}} + [\vec{0}, len(\tau_{\text{in}})/C + l/c] \\ &= [\cdots, (p_{i,\text{in}}, t_i + len(p_{i,\text{in}})/C + l/c), \cdots]\end{aligned}$$

这里，len() 是一个纯函数，用于提取数据包的长度；零向量表明链路不会修改数据包向量。

系统设计——设备建模

多端口网络设备

模型包含两个子模型：数据包级转发模型 (PFM) 和数据包级传输模型 (PTM)。

数据包级转发模型 (PFM)

- 在转发之前，每个入站端口的数据包流通过在数据包向量中添加入口端口 ID $<pid,fid,len,trp,in\_port>$
- 数据包填充：所有入站数据包流被填充至相同的长度，使用空数据包。
- 转发表：假设每个设备的转发表在模拟开始之前生成或给定。转发表是一个函数，接受数据包的流 ID 和入口端口 ID，输出出口端口 ID。 $forward(fid,in\_port) \rightarrow out\_port$

- 转发张量：使用增强后的入站数据包流和转发函数，可以生成转发张量 $F=[f_{i,j,k}]$，这是一个三维的 0-1 张量，形状为 K × K × N，其中 K 是设备端口的数量，N 是入站数据包流中的数据包数量。如果 `f{i,j,k} = 1`，表示第 `k` 个数据包从端口 `i` 被转发到端口 `j`；否则，`f{i,j,k} = 0`。
- 转发操作：将设备的入口流和出口流的集合分别表示为Tin和Tout，通过张量乘法混合入站数据包流以形成出站数据包流，基于设备的转发表。

$$T_{out}=F \cdot T_{in}+Delay(\cdots)$$

系统设计——设备建模

数据包级传输模型 (PTM)

- 数据包增强：PTM 开始时通过增加与传输机制相关的特征来增强数据包向量。
- 数据包处理：使用预训练的 DNN 模型处理入站数据包流，为每个数据包添加延迟。
- 后处理：通过一个后处理阶段（PTM 后处理），精炼出站数据包流，以减少下游设备中的误差传播。

- PFM 和 PTM 共同实现了设备级别的数据包处理，使得 DeepQueueNet 能够高效地模拟网络性能，特别是利用批处理来提高可扩展性。

系统设计——网络建模

Algorithm 1: Iterative Re-Sequencing Algorithm (IRSA)

Input :G (Network Topology), I (Ingress Packet Stream)
Output:L (Packet Stream in Network)

```
/* Initial Inference                                   */
foreach device n in G do
    Perform n's model inference using I;
    Update n's egress packet stream in L;
end
/* Iterate until convergence                           */
for diameter (G) do
    Initiate L';
    foreach device n in G do
        Perform n's model inference using L;
        Update n's egress packet stream in L;
    end
end
def diameter(G):
    /* returns diameter of graph G.                    */
```

基于目标网络拓扑结构直接构建的。它通过连接预训练的链路模型和设备模型来形成一个深度神经网络（DNN）架构，该架构与目标网络在链路级别、端口级别和设备级别上一一对应。

根据网络拓扑结构和入站数据包流来预测网络中的数据包流。

系统设计——批处理数据包

批处理数据包是使 DeepQueueNet 具有高可扩展性的关键因素

——批处理也会带来“误批”问题。

迭代重排序算法 (IRSA)

- 核心思想：通过迭代调整批次并在多次迭代中运行推断直到收敛。
- 算法步骤：
  - 在第 t 次迭代中，所有设备（包括链路）都会从其上游设备拉取第 t-1 次迭代的数据包流。
  - 收集所有数据包流后，每个设备将入站数据包流分割成批次，并执行推断以获得这些数据包的输出序列。
  - 这些输出序列将在下一次迭代中被传递到相应的下游设备。

PTM

数据预处理

- 调度器类型编码：使用 one-hot 编码表示不同类型的调度器，并根据调度器类型添加相应的优先级或权重特征。

- 工作负载计算：通过指数移动平均 (Exponential Moving Average, EMA)计算工作负载特征，以反映数据包到达时系统的工作状态。（平滑因子为 0.95。这表示了入站数据包流中字节数的一个平滑的长期趋势。）

