软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(8):3924−3937 [doi: 10.13328/j.cnki.jos.006541]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于流特征的数据中心非对称流负载均衡方法*

陈中卿[1], 李丹丹[1], 闪德胜[2], 钱叶魁[3], 谢 坤[1], 黄小红[1], 丛 群[4]

[1](北京邮电大学 计算机学院 (国家示范性软件学院), 北京 100876)

[2](中国人民解放军 32147 部队, 陕西 宝鸡 721000)

[3](陆军炮兵防空兵学院 (郑州校区), 河南 郑州 450052)

[4](北京网瑞达科技有限公司 技术研发部, 北京 100876)

通信作者: 钱叶魁, E-mail: qyk1129@163.com; 黄小红, E-mail: huangxh@bupt.edu.cn

摘 要: 数据中心边界广泛部署的地址转换技术产生的非对称流为负载均衡系统的设计带来了挑战. 为了解决软件负载均衡系统不能充分发挥多核处理器和网卡硬件能力的问题, 提出一种基于流特征的非对称流负载均衡方法. 首先, 分析网卡的数据包散列机制, 提出数据包调度算法, 将数据包调度至预期的 CPU 核; 然后, 基于会话报文序列的时间与空间特征, 构建大象流识别算法; 最后, 基于识别结果, 提出负载均衡方法. 实验结果表明, 非对称流负载均衡方法可以正确处理非对称流的负载均衡, 平均吞吐率提升约 14.5%.

关键词: 流特征; 负载均衡; 硬件卸载; 多核处理; 网络地址转换

中图法分类号: TP303

中文引用格式: 陈中卿, 李丹丹, 闪德胜, 钱叶魁, 谢坤, 黄小红, 丛群. 基于流特征的数据中心非对称流负载均衡方法. 软件学报, 2023, 34(8): 3924–3937. http://www.jos.org.cn/1000-9825/6541.htm

英文引用格式: Chen ZQ, Li DD, Shan DS, Qian YK, Xie K, Huang XH, Cong Q. Asymmetric Flow Load Balancing Method Based on Flow Characteristics in Data Center Network. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3924–3937 (in Chinese). http://www.jos.org.cn/1000-9825/6541.htm

Asymmetric Flow Load Balancing Method Based on Flow Characteristics in Data Center Network

CHEN Zhong-Qing[1], LI Dan-Dan[1], SHAN De-Sheng[2], QIAN Ye-Kui[3], XIE Kun[1], HUANG Xiao-Hong[1], CONG Qun[4]

[1](School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China)

[2](PLA 32147 Troops, Baoji 721000, China)

[3](PLA Army Academy of Artillery and Air Defense (Zhengzhou Campus), Zhengzhou 450052, China)

[4](R & D Department, Beijing WRD Technology Co. Ltd., Beijing 100876, China)

Abstract: The asymmetric flow generated by the widely deployed address translation technology brings challenges to the design of load balancing system. To solve the problem of insufficient use of multi-core processors and network card hardware capabilities by software load balancers, an asymmetric flow load balancing method based on flow characteristics is proposed. Firstly, a data packet dispatching algorithm to dispatch packets into expected CPU core via hardware is proposed. Then, an elephant flow detection algorithm is constructed by analyzing of the temporal and spatial characteristics of packet sequences. Finally, based on detected results, a load balance offloading method is proposed. The experimental results show that, asymmetric flow load balancing method can correctly handle the asymmetric flow. Meanwhile, the average throughput rate increases by 14.5%.

Key words: flow characteristics; load balance; hardware offloading; multi-core processing; network address translation

* 基金项目: 国家重点研发计划 (2019YFB1802600)
收稿时间: 2021-01-29; 修改时间: 2021-11-06; 采用时间: 2021-11-08; jos 在线出版时间: 2022-09-30
CNKI 网络首发时间: 2022-11-15

1 介 绍

互联网的高速发展对现有的网络构架提出了挑战, 当一个网络组件过载并成为系统瓶颈时, 网络的延迟将大大增加. 为了避免或减轻过载产生的问题, 通常采用负载均衡的方式平衡多个资源间的压力. 高性能, 可扩展的负载均衡系统作为流量的入口, 对业务的可靠性和性能起着决定性的作用[1]. 传统软件负载均衡系统依赖于通用硬件和操作系统, 性能和规模难以提升.

从线程模型上划分, 目前常见的软件负载方法有 3 类.

1) 所有线程共享一张会话表, 每个线程都可以处理任意会话, 数据包被随机发送至各个线程以实现性能的垂直扩展, 如 OVS Conntrack[2] 以及被广泛使用的 LVS (Linux virtual server)[3]. 在多核系统中, 通常需要引入锁机制以避免竞争.

2) 每个线程负责处理一组地址的会话表, 各线程间不共享数据, 独立执行任务, 每个会话的数据包必须由特定的一个线程处理. Eisenbud 等人[4]使用 Tunnel 技术切分线程间的流量, 每个 CPU 核拥有一个独立的隧道地址, 处理不同的流量. André等人[5]提出了在整个系统中, 每个 CPU 核拥有一个独立的 MAC 地址, 核间不共享数据且不使用锁的架构, 有效提升了扩展性和吞吐率.

3) 基于流水线模型处理数据包[6], 每个线程负责数据包处理的不同阶段. 陈福才等人[7]将控制分组和数据分组分配至不同的线程处理, 使用共享内存的方式通信. 李凯等人[8]使用流水线模型, 将线程分为 I/O 线程和工作线程, 各自处理不同的任务, 并利用多重 Hash 算法平衡各个 CPU 核间的压力.

传统软件网络协议的体系结构开销过大, 无法满足万兆以太网对报文处理速度的要求[9]. 随着高速以太网的普及, 网卡集成了越来越多的硬件卸载能力, 以减少 CPU 处理报文的开销. Durner 等人[10]将部分会话处理工作卸载至网卡, 提高了负载均衡系统的吞吐率. Moon 等人[11]提出将部分网络协议栈的逻辑卸载至网卡, 有效提升了网络性能.

虽然上述方法提高了负载均衡系统的吞吐率, 但是依然存在一些问题. 当数据包通过负载均衡系统时, 负载均衡系统会修改数据包的地址和校验和等信息, 经过负载均衡后导致会话出入方向上数据包的地址不对称, 即此会话为非对称会话. 在软件负载均衡系统中, 非对称会话的数据包由不同的 CPU 核完成处理时, 共享会话表会导致多核间发生数据竞争, 降低整体吞吐率, 不能充分发挥多核处理器性能[12]. 同时, 现有的软件负载均衡系统的设计, 难以充分发挥多核处理器的性能以及网卡的硬件能力.

