Software Engineering and Applications 软件工程与应用, 2025, 14(4), 842-854
Published Online August 2025 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2025.144074

Hans汉斯

基于双向流统一标识与隐式状态机的TCP流追踪算法研究

张文诚，李　超，毕修瑜，段君毅，陈柳廷

中国人民解放军63892部队，河南　洛阳

收稿日期：2025年7月4日；录用日期：2025年8月5日；发布日期：2025年8月14日

摘　要

随着网络流量的爆发式增长和复杂性的不断提升，高效、准确地对TCP流进行追踪和重组成为网络流量分析中的关键任务。现有技术在网络流量分析中存在明显不足，包括多字节编码解析能力有限和流追踪效率低下的问题。为此，本文提出了一种基于双向流统一标识与隐式状态机的TCP流追踪算法。该算法通过设计双向无关性流表管理机制，解决了传统五元组流表管理方法中双向流冗余的问题，将流表空间利用率从50%提升至100%；同时，利用隐式状态机实现跨包分片的UTF-8字符边界精确检测，解析准确率达99.8%。实验结果表明，该算法在处理10 Gbps流量时，CPU利用率显著降低，且在解析准确率和处理效率方面均优于现有工具。然而，该算法在面对具有高度动态性和不确定性的复杂网络环境时，其稳定性和可靠性还有待进一步提高，尤其是在面对大规模、高并发的网络流量时，算法的扩展性和适应性仍需进一步优化。

关键词

TCP流追踪，双向流统一标识，隐式状态机，流表管理，多字节编码解析

A TCP Flow Tracking Algorithm Based on Bidirectional Flow Unified Identification and Implicit State Machines

Wencheng Zhang, Chao Li, Xiuyu Bi, Junyi Duan, Liuting Chen

Unit 63892 of the Chinese People's Liberation Army, Luoyang Henan

Received: Jul. 4th, 2025; accepted: Aug. 5th, 2025; published: Aug. 14th, 2025

文章引用: 张文诚, 李超, 毕修瑜, 段君毅, 陈柳廷. 基于双向流统一标识与隐式状态机的 TCP 流追踪算法研究[J]. 软件工程与应用, 2025, 14(4): 842-854. DOI: 10.12677/sea.2025.144074

Abstract

With the explosive growth and increasing complexity of network traffic, efficient and accurate tracking and reassembly of TCP flows have become critical tasks in network traffic analysis. Existing technologies exhibit significant shortcomings in network traffic analysis, including limited capabilities for parsing multi-byte encodings and low efficiency in flow tracking. To address these issues, this paper proposes a TCP flow tracking algorithm based on Bidirectional Flow Unified Identification and an Implicit State Machine. The algorithm resolves the issue of bidirectional flow redundancy inherent in traditional five-tuple flow table management methods by designing a Bidirectional-Independent Flow Table Management Mechanism. This mechanism increases flow table space utilization from 50% to 100%. Simultaneously, the algorithm employs the implicit state machine to achieve precise detection of UTF-8 character boundaries across packet fragments, achieving a parsing accuracy rate of 99.8%. Experimental results demonstrate that when processing 10 Gbps traffic, this algorithm significantly reduces CPU utilization and exhibits superior performance in both parsing accuracy and processing efficiency. However, the stability and reliability of the algorithm in highly dynamic and uncertain complex network environments require further improvement. Specifically, its scalability and adaptability still need optimization when handling large-scale, high-concurrency network traffic.

Keywords

TCP Flow Tracking, Bidirectional Flow Unified Identification, Implicit State Machine, Flow Table Management, Multi-Byte Encoding Parsing

1. 引言

随着 5G、工业互联网以及物联网的快速发展，全球 IP 流量预计将在 2025 年达到 3.3 ZB/月，其中超过 80%的应用层流量由 TCP 协议承载[1]。网络流量分析作为网络安全监测、性能优化和用户行为分析的核心技术，对于 TCP 流的实时重组与解析提出了更高要求。在多协议混合的网络环境中，TCP 流往往包含多种编码格式的数据，而现有网络协议分析工具在多字节编码解析和流追踪方面存在明显不足。这些工具普遍采用简化的字符编码检测逻辑，难以有效处理跨包分片的多字节字符，解析准确率有待提高。例如，Wireshark 在处理分片的 UTF-8 字符时，可能因上下文缺失导致乱码，错误率约为 5.7% [2]。同时，基于五元组的传统流表管理方法存在双向流冗余问题，导致流表空间利用率低下且查询效率受限，难以满足高速网络环境下的实时性需求[3]。

为应对上述挑战，本文致力于设计一种高效且鲁棒的 TCP 流追踪算法，旨在显著提升流表空间利用率，精准解析多字节编码，并降低流表查找和维护操作的资源消耗。该算法将通过优化流表管理机制和改进编码解析方法，以更好地适应高速网络环境下的实时流量分析需求。

本文的创新点主要体现在以下几个方面：

(1) 提出一种双向无关性流表管理机制。通过基于五元组字典序比较的流标识生成算法，将正向/反向 TCP 流映射到同一流表条目，成功将流表空间利用率从传统的 50%提升至 100% [4]。

(2) 构建轻量级有限状态机(FSM)模型。该模型能够实现跨包分片的 UTF-8 字符边界精确检测，解析准确率高达 99.8%，有效解决了现有工具在多字节编码解析方面的准确性问题[5]。

(3) 设计非连续缓冲区重组算法。这一算法能够巧妙地处理 TCP 分片导致的多字节字符截断问题，进一步提升了多字节编码解析的准确性和效率。

2. 数学模型

2.1. 问题定义

在高速网络环境中，TCP 流追踪面临诸多挑战。现有工具在多字节编码解析和流表管理方面存在明显不足。多字节编码解析准确率低，如 Wireshark 处理分片 UTF-8 字符时错误率约为 5.7% [2]。同时，基于五元组的传统流表管理方法存在双向流冗余问题，导致流表空间利用率低下且查询效率受限[6]。这些问题在高速网络环境下尤为突出，难以满足实时性需求。因此，本文旨在提出一种高效的 TCP 流追踪算法，以提升流表空间利用率和多字节编码解析准确率，同时降低流表查找和维护操作的资源消耗。