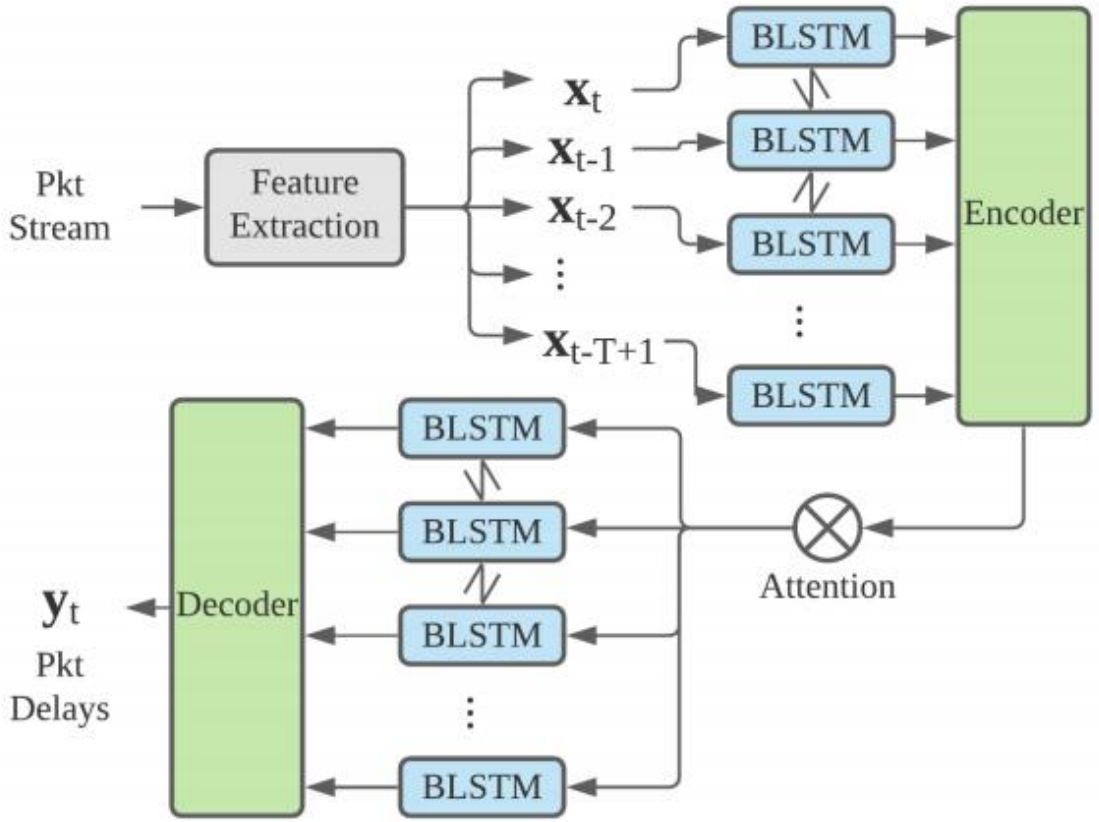
- 数据归一化：通过 MinMaxScaler 归一化数据，确保所有特征都在同一尺度上，从而加速模型的训练过程。

$$x_i' = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

其中 $x_i^{'}$ 是归一化后的值，x_i 是原始数据点，min(x) 和max(x) 分别是数据集中的最小值和最大值。

PTM架构

- 为了封装网络设备复杂的流量管理机制，我们将数据包流视为时间序列，并使用深度神经网络 (DNN) 作为预测模型 (PTM)。这是由于准确的排队模型具有指数级别的复杂度。
- 目标：PTM 需要预测每个事件（数据包）在入站时间序列中所增加的停留时间的时间序列。这是一个序列到序列 (seq2seq) 处理任务。



- 模拟器：使用了轻量级的 Python 模拟器 ns.py 生成训练数据。
- 训练方法：通过回归方法预测数据包在网络设备中的延迟。我们使用设备中数据包的停留时间作为响应变量，从特征提取模块中提取的特征作为预测因子。