针对上述问题, 结合商用网卡的硬件卸载能力[13], 本文将数据中心中的非对称流处理问题分为硬件卸载和软件调度两方面进行研究, 硬件卸载处理软件难以处理的大象流, 软件调度处理会话量大难以硬件卸载的其他流. 基于以上思路, 本文提出了一种基于流特征的数据中心中非对称流负载均衡方法, 该方法通过分析网卡的数据包散列机制, 设计数据包调度算法, 将数据包调度至预期的 CPU 核, 使得每个 CPU 核独立处理会话的特征提取. 然后在每个 CPU 核独立提取会话数据包的空间和时间特征基础上, 构建大象流识别算法, 最后基于识别结果, 提出负载均衡卸载算法.

本文第 1 节介绍研究背景和主要工作. 第 2 节给出基于流特征的数据中心中非对称流负载均衡方法. 第 3 节通过实验对此方法进行分析与验证. 第 4 节为结束语.

2 基于流特征的数据中心非对称流负载均衡方法

基于流特征的数据中心非对称流负载均衡方法由非对称会话数据包调度算法, 大象流识别算法和负载均衡卸载算法这 3 部分组成. 系统整体框架如图 1 所示.

非对称会话数据包调度算法优化多核环境下的会话数据包的调度, 通过分析网卡的数据包散列机制, 设计数据包调度算法, 将数据包调度至预期的 CPU 核, 实现在每个 CPU 核上独立提取会话的特征.

基于上述设计, 提取一定时间内会话的空间和时间特征, 设计大象流识别算法.

基于识别结果, 给出负载均衡卸载算法, 完成大象流和其他流的负载均衡.

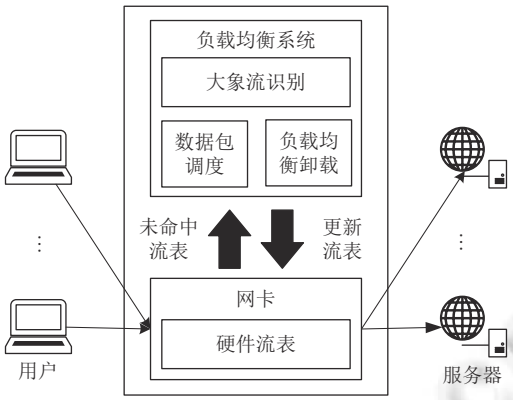


图 1 基于流特征的数据中心非对称流负载均衡方法

2.1 非对称会话数据包调度

RSS (receive side scaling) 是网卡调度数据包的一种技术, 它通过 Toeplitz Hash 算法计算数据包的散列值将数据包调度至不同的网卡队列, 实现了多核处理网卡数据包[14]. 然而, 传统的 RSS 算法为了尽可能平衡多队列之间的压力, 选用了随机矩阵, 产生了非对称会话.

如图 2 所示, 其中 $\boldsymbol{T}_u$ 为用户侧网卡上的 Toeplitz 矩阵, $\boldsymbol{T}_s$ 为服务侧网卡上的 Toeplitz 矩阵, $\boldsymbol{T}_u$ 和 $\boldsymbol{T}_s$ 是 n 维的对称 Toeplitz 方阵, 其维度 n 由网卡硬件决定, $\boldsymbol{T}_u$ 和 $\boldsymbol{T}_s$ 由列向量 $\boldsymbol{K}_u$ 和 $\boldsymbol{K}_s$ 生成. 在处理非对称会话时, 来自用户方向 (实线路径) 和来自服务方向 (虚线路径) 的数据包被网卡调度至不同的接收队列. 由于传统 RSS 算法选用了随机的 $\boldsymbol{T}_u$ 和 $\boldsymbol{T}_s$, 数据包将被随机发送至不同的 CPU 核处理, 相同会话的两个方向流的处理路径非对称. 在负载均衡系统更新会话状态、调整会话策略时, 需要频繁地访问这两个不同的流, 这会导致处理器的不同核间在并发地访问同一个会话的信息时发生数据竞争. 此外, 非对称会话导致不同 CPU 核的会话数量不平衡, 进而导致不同 CPU 核的缓存利用率不均匀, 进而使得不同 CPU 核的处理能力不均衡. 因此本文选择均衡各 CPU 核间的会话数. 为了避免数据竞争, 均衡各 CPU 核间的会话数, 本文提出非对称会话数据包调度算法.

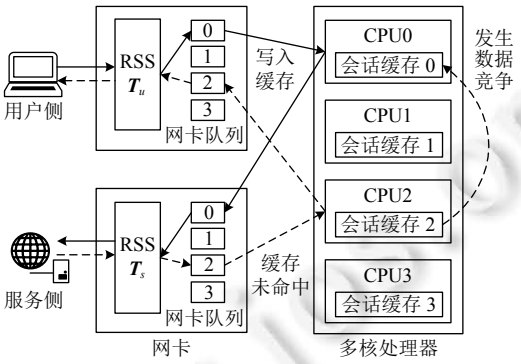


图 2 非对称会话数据包的传统处理路径

若一个会话从用户侧进入, 其地址四元组 $\boldsymbol{p}_u$ 为 $[SIP_u, DIP_u, SPort_u, DPort_u]$(在 IPv4 下对应 96 维行向量) 经过负载均衡系统后, 该四元组被映射为一个至服务侧的新四元组 $\boldsymbol{p}_s$ 为 $[SIP_s, DIP_s, SPort_s, DPort_s]$(在 IPv4 下对应 96 维行向量). 根据 Toeplitz Hash 算法, 用户侧数据包的哈希值 H_u 为:

$$H_u = \boldsymbol{T}_u \cdot \boldsymbol{p}_u^{\mathrm{T}} \tag{1}$$

服务侧数据包的哈希值 H_s 为:

$$H_s = \boldsymbol{T}_s \cdot \boldsymbol{p}_s^{\mathrm{T}} \tag{2}$$

为了使非对称会话出入方向的数据包被调度至相同的队列, 需要满足:

$$H_u \equiv H_s \ (\mathrm{mod}\ Q) \tag{3}$$

其中, Q 为启用的网卡队列总数, 通常与 CPU 核数一致.