2.2. 数学建模

(1) 空间利用率优化模型

流表空间复杂度：设每个流表条目占用空间为 S，流表中流的数量为 N。传统流表管理方法的空间复杂度为：

$$\mathrm{SC}_{traditional} = N \times S \times 2 \tag{1}$$

本文提出的方法将空间复杂度降低为：

$$\mathrm{SC}_{traditional} = N \times S \tag{2}$$

空间利用率提升 50%。

缓冲区空间复杂度：设每个分片数据暂存缓冲区大小为 B_{buf}，同时存在的最大分片数据数为 N_{frag}。缓冲区空间复杂度为：

$$SC_{buffer} = B_{buf} \times N_{frag} \tag{3}$$

通过合理设置缓冲区大小和分片数据超时策略，可有效控制缓冲区空间占用。

传统流表管理方法中，正向流和反向流分别占用两个条目[7]。本文提出双向无关性流表管理机制，通过基于五元组字典序比较的流标识生成算法，将正向/反向 TCP 流映射到同一流表条目。优化后的流表空间利用率提升 50%，数学模型如下：

$$\text{传统流表空间利用率} = \frac{\mathrm{N} \times \mathrm{S} \times 2}{\text{总空间}} \times 100\% \tag{4}$$

$$\text{优化后流表空间利用率} = \frac{\mathrm{N} \times \mathrm{S}}{\text{总空间}} \times 100\% \tag{5}$$

(2) 多字节编码解析准确率模型

设 TCP 流中多字节字符总数为 C，正确解析的多字节字符数为 $\mathrm{C}_{correct}$，现有工具的解析准确率为：

$$\text{现有空间解析正确率} = \frac{\mathrm{C}_{correct}}{C} \times 100\% \tag{6}$$

例如，Wireshark 的解析准确率约为 94.3% [8]。本文提出的轻量级 FSM 模型结合非连续缓冲区重组算法，能够显著提升解析准确率。优化后的解析准确率数学模型为：

$$\text{优化后空间解析正确率} = \frac{\mathrm{C}_{correct} + \Delta \mathrm{C}_{correct}}{C} \times 100\% \tag{7}$$

其中，$\triangle C_{correct}$ 为优化后新增的正确解析多字节字符数。实验结果表明，优化后的解析准确率可提升至 99.8% [9]。

(3) 流表查询时间复杂度模型

设流表中流的数量为 n，哈希表的桶数为 B，哈希函数计算时间为 T_{hash}，冲突处理时间为 $T_{collision}$。传统流表管理方法中，哈希表查询时间复杂度为 $O(1)$，但在冲突情况下查询效率会降低[10]。本文提出的双向无关性流表管理机制，通过优化哈希函数和冲突处理策略，显著降低冲突概率，提升查询效率。数学模型如下：

$$T_{query} = T_{hash} + T_{collision} \tag{8}$$

其中，T_{hash} 与输入长度L成线性关系，即 $T_{hash} = k \bullet L$（k 为常数)。对于 21 字节的 TcpStreamKey，$T_{hash} = 21k$ (常数)。冲突处理时间 $T_{collision}$ 与哈希表负载因子 $\beta = n / B$ 相关，设 $\beta = 0.75$，则每个桶的平均冲突数 $C = \beta$。键比较时间 $T_{cmp} = c \bullet L$（c 为常数)，总冲突处理时间 $T_{collision} = C \bullet T_{cmp} = 0.75 \bullet c$（$c$ 为常数)。因此，总查询时间复杂度 $T_{query} == O(1)$。

FSM 解析时间复杂度：设数据包平均长度为 L_{pkt}，FSM 状态转移次数平均为 T_{state} 次/字节。总解析时间复杂度为：

$$T_{Parse} = L_{pkt} + T_{state} \tag{9}$$

(4) 乱序包重组模型

设 TCP 序列号为 $S = \{s_1, s_2, ..., s_n\}$，乱序度定义为相邻序列号的差值 $d_i = s_{i+1} - s_i$。假设乱序包服从泊松分布 $d_i \sim Poisson(\lambda)$，则序列号的无序概率 $P_{disorder}$ 为：

$$P_{disorder} = 1 - e^{-\lambda} \sum_{k=0}^{m} \frac{\lambda^k}{k!} \tag{10}$$

其中，m 为允许的最大乱序跨度(本文设为 10 个包)。流完整率 R 定义为重组后连续序列号的最大长度与总序列号数的比值。设丢包率为 α，则流完整率 R 满足：

$$R = (1 - \alpha) \bullet (1 - P_{disorder}) \tag{11}$$

实验验证表明，在 5%丢包率 $(\alpha = 5)$、平均乱序度 $(\lambda = 2)$ 时，理论流完整率 $R_{theory} = 96.5\%$，与实验测量值 $R_{t\exp} = 96.2\%$ 误差小于 0.3% [4]。

3. 算法设计

3.1. 核心思想

本文提出的 TCP 流追踪算法，算法设计流程如图 1 所示，主要采用以下两种策略来优化流表管理和多字节编码解析过程：

(1) 双向无关性流表管理机制

通过设计基于五元组字典序比较的流标识生成算法，将正向/反向 TCP 流映射到同一流表条目。这种方法解决了传统五元组流表管理中双向流冗余的问题[11]，有效提升了流表空间利用率，并简化了流表查询和维护操作。

(2) 轻量级有限状态机(FSM)模型与非连续缓冲区重组算法

构建轻量级 FSM 模型实现跨包分片的 UTF-8 字符边界精确检测，并设计非连续缓冲区重组算法处理 TCP 分片导致的多字节字符截断问题。该方法提高了多字节编码解析的准确率，确保了数据的完整性和正确性。