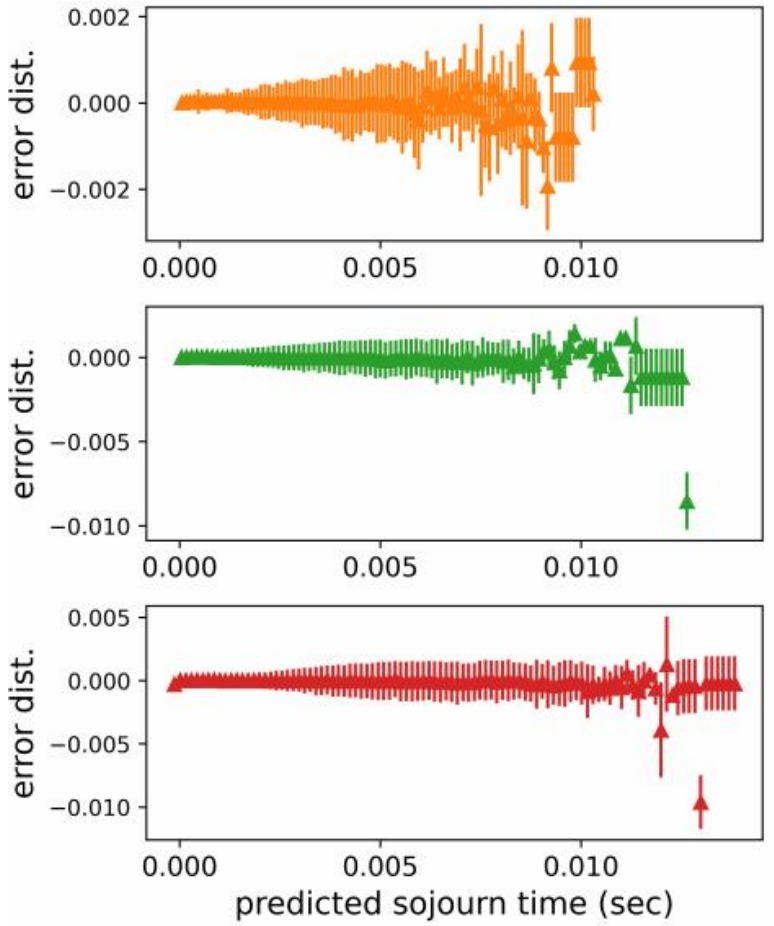
Post-PTM 统计误差校正

- 在网络仿真中，预测数据包在网络设备上的停留时间非常重要。然而，预测误差会随着数据包在网络中的传输而累积。为了解决这个问题，我们引入了一个统计误差校正（SEC）方法

1. 误差收集：当 PTM 训练收敛后，收集每个预测停留时间对应的误差。这些误差按照预测停留时间进行分组，类似于左图所示的数据。

2. DBSCAN 聚类：使用 DBSCAN 算法对附近的预测停留时间的误差进行聚类，形成不同的“箱”（bins）。DBSCAN 是一种密度聚类算法，可以识别出具有相似预测误差的区域。

3. 误差修正：对于落入特定箱内的预测停留时间，计算该箱内所有预测误差的平均值，并从预测中减去这个平均误差。这样做的目的是通过考虑历史误差数据来调整预测值。



HUAWEI

训练 DeepQueueNet

- 训练平台：使用配备有 Nvidia Tesla V100-32GB GPU 的测试平台（Testbed 1）进行训练。
- 推理平台：使用配备有 4 个 Nvidia Tesla V100-16GB GPU 的另一测试平台（Testbed 2）执行推理阶段。
- 模型结构与超参数

Time steps	21
BLSTM	(200,100)
Multi-head attention	3, (64,32)

Table 1: Hyper-parameters of DeepQueueNet.