在负载均衡系统中, 需要维护用户表, 记录不同源 IP 的会话和流量信息[15]. 如果一个用户的会话分散在不同的 CPU 核, 不利于实现基于用户的统计和限流. 因此在用户侧, 选择基于源 IP 地址进行数据包调度. 在 IPv4 环境下, 地址长度为 32 位, 可以通过设置 $\boldsymbol{K}_u$ 的 0–31 位为一个随机值以满足上述要求:

$$\begin{cases} \boldsymbol{K}_u[i] \in \{0,1\},\ i \leqslant 31 \\ \boldsymbol{K}_u[i] = 0,\ i > 31 \end{cases} \tag{4}$$

在服务侧, 需要将数据包送至和用户侧相同的网卡硬件队列. 为了减少计算复杂度, 根据 Toeplitz 的性质[16], 通过构造下列矩阵, 使得 Toeplitz Hash 仅取决于端口号 $SPort_s$:

$$\begin{cases} \boldsymbol{K}_s[i] = 1,\ i = m \\ \boldsymbol{K}_s[i] = 0,\ i \neq m \end{cases} \tag{5}$$

其中, m 表示行向量 $\boldsymbol{p}_u$ 中分量 $SPort_s$ 的起始位置. 设 B_j 为数据包四元组 $\boldsymbol{p}_u$ 对应的二进制序列的第 j 个元素, H_k 为第 k 轮迭代的 Toeplitz Hash 值, 根据 Toeplitz Hash 算法有:

$$\begin{cases} H_k = 0,\ k < m-31 \\ H_k = H_{k-1},\ k > m+1 \end{cases} \tag{6}$$

当 $0 \leqslant k < 32$ 时有:

$$H_{m-k+1} = H_{m-k} \oplus \left(B_{m-k} \wedge \sum_{i=1}^{32} 2^{i-1} \times K_{m-k+32-i} \right) \tag{7}$$

根据公式 (5)–公式 (7) 可以得到, 服务侧的 Toeplitz Hash 值 H_s 为:

$$H_s = \sum_{i=1}^{32} 2^{i-1} \times B_{m-32+i} \tag{8}$$

因此, 根据公式 (3) 和公式 (8) 可知在网卡队列数为 Q 时, 负载均衡创建会话时选择转换后的源端口号 $SPort_s$ 需要满足:

$$ReverseBits(SPort_s) \equiv H_u \ (\mathrm{mod}\ Q) \tag{9}$$

其中, *ReverseBits* 为二进制逆序函数.

根据公式 (9) 可知, 要避免负载均衡中的数据包的处理路径非对称问题, 需要选择合适的 $SPort_s$ 使得用户侧和服务侧的数据包进入相同的队列. 非对称会话数据包调度算法首先根据公式 (4) 和公式 (5) 初始化用户侧和服务侧网卡参数, 然后根据输入的参数生成可能的 $SPort_s$, 并根据公式 (9) 验证其正确性, 最后输出满足条件的 $SPort_s$, 如果未能找到合适的 $SPort_s$ 则输出 0. 非对称会话数据包调度算法如算法 1.

算法 1. 非对称会话数据包调度算法.

输入: H_u, Q;

输出: $SPort_s$.

1. 根据公式 (4) 和公式 (5) 初始化用户侧和服务侧网卡参数;
2. LET $rss \leftarrow H_u$;
3. LET mask ← length(IndirectionTable)–1;
4. REPEAT
5. $SPort_s \leftarrow ReverseBits(rss)$;
6. IF $SPort_s$ 满足公式 (9) THEN
7. CheckPortStatus($SPort_s$);
8. IF $SPort_s$ is available THEN

```
9.                SetUsedPort(SPort_s);
10.               RETURN;
11.           END IF
12.       END IF
13.       rss ← (rss+Q) & 0xffff;
14. UNTIL (rss–H_u) > 0xffff;
15. SPort_s ←0;
```

2.2 大象流识别

在网络中, 根据流量特征, 会话主要分为大象流, 老鼠流和其他流. 其中, 大象流在会话中的占比约为 0.1%, 但其数据量占总数据量的 20%, 并可能对老鼠流形成线头阻塞[17]. 另一方面, 大象流的建立和销毁的频率较低, 转发所消耗 CPU 资源多. 网卡的硬件卸载能力可以减少 CPU 处理报文的开销, 然而网卡等硬件提供的流表数量有限, 无法卸载全量会话. 为了有效地利用有限的硬件流表资源, 降低 CPU 处理数据包的开销, 提出大象流识别算法.

本文基于公开的网络流量数据集 CSE-CIC-IDS2018 on AWS[18]作为训练集, 对数据集中的会话特征进行统计分析.

为了实时且准确地识别大象流, 首先对数据集中会话的持续时间进行分析. 如图 3 所示, 仅有 0.5% 会话的持续时间小于 120 s, 有 32% 的流的会话时间大于 1 200 s. 经过统计, 在会话开始 60 s 内进行大象流识别可以覆盖超过 99.7% 的会话. 因此, 本文将会话的处理时间分为识别窗口 W_p 和转发窗口 W_f. 识别窗口用于计算会话特征, 识别会话类别, 为保证识别的实时性和覆盖率, 识别窗口应小于 60 s. 转发窗口仅转发会话报文, 不更新会话特征.

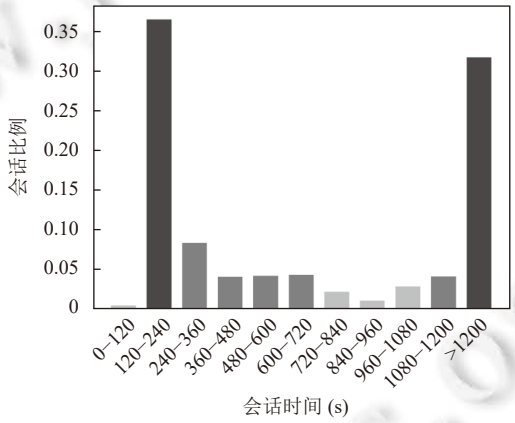


图 3 会话持续时间分布图

在识别窗口内, 负载均衡系统统计会话报文的单向最大平均包大小 S_{avg}, 会话已存在时间 T_{sess} 和最大单向包速率 $R_{\max}$ 这 3 种特征, 将其量化为特征向量 $\left[S_{\text{avg}}, T_{\text{sess}}, R_{\max}\right]$.