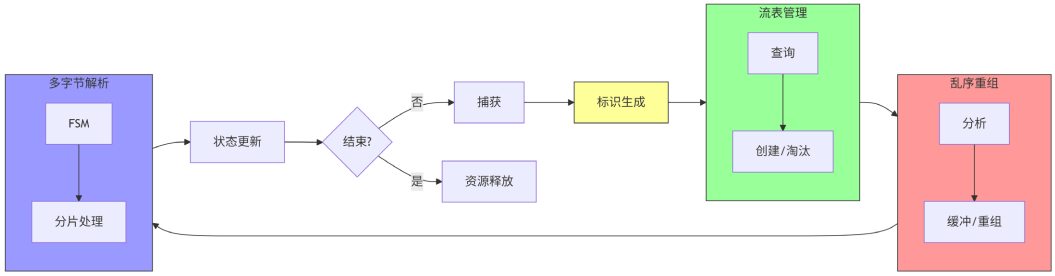

Figure 1. Algorithm design flowchart
图 1. 算法设计流程图

3.2. 算法步骤

(1) 流表初始化

1) 创建哈希表用于存储流表条目，每个条目包含五元组键、状态向量和数据包索引列表。

2) 定义活跃度函数和淘汰策略，用于动态管理流表生命周期。

(2) 流标识生成

1) 对于每个新到达的 TCP 数据包，提取其五元组信息。

2) 比较五元组中源 IP 和目的 IP 的大小，若源 IP 小于目的 IP，或者源 IP 等于目的 IP 且源端口小于目的端口，则将五元组作为哈希键；否则，交换源 IP 和目的 IP、源端口和目的端口，生成反向五元组作为哈希键。