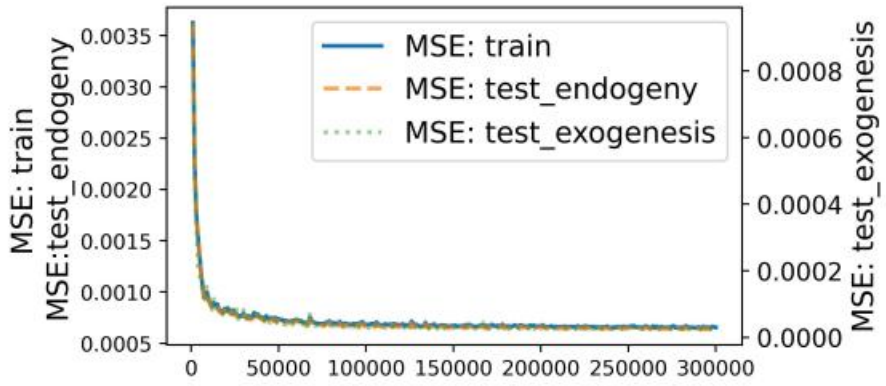
- 数据集与训练细节

- 数据集：基于 ns.py 模拟器中的 K 端口交换机收集了 3,500 条数据包流样本。
- 数据包到达模式：数据包到达端口遵循三种随机过程：Markovian 随机过程 (MAP)、泊松过程和开关过程。
- 负载因子：通过调整这些随机过程的强度，使得每个端口的负载因子在 [0.1, 0.8] 的范围内变化。
- 数据包流长度：每条数据包流的时间长度为 max{1.6e4, 4K * 1e3} + 时间步长 -1单位时间，即每个数据包流中有 max{1.6e4, 4K * 1e3} 个数据点用于序列到序列的学习。
- 路由方案与流量矩阵：数据集中包含了 3,500 种随机生成的路由方案和多种不同强度的流量矩阵。
- 调度策略：配置了交换机的数据包调度策略，包括先进先出 (FIFO)、严格优先级 (SP)、缺省比率轮询 (DRR) 和加权公平队列 (WFQ)。
- 训练/验证划分：使用 80% 的样本进行训练，剩余 20% 的样本用于验证。

训练 DeepQueueNet

- 训练过程与评估
  - 损失函数与优化器：采用均方误差 (MSE) 作为损失函数，使用 Adam 优化器进行优化，固定学习率为 0.001。
  - 批量大小：使用 256 个样本估计梯度。
  - 收敛时间：对于 2 端口、4 端口和 64 端口的交换机模型，分别需要大约 1 小时 40 分钟、3 小时 22 分钟和 11 小时 24 分钟达到收敛。
  - 损失曲线：下图显示了 4 端口交换机模型训练过程中的损失变化。
  - 评估指标：使用归一化的 Wasserstein 距离 ω1来衡量模型的准确性，

    其中ω1= $W_1(\text{prediction}, \text{label})/W_1([0]^{\text{len(label)}}, \text{label})$。 如果预测是准确的(接近ground-truth delayes)，那么ω1接近0(越低越好)。



训练 DeepQueueNet

	Device	No. of class	w_1	w_1 (Refined)
FIFO	2-port	1	0.006967	-
	4-port	1	0.022549	0.009185
	8-port	1	0.032825	0.016606
	16-port	1	0.047078	0.032306
	32-port	1	0.051662	0.049363
	64-port	1	0.053358	0.052345
Multi-level	4-port	2	0.028651	0.016296
	4-port	3	0.036128	0.024085

实验

- 通用性：通过广泛的实验，我们展示了 DeepQueueNet 在面对流量生成模型变化、拓扑变化、流量管理配置变化时其估计准确性（以 ω1衡量）依然保持较高水平，无需重新训练模型。
- 可扩展性：我们展示了 DeepQueueNet 可以通过多 GPU 并行加速。在部署于 4-GPU 集群上时，DeepQueueNet 展示了几乎线性的加速效果。

	numToRsAndUplinks	numOfServersPerRack	numClusters
FatTree16	2	4	2
FatTree64	4	4	4
FatTree128	4	4	8

Table 3: MimicNet's Parameter setting for different sizes of FatTree.

- 实验设置
  - 实验平台：Testbed 2，且使用相同的超参数设置。
  - 真实数据生成：使用 ns.py 模拟器生成真实数据，网络中的链路带宽为 10Gbps。
  - 比较对象：对于一般拓扑，我们的主要比较对象是 RouteNet；对于 FatTree 拓扑，比较对象是 MimicNet 。
  - RouteNet 的设置与训练：按照最新论文《RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN》中的方法设置并训练 RouteNet。
  - MimicNet 的设置与训练：确保流量生成过程与 DeepQueueNet 一致，并使用默认的 MimicNet 模型和训练方法 。MimicNet 的参数设置总结在表 3 中。

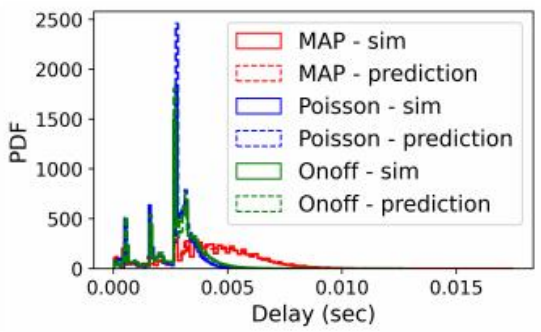
DeepQueueNet的通用性评估

流量生成模型

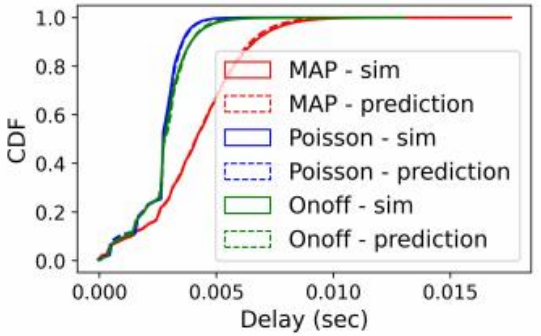
- 使用的模型：
  - MAP（马尔可夫到达过程）：能够很好地拟合实际的流量轨迹。
  - Poisson(泊松分布)：通过变化的强度 λ 产生不同负载的流量，使链路负载从 0.1 到 0.9 不等。
  - Onoff(开关模型)：具有“开”状态转移概率 0.2 和“关”状态转移概率 0.5。