会话报文平均包大小 S_{avg} 是识别窗口内单个会话出方向和入方向的平均包大小的最大值, 反映了该会话的空间特征.

$$S_{\text{avg}} = \max\left\{\frac{1}{n_{\text{input}}}\sum_{i=0}^{n_{\text{input}}} S_{\text{input}}(i), \frac{1}{S_{\text{input}}(i)}\sum_{i=0}^{n_{\text{output}}} S_{\text{output}}(i)\right\} \tag{10}$$

其中, S_{input} 和 S_{output} 表示该会话入方向和出方向的报文大小序列.

会话生存时间 T_{sess} 表示在识别窗口内, 该会话最后一次收到报文的时间和最后一次发送报文的时间, 距会话开始的时间之差的最大值, 反映了会话的时间特征.

$$T_{\text{sess}} = \max\left\{t_{\text{last\_in}} - t_0, t_{\text{last\_out}} - t_0\right\} \tag{11}$$

其中, t_0 表示会话开始的时间, $t_{\text{last\_in}}$ 表示该会话最后接收报文的时间, $t_{\text{last\_out}}$ 表示该会话最后发送报文的时间.

最大单向包速率 $R_{\max}$ 是识别窗口时间内会话单方向上的最大速率, 反映了会话的平均速率特征.

$$R_{\max} = \max\left\{\frac{\sum_{i=0}^{n_{\text{input}}} S_{\text{input}}(i)}{t_{\text{last\_in}} - t_0}, \frac{\sum_{i=0}^{n_{\text{output}}} S_{\text{output}}(i)}{t_{\text{last\_out}} - t_0}\right\} \tag{12}$$

将特征向量 $\left[S_{\text{avg}}, T_{\text{sess}}, R_{\max}\right]$ 作为聚类分析的输入. 聚类的目的是将训练数据集划分为不同的簇, 从而得到带标注的特征向量集合. 本文选择 BIRCH 作为聚类模型, 它基于特征树 CF 树, 运行速度快, 适合于数据量大情况. 则对于有 N 个数据的簇, 其聚类特征为三元组 $\overrightarrow{CF} = \left\langle N, \vec{\Sigma}_l, \Sigma_s \right\rangle$, 其中矢量特征 $\vec{\Sigma}_l$ 为:

$$\vec{\Sigma}_l = \left(\sum_{n=1}^{N} S_{\text{avg}}(n), \sum_{n=1}^{N} T_{\text{sess}}(n), \sum_{n=1}^{N} R_{\max}(n)\right)^{\text{T}} \tag{13}$$

标量特征 Σ_s 为:

$$\Sigma_s = \sum_{n=1}^{N}\left(S_{\text{avg}}(n)^2 + T_{\text{sess}}(n)^2 + R_{\max}(n)^2\right) \tag{14}$$

通过构建 CF 树, 以 CF 树叶元项对应的子簇为基础, 对数据点进行聚类得到分类结果 C, 并更新特征向量为 $\left[S_{\text{avg}}, T_{\text{sess}}, R_{\max}, C\right]$.

大象流在所有会话中比例小于其他类别, 为了避免出现过拟合现象, 有必要对训练集进行数据增强. 经过统计, 会话特征参数满足正态分布, 本文选用高斯噪声进行数据增强, 设 ξ_1, ξ_2 为 [0, 1] 内的一组均匀分布的伪随机数, 根据 Box-Muller 变换[19]有:

$$N = \sqrt{-2\ln(1-\xi_1)}\cos 2\pi\xi_2 \tag{15}$$

其中, N 为满足高斯分布的随机变量, 对大象流簇中的特征向量 $\left[S_{\text{avg}}, T_{\text{sess}}, R_{\max}, C\right]$ 加入高斯噪声产生一组新的特征向量加入训练集.

使用数据增强后的训练集训练机器学习算法, 得到识别模型. 考虑到大象流的识别是二分类问题且执行速度要求高, 本文选择 CART 决策树算法作为大象流的识别算法.

基于分类结果 C 将训练数据集划分为非大象流簇样本集 $\left\{\left(\vec{x_1}, y_1\right)\right\}$ 和大象流簇样本集 $\left\{\left(\vec{x_2}, y_2\right)\right\}$. 将训练集 $D = \left\{\left(\vec{x_1}, y_1\right), \left(\vec{x_2}, y_2\right)\right\}$ 和属性集 $A = \left\{S_{\text{avg}}, T_{\text{sess}}, R_{\max}\right\}$ 作为输入, 使用 *Gini* 系数来选择划分属性, *Gini* 系数反映了从样本集 D 中随机抽取两个样本, 其类别标记不一致的概率. 在二分类情况下 *Gini* 系数定义如下:

$$Gini(D) = 2\frac{|C_1|}{|D|}\left(1 - \frac{|C_1|}{|D|}\right) \tag{16}$$

其中, C_1 是大象流样本集. 根据特征 A, 把 D 分成 D_1 和 D_2, 则在特征 A 下样本 D 的 *Gini* 系数表示为:

$$Gini(D, A) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2) \tag{17}$$

基于 *Gini* 指数挑选划分属性, 递归选择最优划分属性对数据集划分, 构造决策树, 即获得了一个从属性值到流量类型分类的映射关系 $\mathcal{F}: X \to Y$, 其中, X 表示训练样本集, $Y = \{0, 1\}$ 表示分类标记, 1 代表大象流, 0 代表其他流.

大象流识别算法首先根据 $\boldsymbol{p}_u$ 查找会话表得到会话信息, 然后根据会话已存在时间判断会话是否在识别窗口内, 如果是则更新会话的统计信息, 否则根据识别模型计算会话特征, 最后将会话特征输入分类模型得到会话的分类结果. 大象流识别算法如算法 2.

算法 2. 大象流识别算法.

输入: $\boldsymbol{p}_u$, S_{input}, S_{output}, t_0, $t_{\text{last\_in}}$, $t_{\text{last\_out}}$, W_p;
输出: 会话分类标记 Y.