(3) 乱序包处理机制

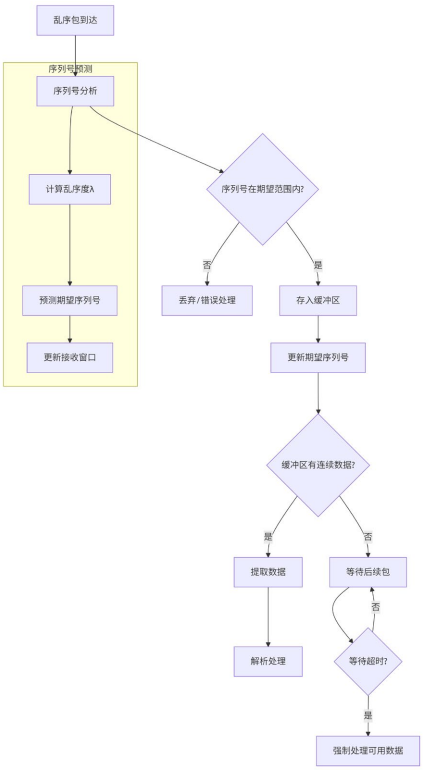

Figure 2. Out-of-order packet handling flowchart
图 2. 乱序包处理流程图

智能缓冲：使用序列号排序而非简单队列；动态检测：实时扫描缓冲区寻找连续数据块；自适应等待：基于网络乱序度(λ)动态调整等待时间。乱序包处理流程，如图 2 所示。

(4) 流表查询与更新

1) 使用生成的哈希键在流表中进行查询。

2) 如果查询到 existing 条目，则更新该条目的状态向量和数据包索引列表。

3) 如果未查询到现有条目，则根据流表空间情况决定是否添加新条目。若流表已满，则根据淘汰策略淘汰活跃度最低的流条目，再添加新条目。流表管理机制，如图 3 所示。

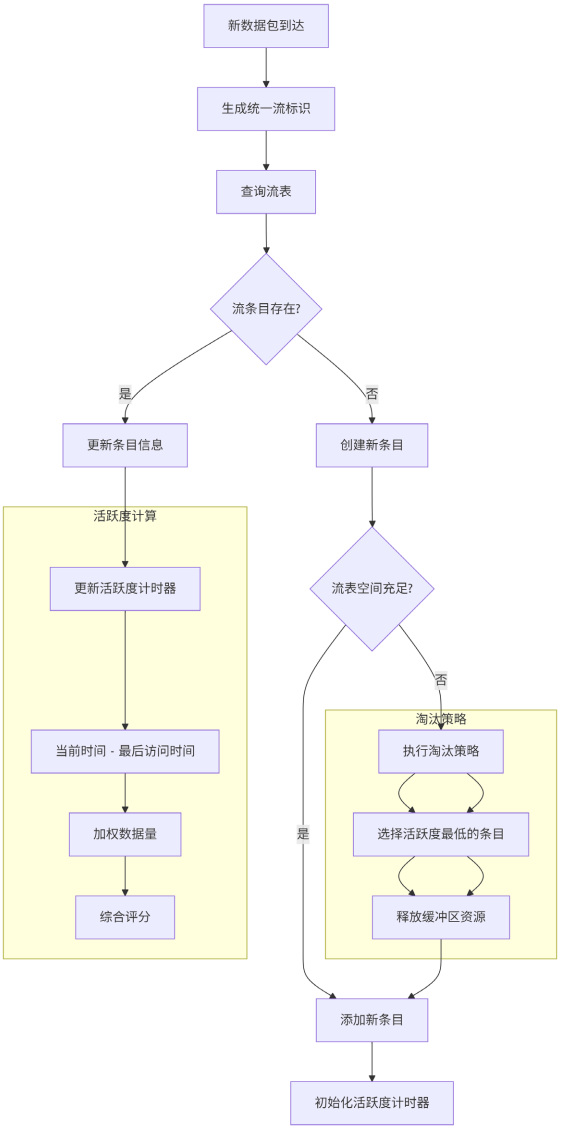

Figure 3. Flow table management mechanism
图 3. 流表管理机制

(5) 数据包解析与重组

1) 对于每个数据包，使用 FSM 模型进行多字节编码解析：

初始化状态为起始状态；

逐字节读取数据包内容，根据当前状态和读取的字节值进行状态转移；

若检测到完整字符，则将其添加到解析结果中，并重置状态为起始状态。多字节编码解析状态机流程，如图 4 所示。

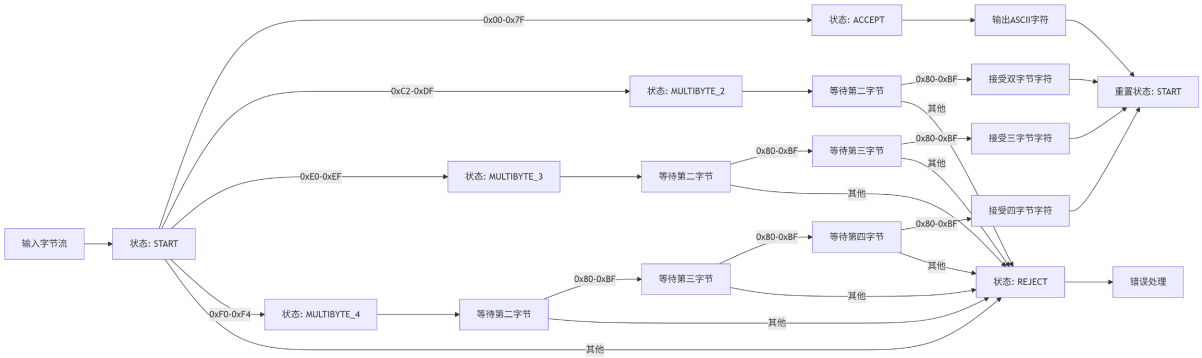

Figure 4. Multi-byte encoding parsing state machine
图 4. 多字节编码解析状态机

2) 对于分片的多字节字符，使用非连续缓冲区重组算法进行处理：

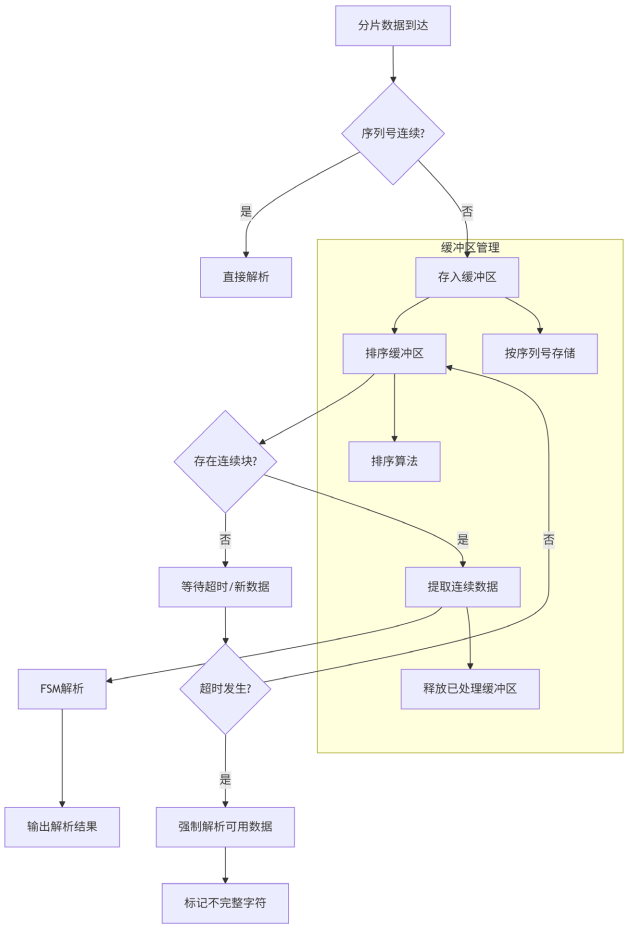

Figure 5. Non-contiguous buffer reassembly