- 实验设置：流量源和目的地随机均匀选择。
- 网络拓扑：使用了一个 FatTree 网络，k=4，共有 16 台服务器（即 FatTree16）。
- 队列管理策略：作为基线配置使用了先进先出（FIFO）队列管理策略。

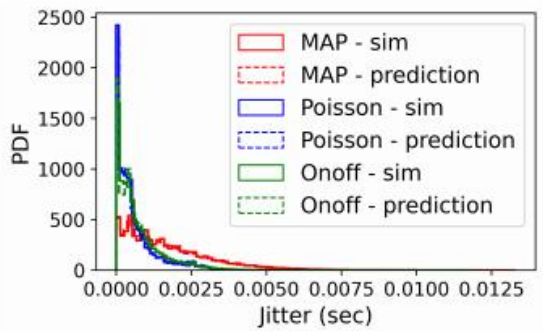
DeepQueueNet的通用性评估



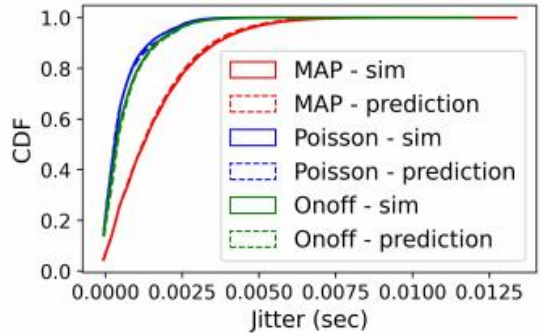
(a) PDF of delay.



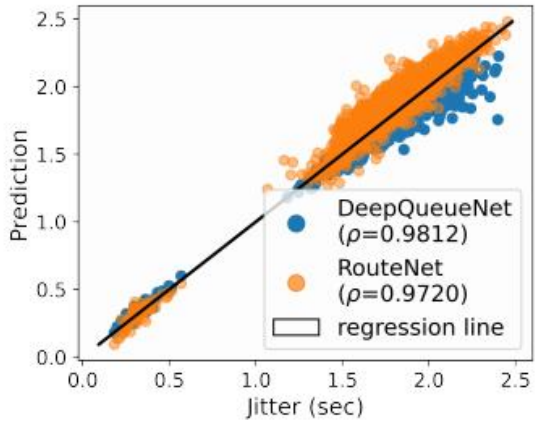
(b) CDF of delay.



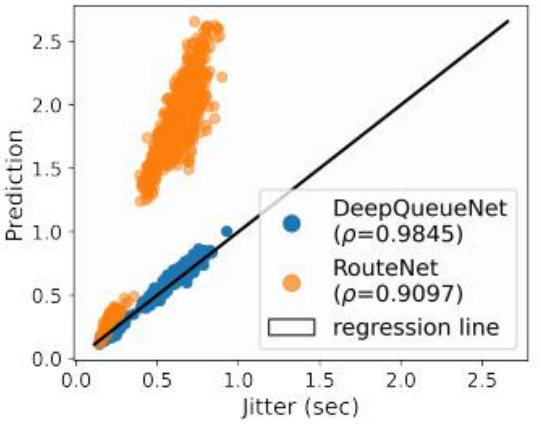
(c) PDF of jitter.



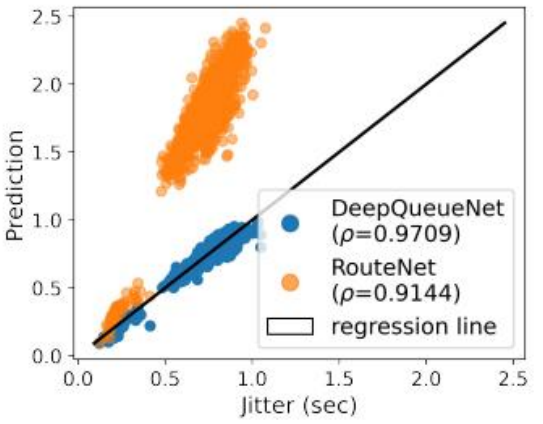
(d) CDF of jitter.