1. LET session←LookupSession($\boldsymbol{p}_u$);
2. IF max($t_{\text{last\_in}}$, $t_{\text{last\_out}}$) > W_p THEN
3. 根据公式 (10)–公式 (12) 计算会话特征 S_{avg}, T_{sess}, $R_{\max}$;
4. LET Y ← $\mathcal{F}\left(\left[S_{\text{avg}}, T_{\text{sess}}, R_{\max}\right]\right)$;
5. ELSE IF IsInput(session) THEN
6. S_{input} ← Append(S_{input}, GetPacketSize(session));
7. $t_{\text{last\_in}}$ ← Time();
8. ELSE
9. S_{output} ← Append(S_{output}, GetPacketSize (session));
10. $t_{\text{last\_out}}$ ← Time();
11. END IF

2.3 负载均衡卸载

为了尽可能地均衡不同 CPU 核间的压力, 本文考虑流量大小和会话数量等特征进行流量识别, 并对识别的大象流和非大象流设计不同的负载均衡方案. 根据非对称会话数据包调度算法和大象流识别算法, 本文将负载均衡中的报文处理路径分为快速路径和慢速路径[20]. 快速路径负责处理易于硬件卸载的工作, 包括会话数据包的解析修改与转发. 慢速路径处理会话的新建和大象流识别. 不同特征的流, 其处理路径有所不同. 大象流在卸载前由慢速路径处理, 在卸载后由快速路径处理. 老鼠流和其他流仅由慢速路径进行处理.

快速路径由网卡进行硬件卸载, 当数据包报文到达网卡并命中网卡流表时, 网卡在报文中加入元数据并送入指定的 CPU 核, 然后根据元数据查找得到对应的会话信息, 最后修改并转发该数据包. 慢速路径由 CPU 执行数据中心中非对称会话负载均衡卸载算法, 控制会话是否进行硬件卸载.

数据中心中非对称会话负载均衡卸载算法首先根据 $\boldsymbol{p}_u$ 进行会话匹配, 若未匹配成功, 则根据非对称会话数据包调度算法选择合适的源端口号完成会话的新建, 然后根据大象流识别算法识别会话类型, 最后将大象流会话的信息写入硬件流表, 完成硬件卸载. 数据中心中非对称会话负载均衡卸载算法的输出为会话的硬件卸载标志, 如果会话未能进行硬件卸载, 则该会话将由 CPU 完成软件转发. 数据中心中非对称的负载均衡卸载算法如算法 3.

算法 3. 数据中心中非对称会话负载均衡卸载算法.

输入: $\boldsymbol{p}_u$, $\boldsymbol{p}_s$;
输出: 会话卸载标志 offload.

1. LET session←LookupSession($\boldsymbol{p}_u$);
2. IF session not exist THEN
3. 根据算法 1 计算 $SPort_s$;
4. UpdateSessionPort($\boldsymbol{p}_s$, $SPort_s$);
5. END IF
6. IF Offloading(session) THEN
7. offload←true;
8. RETURN;
9. END IF
10. 根据算法 2 计算 Y;
11. IF Y=1 THEN
12. offload←OffloadSession(session);
13. END IF

3 实验与分析

本节对基于流特征的数据中心非对称流负载均衡方法的负载均衡效果和吞吐率进行对比实验和分析.

3.1 实验环境

用于实验验证本文所提出的基于流特征的数据中心非对称流负载均衡方法的测试环境拓扑如图 4 所示, 所使用的主机配置如表 1 所示.

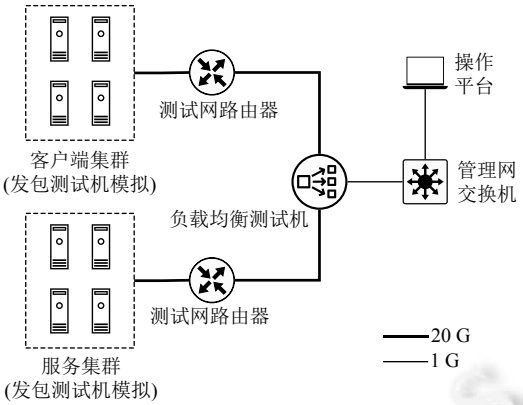


图 4 实验环境网络拓扑图

表 1 实验环境

项目	配置
OS	CentOS release 7.6 (Core)
CPU	Intel (R) Xeon (R) CPU E3-1230 V5
内存	16 GB (2 × 8 GB DDR4 2133 MHz)
网卡	Intel X710 for 10 GbE SFP+ (rev 02)

测试软件使用 Cisco TRex 作为数据包发生器, 运行在与被测主机同等硬件规格硬件上. 被测主机关闭了超线程, 睿频和节能模式. 测试环境网络拓扑图如图 4 所示, 由发包测试机、负载均衡测试机、测试网路由器和管理网交换机组成. 使用发包测试机模拟了客户端集群和服务端集群. 发包测试机、负载均衡测试机和测试网路由器间使用 4 组 10 G SFP+铜缆直连. 发包测试机、负载均衡测试机与管理端通过管理交换机连接, 用于模拟参数的调整和监控.

3.2 非对称会话数据包调度

在这个实验中, 对本文的非对称会话数据包调度算法的调度效果进行验证. 本节将对比网卡原始的数据包调度方法 (基于随机矩阵的 RSS 调度算法) 和本文算法 (基于非对称 RSS 的调度算法). 实验随机构造了 10–10^7 个非对称会话, 测试在 4 个队列情况下不同算法下各 CPU 核间的负载均衡程度, 衡量指标有会话本地率和负载偏差值.

会话本地率:

$$L = \frac{S_{\text{Local}}}{S_{\text{Total}}} \tag{18}$$

会话本地率衡量会话创建时所在的 CPU 核与该会话后续报文处理时所在的 CPU 核相同的概率, 其中 S_{Local} 代表会话报文始终在一个 CPU 核的会话数量, S_{Total} 代表总的会话数量.

负载偏差值:

$$D = \frac{\sqrt{\dfrac{1}{N}\sum_{i=1}^{N}(S_i - \overline{S})}}{\overline{S}} \tag{19}$$

负载偏差值衡量调度到各个 CPU 核的会话量的偏差值, 其中 N 代表总会话数量, S_i 表示调度至第 i 个核的会话数量. 负载偏差值越低说明各个 CPU 核的负载越均匀.

由图 5 可知, 在非对称会话环境下, 标准 RSS 算法的会话本地率在 0.2–0.25 之间. 本文算法在不同会话数下, 会话本地率均能达到 1.0, 实现了相同会话的数据包由同一 CPU 核完成处理.