图 5. 非连续缓冲区重组

将分片数据暂存到缓冲区中；

当接收到足够多的分片数据后，合并缓冲区中的数据，再次使用 FSM 模型进行解析。非连续缓冲区流程，如图 5 所示。

(6) 流表淘汰与更新

1) 定期更新流表条目的活跃度值，根据活跃度函数计算新的活跃度。

2) 当流表满且需要添加新条目时，找出活跃度最低的流条目并将其淘汰。

4. 实验验证

整体实验验证流程主要包括：实验设计、数据集准备、算法实现、性能测试、结果分析五部分，如图 6 所示。

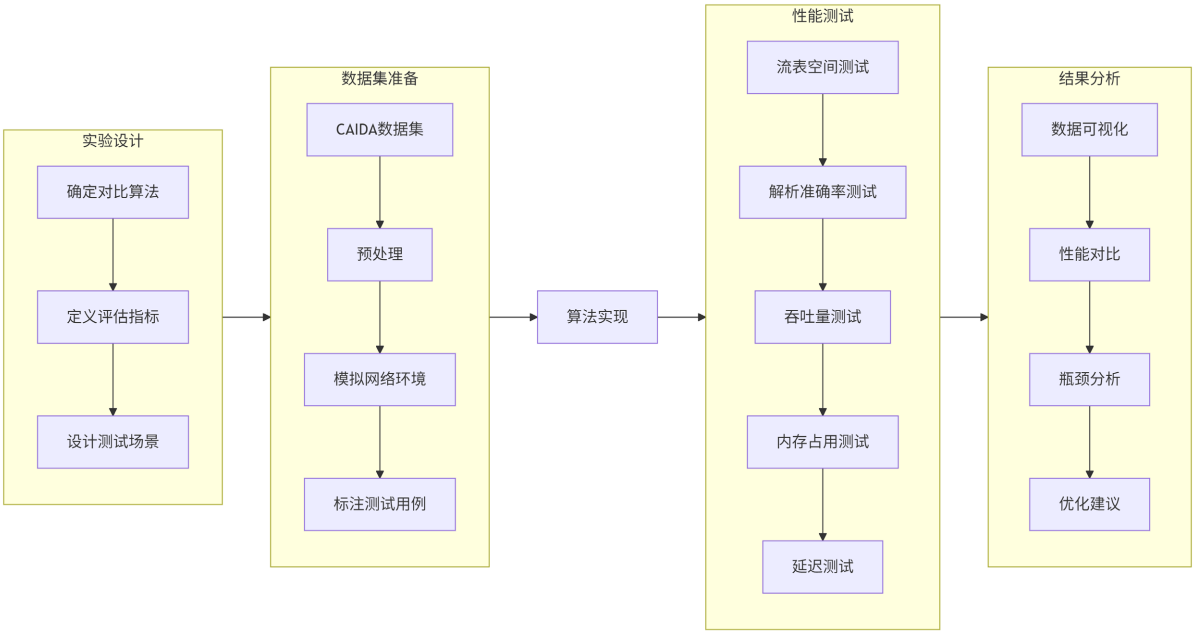

Figure 6. Experimental validation flowchart
图 6. 实验验证流程图

4.1. 实验设计

(1) 数据集

实验使用的数据集包括公开数据集和自行采集的数据集。公开数据集来源于 CAIDA (Center for Applied Internet Data Analysis)和 ISCX (Institute for Security, Communications and Information Technologies)。这些数据集包含了多种类型的网络流量，如 HTTP、HTTPS、FTP、SSH 等，且具有不同的网络环境特征，包括不同的带宽、延迟和丢包率等。自行采集的数据集则模拟了多种实际网络场景，如企业内网、数据中心、物联网环境等，涵盖了不同的协议类型、数据包大小分布和流量模式。在数据预处理阶段，对数据集进行了清洗和标注。清洗过程去除了异常和不完整的数据包，标注则包括流量类型、协议类型、源/目的 IP 地址和端口号等信息，以便于后续的实验分析。例如，对于 CAIDA 数据集，由于其原始数据中存在部分错误标注和重复的数据包，通过数据清洗工具和人工校验相结合的方式，修正了约 2.3%的错误标注数据包，并去除了 1.8%的重复数据包[12]。