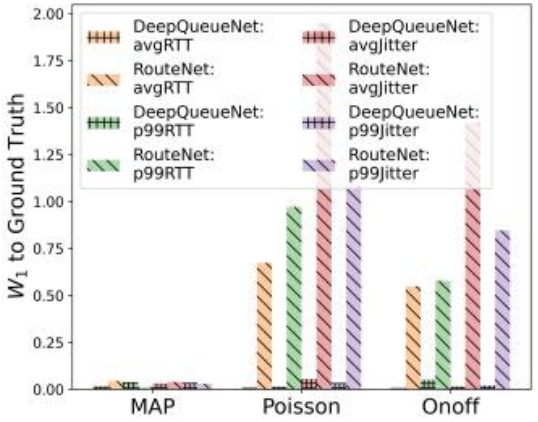
(e) MAP: avgJitter.



(f) Poisson: avgJitter.



(g) Onoff: avgJitter.



(h) W_1: RouteNet vs. DeepQueueNet.

DeepQueueNet的通用性评估

		avgRTT (w_1)	p99RTT (w_1)	avgJitter (w_1)	p99Jitter (w_1)
DQN	MAP	0.017	0.039	0.031	0.035
	Poisson	0.009	0.014	0.055	0.035
	Onoff	0.006	0.051	0.016	0.022
	BC-pAug89	0.003	0.012	0.009	0.007
	Anarchy	0.026	0.068	0.097	0.030
RN	MAP	0.044	0.014	0.041	0.030
	Poisson	0.674	0.972	1.951	1.083
	Onoff	0.549	0.578	1.420	0.847

展示了DeepQueueNet(DQN)和RouteNet(RN)在不同流量生成模型下的Wasserstein距离。这个表格对比了两个模型在处理不同流量生成模型时的通用性。数值越小说明模型的预测越接近实际情况。从表中可以看出，DeepQueueNet在各种情况下都取得了较低的Wasserstein距离，显示出优于RouteNet的泛化能力。

DeepQueueNet的通用性评估

拓扑通用性

- 实验设置
- 拓扑类型：研究了三种不同的网络拓扑以及来自Internet Topology Zoo的两个广域网拓扑
  - Line：线性拓扑。
  - 2d Torus：二维环形拓扑。
  - FatTree6：FatTree拓扑，k=6。
  - Abilene：一个广域网拓扑。
  - GÉANT：另一个广域网拓扑。
- 流量生成模型：使用Poisson到达过程作为基线配置。

DeepQueueNet的通用性评估

		avgRTT (w_1)	p99RTT (w_1)	avgJitter (w_1)	p99Jitter (w_1)
DQN	Line4	0.0003	0.0023	0.0037	0.0000
	Line6	0.0002	0.0016	0.0041	0.0002
	Abilene	0.0017	0.0042	0.0137	0.0011
	GÉANT	0.0009	0.0013	0.0023	0.0032
	2dTorus(4x4)	0.0041	0.0274	0.0233	0.0158
	2dTorus(6x6)	0.0066	0.0277	0.0319	0.0156
	FatTree16	0.0086	0.0145	0.0548	0.0349
	FatTree64	0.0176	0.0395	0.0291	0.0215
	FatTree128	0.0133	0.0532	0.0395	0.0438
RN	Line4	0.5266	0.1978	5.3139	2.2943
	Line6	0.6561	0.2555	2.9424	1.4225
	Abilene	0.4781	0.1984	4.4904	5.5434
	GÉANT	0.4589	0.1328	4.1983	5.4555
	2dTorus(4x4)	1.2527	1.4644	5.5023	2.3050
	2dTorus(6x6)	1.1005	0.5959	4.6492	2.6154
	FatTree16	0.6737	0.9723	1.9510	1.0834
	FatTree64	0.1406	0.3370	5.2964	2.2873
	FatTree128	0.9824	0.6397	6.7684	1.8705
MN	FatTree16	0.0090	0.0135	0.1559	0.1172
	FatTree64	0.0167	0.0179	0.1687	0.0625
	FatTree128	0.0172	0.0194	0.1628	0.0667

- DeepQueueNet在所有场景中都有较低的Wasserstein距离，表明其预测接近实际值。
- RouteNet在一些场景中表现不佳，特别是在复杂网络如FatTree16和FatTree64中。
- 对于FatTree拓扑，MimicNet的准确性较高，但只适用于FatTree拓扑，而DeepQueueNet在其他拓扑中也有很好的表现。