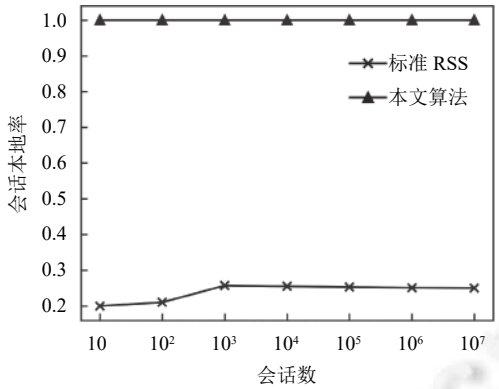


图 5　不同会话数下的会话本地率

由图 6 所示, 在会话数小于 10^3 时, 本文算法的负载偏差值大于标准 RSS 算法, 负载偏差值和标准 RSS 算法差距小于 0.2, 不同 CPU 核间的会话数不均衡. 在实际网络环境中, 典型的会话数通常在 10^4 以上, 此时本文算法和标准 RSS 算法的负载偏差值接近, 会话负载均匀分布在各个 CPU 核间. 因此本文算法带来的会话数较少时负载不均匀的现象可以接受.

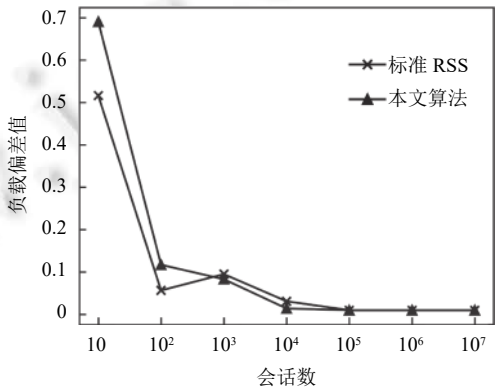


图 6　不同会话数下的负载偏差值

其次, 比较不同会话数下调度非对称流数据包至相同 CPU 核和不同 CPU 核的负载均衡吞吐率和 LLC (last level cache) 命中率, 实验结果如图 7 所示.

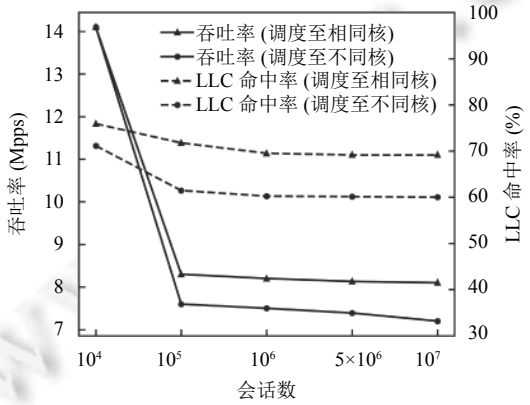


图 7　不同会话数下的吞吐率和 LLC 命中率

实验结果表明: 与随机调度至不同的 CPU 核相比, 把非对称流的数据包调度至相同的 CPU 核, 其负载均衡吞吐率和 LLC 命中率表现更好. 在会话数为 10^4 时两种调度方法的吞吐率都达到了网卡极限, LLC 命中率相近. 在会话数大于 10^5 时, 把非对称流的数据包调度至相同的 CPU 核的吞吐率平均提升了 11.7%, LLC 命中率平均提升了 14.9%. 此时, 吞吐率和 LLC 命中率趋于稳定, 这是因为会话所占的内存远大于 LLC 的大小, 访存的延迟和速率成为了系统的瓶颈.

3.3 大象流识别结果分析

在这个实验中, 基于公开的 CSE-CIC-IDS2018 on AWS 数据集, 将其按照 3:1 的比例随机划分为训练集和测试集. 使用本文的训练算法提取训练集中的会话特征, 并对比不同决策树算法和检测窗口大小下测试集中流量的大象流识别结果. 使用混淆矩阵来评价分类模型的表现效果, 如表 2 所示, 矩阵的每一行表示样本的真实情况, 每一列表示模型识别的样本情况:

表 2 混淆矩阵

	模型识别为正类	模型识别为负类
真实类别为正类	*TP*	*FN*
真实类别为负类	*FP*	*TN*

评价指标选用精确率和召回率.

精确率:

$$Precision = \frac{TP}{TP+FP} \tag{20}$$

召回率:

$$Recall = \frac{TP}{TP+FN} \tag{21}$$

精确率衡量划分到某个类别中的会话中属于此类别的比例, 精确率越高说明分类器分类越准确. 召回率衡量实际属于某个类别的所有会话中被正确划分到该类别的比例, 召回率越高说明分类器在该类上漏掉的会话越少.

下面给出在不同识别窗口下, 不同决策树算法的精确率和召回率.

从图 8 可知, 在大象流识别的场景中, 选用不同的识别窗口大小, 结果的召回率差异显著. 在识别窗口小于 20 s 时召回率低于 60%, 精确率小于 90%. 在识别窗口大于 30 s 时召回率大于 80%, 精确率大于 90%. 综合考虑识别的准确度和实时性, 当窗口大小设置为 30 s, 决策树生长算法使用 CART 时, 可以得到较好的表现.

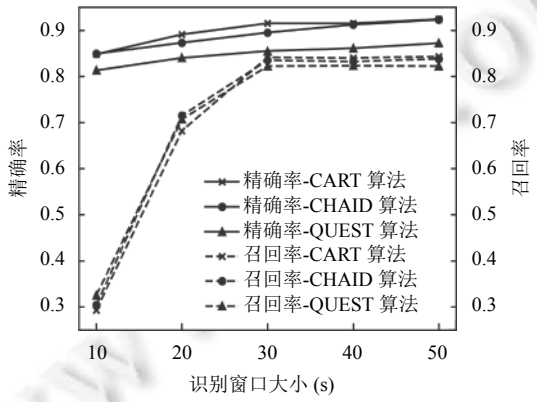


图 8 不同识别窗口下的精确率和召回率

下面, 对比本文算法与 Sample[21]、Multistage[21]和 Hits[22]算法的大象流识别效果, 测试结果如表 3 所示.

从表 3 可知, 本文算法在识别窗口为 30 s 和 40 s 时, 识别大象流的流量占比均为 0.84, 在窗口为 50 s 时, 识别

大象流的流量占比为 0.85, 识别窗口大小对识别结果的影响较小. 而 Hits, Sample 和 Multistage 识别大象流的流量占比随着阈值的改变, 波动较大. 与这些大象流识别算法相比, 本文算法的阈值设置 (识别窗口大小) 对识别结果的影响较小同时又能实现较好的识别效果.