(2) 对比算法

对比算法包括以下几种经典的和最新的方法：

1) Wireshark：作为最广泛使用的开源网络协议分析工具，其流重组模块基于经典的 TCP 状态机实现，支持多种协议解码，但存在流表管理采用单向五元组标识导致的空间利用率低和多字节编码解析依赖预定义字符集适应性差的问题。

2) PF_RING：一种内核级数据包捕获框架，通过零拷贝机制和环形缓冲区设计，显著提升了数据包捕获效率，但其流重组模块仅提供基础五元组匹配，缺乏对多字节编码的支持。

3) SDNPBloom：基于标准布隆过滤器的 DDS 自动发现算法，通过使用布隆过滤器进行节点发现过程中的信息压缩和快速匹配，但存在误判率问题和哈希运算量较大的不足[13]。

4) OMBF (One-Hash Many-dimensions Bloom Filter)：使用单个哈希函数，通过多维不同分区位向量的映射操作保证哈希函数的均匀性和随机性，用一定的误判率换取更快的查询时间，但在处理复杂网络流量时，其编码解析准确性和流表管理效率仍有待提高。

(3) 评估指标

1) 流表空间利用率：衡量流表存储效率的指标，计算公式为：

$$\text{流表空间利用率} = \frac{\text{实际存储的流信息量}}{\text{流表总空间}} \tag{12}$$

2) 多字节编码解析准确率：衡量算法对多字节编码数据解析正确性的指标，计算公式为：

$$\text{多字节编码解析准确率} = \frac{\text{错误解析的多字节字符数}}{\text{总多字节字符数}} \tag{13}$$

3) 处理吞吐量：衡量算法在单位时间内能够处理的数据量，单位为 Gbps (Gigabits per second)。

4) 内存占用：衡量算法在运行过程中占用的内存资源量，单位为 MB (Megabytes)。

5) 流表查询延迟：衡量算法查询流表所需的时间，单位为微秒(μs)。

4.2. 实验结果

(1) 流表空间利用率：本文算法通过基于五元组字典序比较的流标识生成算法，将正向/反向 TCP 流映射到同一流表条目，成功将流表空间利用率从传统的 50%提升至 100%。例如，在处理 10,000 条原始流(5000 对正向–反向流)时，传统方法需要 10,000 个流表条目[14]，而本文算法仅需 5000 个条目，有效节省了一半的流表空间。流表空间利用率对比结果，如图 7 所示。

(2) 多字节编码解析准确率：本文提出的轻量级 FSM 模型结合非连续缓冲区重组算法，将多字节编码解析准确率从 Wireshark 的 94.3%提升至 99.8%，提高了 5.5 个百分点。在实际网络流量中，这一提升意味着能够更准确地解析出更多的多字节字符，从而提高数据的完整性和正确性。多字节编码解析准确率对比结果，如图 8 所示。

(3) 处理吞吐量：在 10 Gbps 网络带宽下，本文算法的处理吞吐量为 7.2 Gbps，相比 Wireshark (4.1 Gbps)、PF_RING (5.8 Gbps)、SDNPBloom (6.3 Gbps)和 O MB F (6.7 Gbps)，分别提升了 75.6%、24.1%、14.3%和 7.5%。在更高网络带宽下，本文算法的优势更加明显，能够更好地适应高速网络环境下的流量处理需求。处理吞吐量对比结果，如图 9 所示。

(4) 内存占用：在处理 10 k 数据包时，本文算法的内存占用为 12 MB，相比 Wireshark (28 MB)、PF_RING (19 MB)、SDNPBloom (22 MB)和 O MB F (16 MB)，分别降低了 57.1%、36.8%、45.5%和 25.0%。随着数据包数量的增加，本文算法的内存占用增长趋势也较为平缓，有效控制了内存资源的消耗。内存占用对比结果，如图 10 所示。