HUAWEI

DeepQueueNet的通用性评估

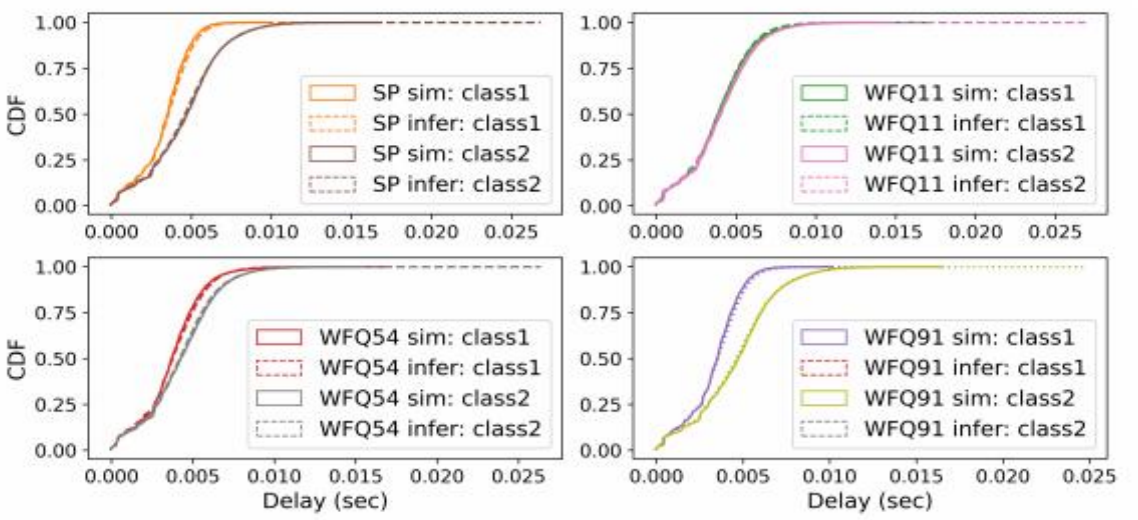
流量管理的通用性

实验设置

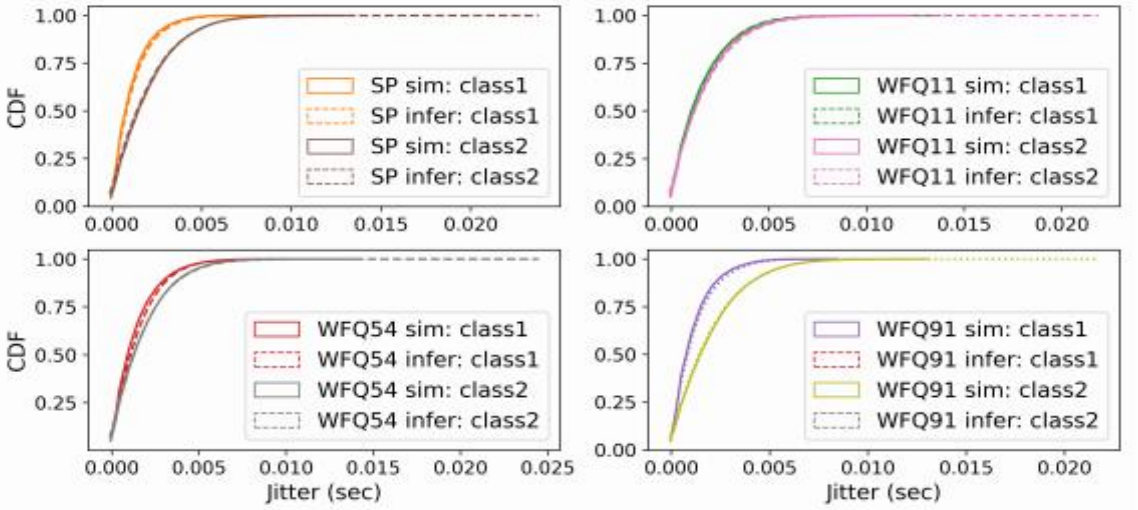
- 流量调度器类型：选择了两种类型的包调度器，代表DRR和WRR的SP和WFQ。
  - 严格优先级 (SP) 和加权公平队列 (WFQ)
  - 对于二类 WFQ，我们改变了权重比例，分别为 1:1、5:4 和 9:1。对于三类 WFQ，我们使用 1:1:1 的权重比例。
  - 对于 SP，我们将不同优先级的流量标记为相等。

- 拓扑和流量源：固定拓扑为FatTree16，并且所有的流量源都是MAP。

DeepQueueNet的通用性评估



(a) CDFs of delays.