表 3 不同识别算法比较

方法	参数	识别数量	流量占比
本文算法	窗口=30 s	1 298	0.84
	窗口=40 s	1 298	0.84
	窗口=50 s	1 307	0.85
Hits[22]	阈值=0.3%	1 199	0.81
	阈值=0.4%	1 142	0.79
	阈值=0.6%	1 140	0.79
Multistage[21]	阈值=0.3%	971	0.73
	阈值=0.4%	946	0.60
	阈值=0.6%	919	0.58
Sample[21]	阈值=0.3%	939	0.62
	阈值=0.4%	919	0.77
	阈值=0.6%	892	0.76

3.4 吞吐率

VPP 是 Cisco 开源的成熟的高性能包处理框架. VPP 的负载均衡模块使用了 Handoff 方式完成会话数据包的调度, 本小结将对比 Handoff 方法 (即基于软件的会话数据包调度) 和本文算法 (数据中心中非对称流负载均衡卸载算法) 在不同会话数下的数据包处理吞吐率.

实验使用 Cisco TRex 发包软件产生 10^4–10^7 个的会话流量, 发送至负载均衡设备, 测试系统的吞吐率. 根据真实环境下的大象流比例和流量特征, 模拟会话中大象流的占比在 0.05%–0.1% 之间, 会话数据包的平均包大小为 220 字节, 包大小的分布如图 9.

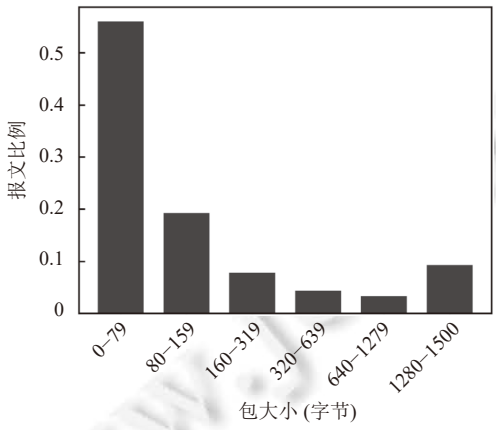


图 9 包大小分布

下面, 展示不同会话数下的大象流识别的精确率. 随机产生不同数量的会话, 测试不同会话数下大象流识别算法的精确率, 测试结果如表 4 所示.

从表 4 可知, 随着会话数量和大象流比例的增加, 识别精确率也在增大. 在实际网络环境中, 典型的会话数通常在 10^4 以上, 大象流比例约为 0.1%, 此时大象流识别精确率大于 85%.

表 4 不同会话数下的大象流识别精确率

会话数	0.05%大象流	0.1%大象流
10^4	0.84	0.85
10^5	0.87	0.88
10^6	0.87	0.87
5×10^6	0.88	0.90
10^7	0.90	0.91

其次, 比较不同会话数下的负载均衡吞吐率. VPP 的负载均衡模块没有大象流识别和硬件卸载功能, 因此 VPP 的测试结果为不同大象流比例下吞吐率的平均值.

从图 10 可知, 本文算法较 VPP 的 Handoff 算法有了较大的提升. 在会话数为 10^4 时两种算法的吞吐率都达到了网卡极限, 在会话数大于 10^5 时, 吞吐率较 VPP 平均提升了 14.5%.

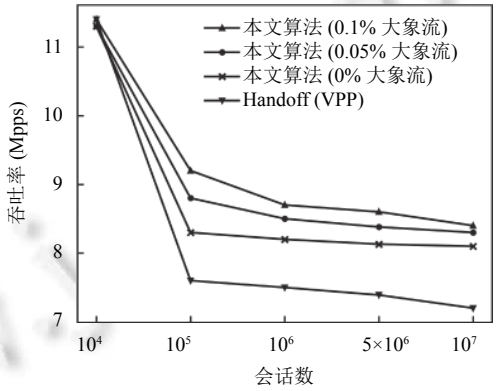


图 10 不同会话数下的吞吐率

通过实验可以看出, 本文算法在实现负载均衡的前提下, 提高了处理大量会话下的吞吐率, 且在不同会话数和不同大象流比例下都有较好的表现.

4 结束语

本文提出了一种基于流特征的数据中心非对称流负载均衡方法. 首先, 通过分析网卡的数据包散列机制, 提出了非对称数据包调度算法, 将数据包调度至预期的 CPU 核, 避免了 CPU 核直接发生数据竞争. 然后, 根据网络流量的特点, 提取了会话报文序列的时间与空间特征, 构建了大象流识别模型, 提出了大象流识别算法. 最后, 基于识别算法, 提出了负载均衡卸载算法, 将报文的处理路径分为了慢速路径和快速路径, 实现了利用有限的网卡的硬件能力卸载 CPU 的部分处理工作. 实验表明, 本文算法可正确处理数据中心中非对称流的负载均衡, 与现有的软件负载均衡方法相比较, 平均吞吐率提升约 14.5%, 充分发挥了多核处理器和网卡硬件能力.

References:

[1] Silva WJA, Dias KL, Sadok DFH. A performance evaluation of software defined networking load balancers implementations. In: Proc. of the 2017 Int'l Conf. on Information Networking (ICOIN). Da Nang: IEEE, 2017. 132–137. [doi: 10.1109/ICOIN.2017.7899491]

[2] Jackson EJ, Walls M, Panda A, Pettit J, Pfaff B, Rajahalme J, Koponen T, Shenker S. Softflow: A middlebox architecture for open vswitch. In: Proc. of the 2016 USENIX Annual Technical Conf. Denver: USENIX Association, 2016. 15–28.

[3] Zhang WS. Linux virtual server for scalable network services. In: Proc. of the 2000 Ottawa Linux Symp. Ottawa, 2000. 1–10.

[4] Eisenbud DE, Yi C, Contavalli C, Smith C, Kononov R, Man-Hielscher E, Cilingiroglu A, Cheyney B, Shang WT, Hosein J E. Maglev: A fast and reliable software network load balancer. In: Proc. of the 13th Symp. on Networked Systems Design and Implementation.

Lombard: USENIX, 2016. 523–535.

[5] André F, Gouache S, Le Scouarnec N, Monsifrot A. Don't share, don't lock: Large-scale software connection tracking with krononat. In: Proc. of the 2018 USENIX Annual Technical Conf. Boston: USENIX Association, 2018. 453–465.

[6] Jin MX, Wang CS, Li P, Han ZJ. Survey of load balancing method based on DPDK. In: Proc. of the 4th IEEE Int'l Conf. on Big Data Security on Cloud (BigDataSecurity), the IEEE Int'l Conf. on High Performance and Smart Computing (HPSC) and the IEEE Int'l Conf. on Intelligent Data and Security (IDS). Omaha: IEEE, 2018. 222–224. [doi: 10.1109/BDS/HPSC/IDS18.2018.00054]

[7] Chen FC, He WZ, Cheng GZ, Huo SM, Zhou DC. Design of key technologies for intranet dynamic gateway based on DPDK. Journal on Communications, 2020, 41(6): 139–151 (in Chinese with English abstract).

[8] Li K, Ye, L, Yu XZ, Hu Y. Traffic dynamic load balancing method based on DPDK. Intelligent Computer and Applications, 2017, 7(4): 85–89, 91 (in Chinese with English abstract).

[9] Gallenmüller S, Emmerich P, Wohlfart F, Raumer D, Carle G. Comparison of frameworks for high-performance packet IO. In: Proc. of the 2015 ACM/IEEE Symp. on Architectures for Networking and Communications Systems. Oakland: IEEE, 2015. 29–38. [doi: 10.1109/ANCS.2015.7110118]

[10] Durner R, Varasteh A, Stephan M, Machuca CM, Kellerer W. HNLB: Utilizing hardware matching capabilities of NICS for offloading stateful load balancers. In: Proc. of the 2019 IEEE Int'l Conf. on Communications (ICC). Shanghai: IEEE, 2019. 1–7. [doi: 10.1109/ICC.2019.8761434]

[11] Moon YG, Lee SE, Jamshed MA, Park K. AccelTCP: Accelerating network applications with stateful TCP offloading. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 77–92.

[12] Wu HS, Wang CJ, Xie JY. A lock-free multi-processing session persistence mechanism for load balancing in multi-core environment. Journal of Electronics and Information Technology, 2013, 35(4): 982–987 (in Chinese with English abstract).

[13] Zheng P, Narayanan A, Zhang ZL. A closer look at NFV execution models. In: Proc. of the 3rd Asia-Pacific Workshop on Networking 2019. New York: ACM, 2019. 85–91. [doi: 10.1145/3343180.3343188]

[14] Zhang Y, Wu HS. Comparison and analysis of Hash algorithm for multi-process load balancing. Computer Engineering, 2014, 40(9): 71–76 (in Chinese with English abstract).

[15] He KQ, Hu CC, Jiang JC, Zhou YC, Liu B. A2C: Anti-attack counters for traffic measurement. In: Proc. of the 2010 IEEE Global Telecommunications Conf. Miami: IEEE, 2010. 1–5. [doi: 10.1109/GLOCOM.2010.5684142]

[16] Woo S, Park KS. Scalable TCP session monitoring with symmetric receive-side scaling. Technical Report, Daejeon: KAIST, 2012. 163–169.

[17] Liu JL, Huang JW, Jiang WC, Wang JX. Survey on load balancing mechanism in data center. Ruan Jian Xue Bao/Journal of Software, 2021, 32(2): 300–326 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6151.htm [doi: 10.13328/j.cnki.jos.006151]

[18] Canadian Institute of Cybersecurity. CSE-CIC-IDS2018 on AWS. https://www.unb.ca/cic/datasets/ids-2018.html (2018-10-08)[2020-08-13].

[19] Scott DW. Box-muller transformation. WIREs Computational Statistics, 2011, 3(2): 177–179. [doi: 10.1002/wics.148]

[20] van Tu N, Yoo JH, Hong JWK. Accelerating virtual network functions with fast-slow path architecture using eXpress data Path. IEEE Trans. on Network and Service Management, 2020, 17(3): 1474–1486. [doi: 10.1109/TNSM.2020.3000255]

[21] Estan C, Varghese G. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Trans. on Computer Systems (TOCS), 2003, 21(3): 270–313. [doi: 10.1145/859716.859719]

[22] Wang H, Gong ZH. Hits and Holds: Two algorithms for identifying the elephant flows. Ruan Jian Xue Bao/Journal of Software, 2010, 21(6): 1391–1403 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3522.htm [doi: 10.3724/SP.J.1001.2010.03522]

附中文参考文献:

[7] 陈福才, 何威振, 程国振, 霍树民, 周大成. 基于DPDK的内网动态网关关键技术设计. 通信学报, 2020, 41(6): 139–151.

[8] 李凯, 叶麟, 余翔湛, 胡阳. 基于DPDK的流量动态负载均衡方法. 智能计算机与应用, 2017, 7(4): 85–89, 91.

[12] 吴和生, 王崇骏, 谢俊元. 一种多核环境中无锁的多进程负载均衡会话保持方案. 电子与信息学报, 2013, 35(4): 982–987.

[14] 张莹, 吴和生. 面向多进程负载均衡的Hash算法比较与分析. 计算机工程, 2014, 40(9): 71–76.

[17] 刘敬玲, 黄家玮, 蒋万春, 王建新. 数据中心负载均衡方法研究综述. 软件学报, 2021, 32(2): 300–326. http://www.jos.org.cn/1000-9825/6151.htm [doi: 10.13328/j.cnki.jos.006151]

[22] 王宏, 龚正虎. Hits和Holds: 识别大象流的两种算法. 软件学报, 2010, 21(6): 1391–1403. http://www.jos.org.cn/1000-9825/3522.htm [doi: 10.3724/SP.J.1001.2010.03522]

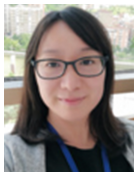
陈中卿(1996－), 男, 硕士, 主要研究领域为软件定义网络, 下一代互联网.

李丹丹(1987－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为网络安全, 密码学.

闪德胜(1963－), 男, 高级工程师, 主要研究领域为网络测量, 网络安全.

钱叶魁(1980－), 男, 博士, 教授, 主要研究领域为网络测量, 网络安全.

谢坤(1984－), 男, 博士, 讲师, 主要研究领域为下一代网络, 网络资源规划与性能优化, 智能路由算法.



黄小红(1979－), 女, 博士, 教授, CCF 专业会员, 主要研究领域为计算机网络应用, 下一代互联网, 网络安全.

丛群(1980－), 男, 硕士, 主要研究领域为网络管理.