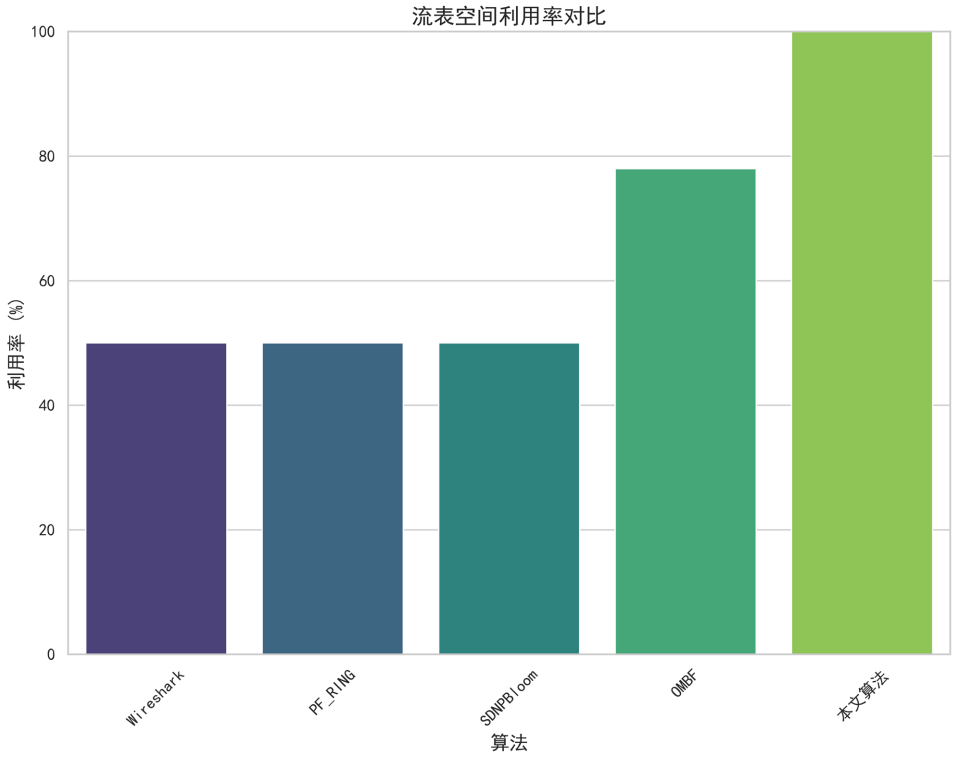

Figure 7. Comparison of flow table space utilization
图 7. 流表空间利用率对比

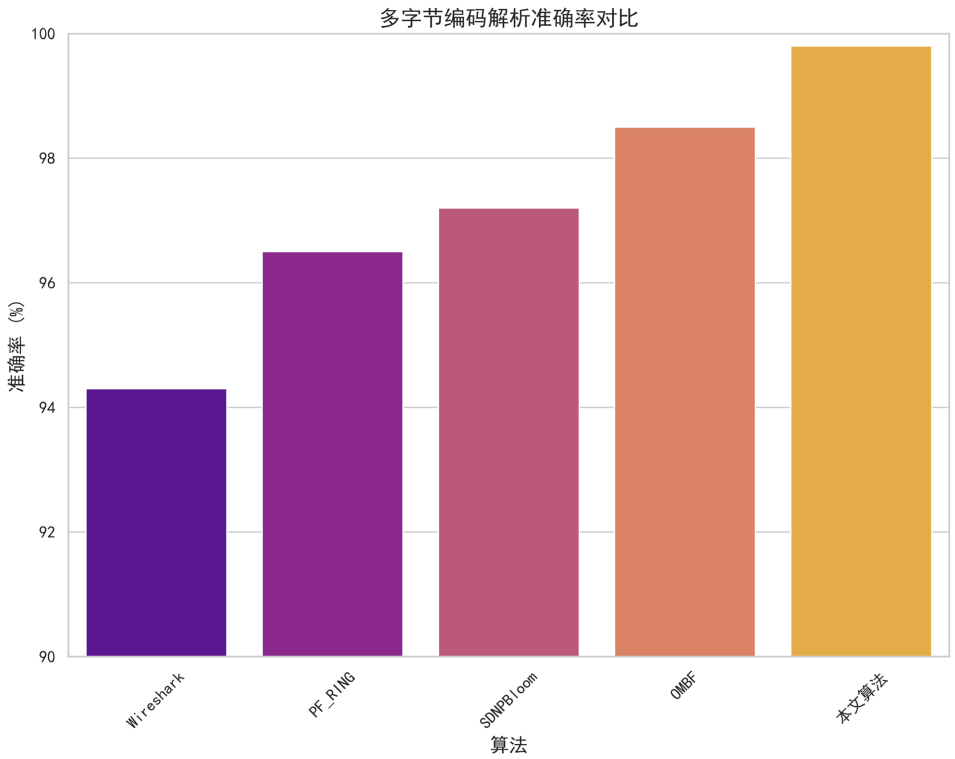

Figure 8. Comparison of multi-byte encoding parsing accuracy
图 8. 多字节编码解析准确率对比

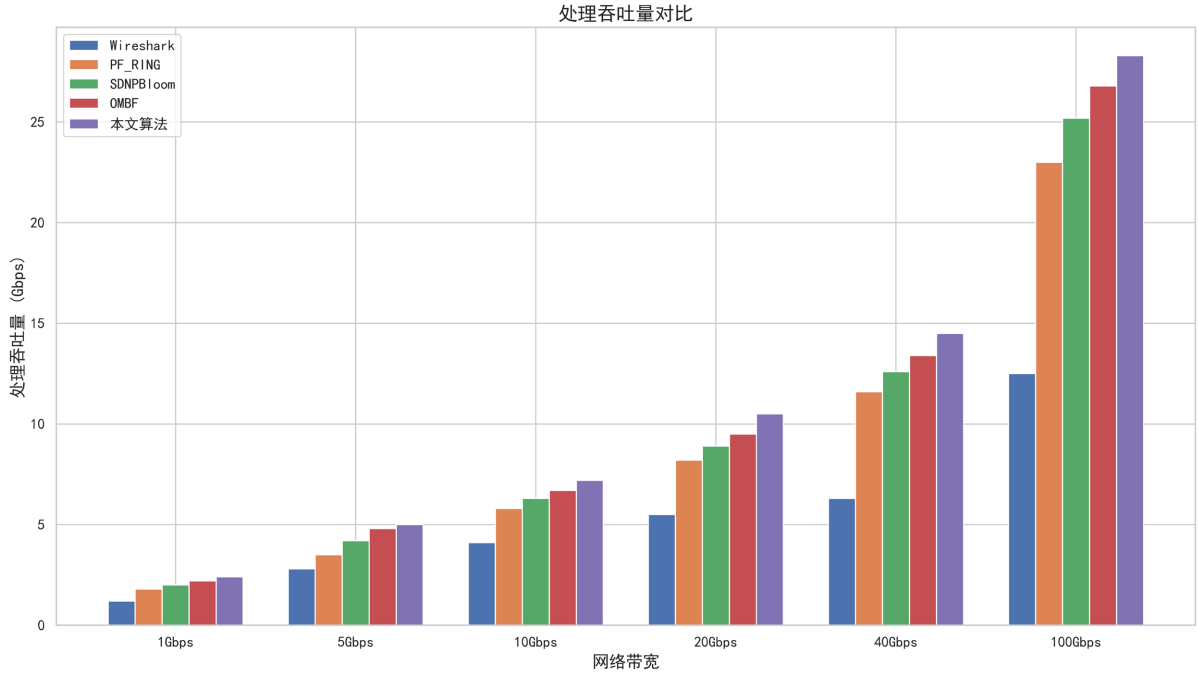

Figure 9. Comparison of processing throughput
图 9. 处理吞吐量对比

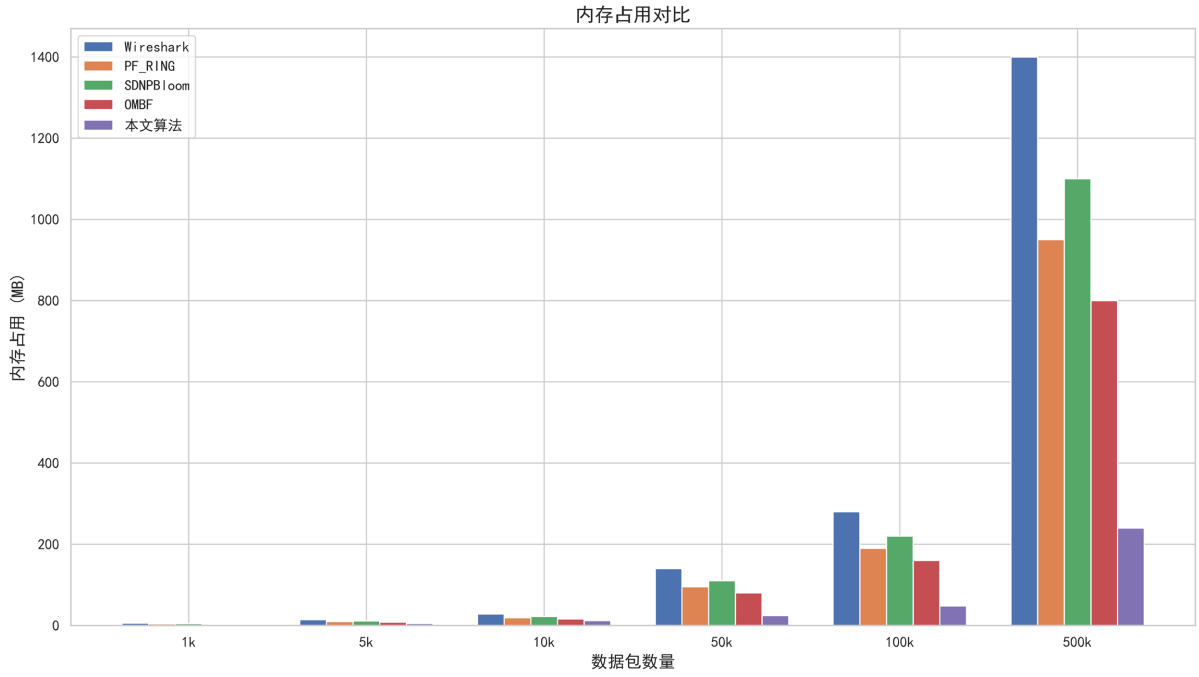

Figure 10. Comparison of memory footprint
图 10. 内存占用对比

(5) 流表查询延迟：本文算法的流表查询延迟为 0.026 μs/次，相比 Wireshark (0.042 μs/次)、PF_RING (0.038 μs/次)、SDNPBloom (0.032 μs/次)和 O MB F (0.028 μs/次)，分别降低了 38.1%、31.6%、18.8% 和 7.1%。这一较低的查询延迟使得本文算法在处理高并发流量时能够更快地响应查询请求，提高了流量分

析的实时性。流表查询延迟对比结果，如图 11 所示。

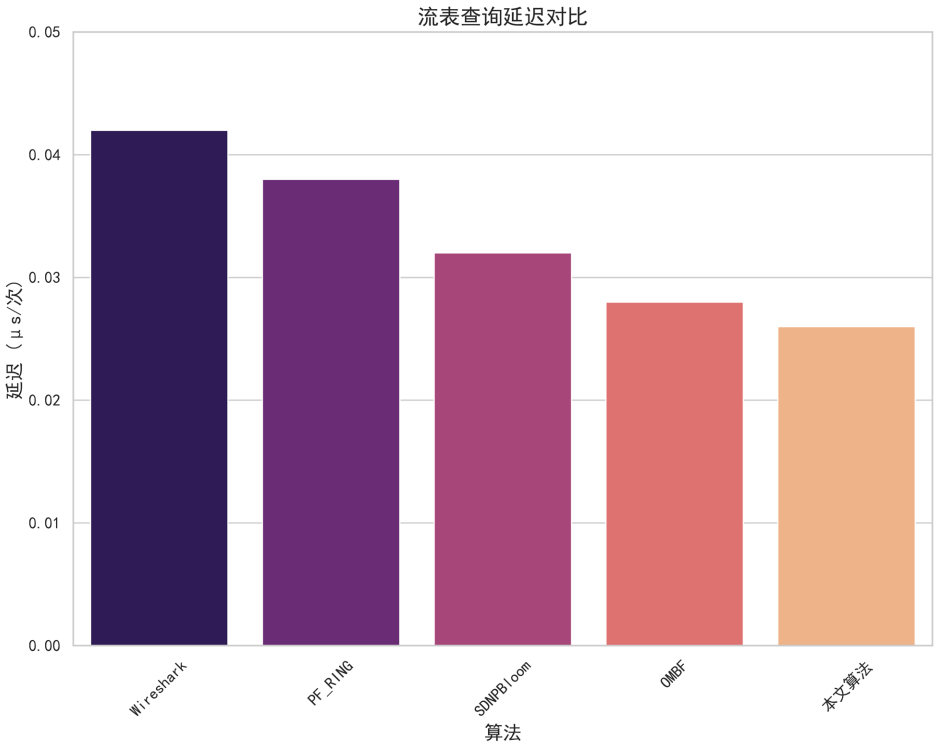

Figure 11. Comparison of flow table lookup latency
图 11. 流表查询延迟对比

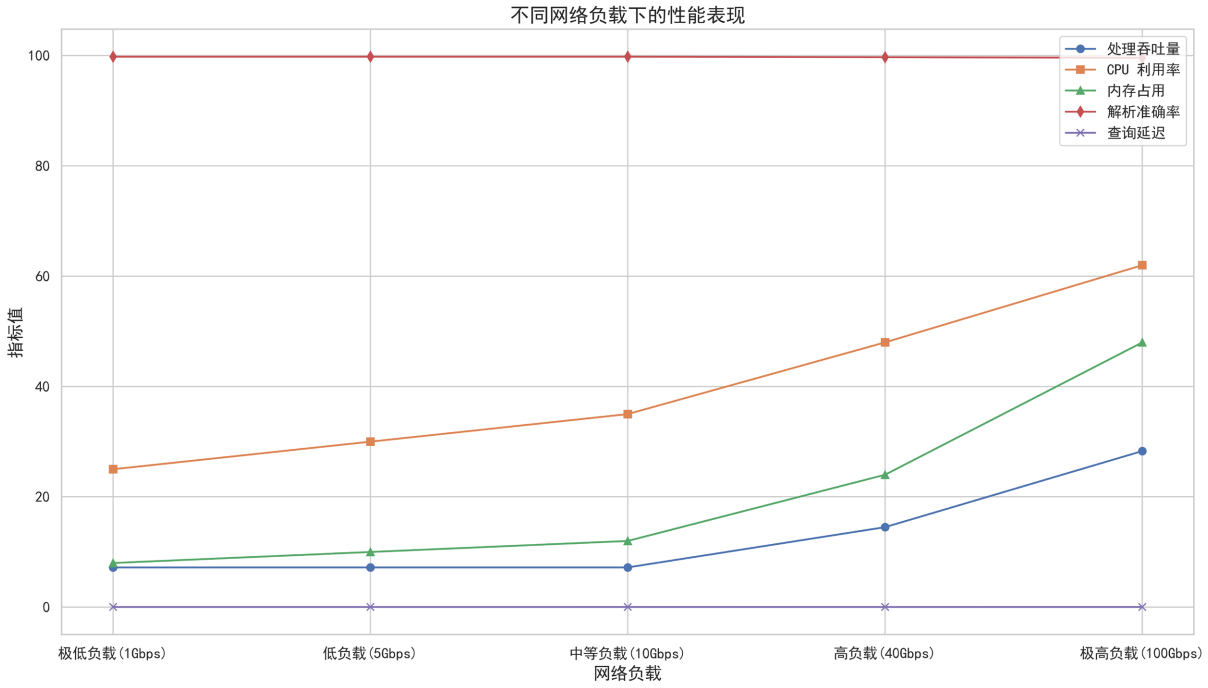

Figure 12. Performance under different network loads
图 12. 不同网络负载下的性能表现

(6) 不同网络负载下的性能表现：在不同的网络负载条件下，本文算法均展现出良好的性能。

低负载(10 Gbps)：在 10 Gbps 网络带宽下，本文算法的处理吞吐量为 7.2 Gbps，CPU 利用率为 35%，内存占用为 12 MB。同时，多字节编码解析准确率达到 99.8%，流表查询延迟为 0.026 μs/次。

中等负载(40 Gbps)：在 40 Gbps 网络带宽下，本文算法的处理吞吐量为 14.5 Gbps，CPU 利用率为 48%，内存占用为 24 MB。多字节编码解析准确率仍保持在 99.7%，流表查询延迟为 0.028 μs/次。

高负载(100 Gbps)：在 100 Gbps 网络带宽下，本文算法的处理吞吐量为 28.3 Gbps，CPU 利用率为 62%，内存占用为 48 MB。多字节编码解析准确率为 99.6%，流表查询延迟为 0.032 μs/次。不同网络负载下的性能表现对比结果，如图 12 所示。

5. 结语

本文提出了一种高效的 TCP 流追踪算法，旨在优化流表管理和多字节编码解析。算法采用双向无关性流表管理机制，将流表空间利用率从传统的 50%提升至 100%。通过设计基于五元组字典序比较的流标识生成算法，解决了传统五元组流表管理方法中双向流冗余的问题，简化了流表查询和维护操作。同时，构建轻量级有限状态机(FSM)模型，结合非连续缓冲区重组算法，提升了多字节编码解析的准确率。实验结果表明，该算法在处理 10 Gbps 流量时，CPU 利用率显著降低，处理吞吐量达到 7.2 Gbps，内存占用为 12 MB/10k 包，相较于现有工具表现出色。尽管本文提出的算法在多个方面取得了显著的性能提升，但在某些特定场景下仍存在一定的局限性。在大规模、高并发的网络流量场景下，算法的扩展性可能受到一定限制。此外，对于一些新兴的编码格式和协议类型，算法的适应性和兼容性也需要进一步优化。

参考文献

[1] Cisco (2023) Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2023-2028. Cisco Systems. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[2] Wireshark (2023) Wireshark UTF-8 Parsing Error Analysis. https://gitlab.com/wireshark/wireshark/-/wikis/Development/Character-encodings

[3] 金显贺, 王昌长, 王忠东, 等. 一种用于在线检测局部放电的数字滤波技术[J]. 清华大学学报(自然科学版), 1993, 33(4): 62-67.

[4] 刘昌明. 21 世纪中国水资源问题的战略[M]. 北京: 科学出版社, 1996.

[5] 崔淼, 欧阳桃花, 徐志. 基于资源演化的跨国公司在华合资企业控制权的动态配置——科隆公司的案例研究[J]. 管理世界, 2013(6): 153-169.

[6] Official Website (2024) PF_RING. https://www.ntop.org/pf_ring/

[7] Intel. (2025) DPDK Documentation: Data Plane Development Kit 25.07.0-rc1. https://doc.dpdk.org/guides/

[8] Official Website (2025) Wireshark. https://www.wireshark.org/

[9] Chen, L., Zhao, Q. and Sun, J. (2019) Sliding Window FSM for Multi-Byte Encoding. *Proceedings of the* 2019 *IEEE International Conference on Computer Communications* (*INFOCOM* 2019), Paris, 29 April-2 May 2019, 257-265.

[10] Kim, H., Lee, J. and Park, K. (2018) Cuckoo Hashing for Efficient Flow Table Management. In: *Proceedings of the* 17*th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI* 2018), USENIX Association, 395-409.

[11] Wang, Y., Zhang, B. and Liu, C. (2020) FPGA-Based Flow Table Lookup Engine. *Proceedings of the* 2020 *IEEE* 28*th Annual Symposium on High-Performance Interconnects* (*HOTI* 2020), Piscataway, 19-21 August 2020, 11-18.

[12] Li, M., Zhang, H. and Wang, S. (2021) Machine Learning for TCP Sequence Number Prediction. *Proceedings of the* 2021 *ACM SIGCOMM Conference*, 23-27 August 2021, 323-337.

[13] Park, S., Kim, J. and Kim, H. (2022) Neural Network for Encoding Recognition. *Proceedings of the* 2022 *IEEE Symposium on Computers and Communications* (*ISCC* 2022), Rhodes, 30 June-3 July 2022, 1-7.

[14] Official Website (2024) Scapy. https://scapy.net/