(b) CDFs of jitters.

从图中可以看出，不论采用哪种流量管理配置，DeepQueueNet的推理结果与模拟结果都非常接近，这表明DeepQueueNet在不同流量管理配置下都能提供准确的延迟和抖动预测。

DeepQueueNet的通用性评估

		avgRTT (w_1)	p99RTT (w_1)	avgJitter (w_1)	p99Jitter (w_1)
2-class	WFQ	0.020	0.046	0.037	0.042
	SP	0.023	0.050	0.040	0.045
3-class	WFQ	0.027	0.050	0.048	0.041
	SP	0.028	0.029	0.032	0.026

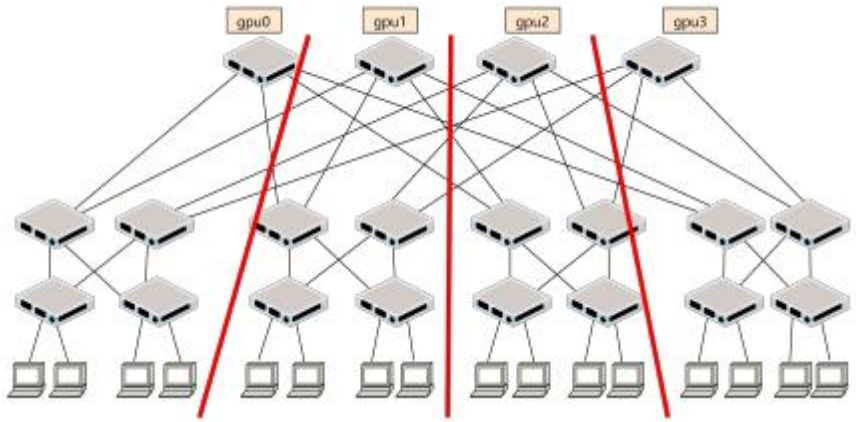
表展示了DeepQueueNet在End-to-End延迟/jitter预测上的性能。这里使用了归一化的Wasserstein距离来进行评估。表中列出了两种不同类型的流量管理机制（2-class和3-class），以及对应的调度算法（WFQ和SP）。

DeepQueueNet的可扩展性评估

- DeepQueueNet 在模拟规模方面具有很高的可扩展性，因为使用 DeepQueueNet 进行估计就是模型推断，可以利用流行的机器学习框架的分布式运行时在多个设备或机器上并行运行模型推断。

实验设置

- 在 FatTree16/128 拓扑上展示了 DeepQueueNet 的并行加速能力。对于每种网络配置，都在相同的生成工作负载集上运行 DeepQueueNet、OMNet++ 和 MimicNet。
- 模型分割：DeepQueueNet 模型被均匀分割到 GPU 上。例如，下图展示了 FatTree16 拓扑在 4 个 GPU 上的分割情况。目前，这种分割是手工完成的，而自动分割和平行化则作为未来的工作。
- 并行加速示例：表 7 中展示了 FatTree16/128 拓扑上的并行加速结果



DeepQueueNet的可扩展性评估

topology	method	# GPUs	time	speedup
FatTree16	DES	0	2h22m11s	-
	MimicNet	1	4m2s	-
	DeepQueueNet	1	5m12s	baseline
	DeepQueueNet	2	2m45s	1.89-fold
	DeepQueueNet	4	1m27s	3.59-fold
FatTree64	DES	-	9h23m53s	-
	MimicNet	1	11m18s	-
	DeepQueueNet	1	29m17s	baseline
	DeepQueueNet	2	15m12s	1.93-fold
	DeepQueueNet	4	8m5s	3.62-fold
FatTree128	DES	0	20h15m39s	-
	MimicNet	1	11m34s	-
	DeepQueueNet	1	1h6m18s	baseline
	DeepQueueNet	2	33m23s	1.99-fold
	DeepQueueNet	4	17m5s	3.88-fold

- 在单个GPU上，MimicNet在所有FatTree拓扑规模上都优于DeepQueueNet，因为MimicNet针对该拓扑进行了优化，而DeepQueueNet需要进行迭代程序(IRSA)以保证正确性。
- 随着额外GPU数量的增加，DeepQueueNet几乎实现了近似线性的速度提升。

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI