面向持久性连接的自适应拟态表决器设计与实现

周大成，陈鸿昶，程国振，何威振，商珂，扈红超
（信息工程大学信息技术研究所，河南 郑州 450001）

摘 要：针对当前拟态表决器以连接为单位“连接内数据传输结束后再表决、转发”的机制难以适应 HTTP 1.1 协议在持久性连接、分块传输编码的应用场景中开销过大的问题，设计实现了面向持久性连接的自适应拟态表决器，在数据持续传输过程中滑动窗口式表决、释放异构冗余执行体的分块传输编码报文，通过分析滑动窗口式表决的分块传输编码报文的数据特征，构建了表决算法选择策略集，给出了表决准确性维护方案；提出了基于存贮模型的自适应表决窗口控制策略，为待表决数据提供最佳的切分方案。基于原型系统的实验结果表明，自适应拟态表决器在具有可接受的表决准确度下，降低了内存开销并提升了拟态表决效率。

关键词：拟态防御；拟态表决；超文本传输协议；持久性连接；分块传输编码
中图分类号：TP309.1
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2022081

Design and implementation of adaptive mimic voting device oriented to persistent connection

ZHOU Dacheng, CHEN Hongchang, CHENG Guozhen, HE Weizhen, SHANG Ke, HU Hongchao
Institute of Information Technology, Information Engineering University, Zhengzhou 450001, China

Abstract: Aiming at the problem that the current mimic voting usually adopts the mechanism of voting and forwarding after collecting all the transmission data in the connection, which has heavy overhead the application scenarios of the HTTP 1.1 protocol with the form of the persistent connection and the chunked transfer encoding, an adaptive mimic voting device was designed and implemented, where the chunked transfer encoded packets from heterogeneous redundant executives were voted and then released in the order as a sliding window during the voting device receives packets in the persist connections. By analyzing the characteristics of the chunked transfer encoded packets of the sliding window voting, a voting algorithm selection strategy set was constructed to maintain the voting accuracy, and an adaptive voting window control strategy based on the inventory model was proposed to provide the best segmentation scheme for the chunked transfer encoded packets to be voted. The experimental results based on the prototype system show that the adaptive mimic voting device reduces the memory overhead and improves the mimic voting efficiency with acceptable voting accuracy.

Keywords: mimic defense, mimic voting, HTTP, persistent connection, chunked transfer encoding

0 引言

网络空间拟态防御（CMD, cyber mimic defense）[1]技术是一种内生安全的解决方案，通过动态异构冗余（DHR, dynamic heterogeneous redundancy）架构将难以观测的未知漏洞后门引起的网络威胁事件转化为拟态表决器可以判别的差模扰动现象，从而达到发现并处置网络威胁的目的。异构

收稿日期：2022−01−06；**修回日期**：2022−03−24
通信作者：程国振，guozhencheng@hotmail.com
基金项目：国家自然科学基金资助项目（No.62072467）；国家重点研发计划基金资助项目（No.2021YFB1006200, No.2021YFB1006201）
Foundation Items: The National Natural Science Foundation of China (No.62072467), The National Key Research and Development Program of China (No.2021YFB1006200, No.2021YFB1006201)

冗余执行体的架构和存在的漏洞具有差异性，经有效的异构执行体调度，共同漏洞出现的可能性显著降低[2]。因此，一种特定的漏洞利用方法难以同时扰动所有的异构冗余执行体。而攻击基于 DHR 架构的软件需同时扰动所有或大多数异构冗余的软件执行体，其攻击成功的概率显著降低。

拟态表决器是 DHR 架构安全输出和异常检测的关键组件，从异构的软件执行体对同一攻击的差异性响应出发，以大数判决[3-4]为基础，通过交叉比较异构冗余的软件执行体的行为特征确定是否存在网络攻击。为了提高 DHR 架构下系统的安全性和可用性，研究者针对拟态表决方法展开了广泛研究。文献[5-7]从语义相似度分析出发，提出针对异构同义数据的表决方法，提高拟态表决器的环境适应性；文献[8-9]从异构冗余执行体的历史行为出发，提出基于统计的表决方法，给拟态表决器的多模逃逸问题提供解决思路；文献[10-11]通过综合分析表决过程中多个要素的特征，提出基于多目标决策的表决方法，进一步降低拟态表决器的虚警概率；文献[12-13]基于异构冗余执行体的任务完成时间，设计优先表决输出的方法，提高拟态表决器的性能。

但是，现有拟态表决方法普遍依赖异构冗余执行体的完整输出数据。随着基于 HTTP 1.1 协议持久性连接传输数据的应用场景越来越多，如微服务中需要持久性连接来完成频繁的应用程序接口（API, application program interface）调用[14]、流媒体以持久性连接提供不间断的高效数据输出[15]，拟态表决器的设计面临新的挑战。HTTP 1.1 协议可通过持久性连接和分块传输编码技术完成数据流传输，客户端可以通过一个 TCP 连接按序发送多个 HTTP 请求包，服务端可以将数据量较大或动态生成的数据以分块传输编码，形成多个 HTTP 分块报文，依次发送给客户端，从而达到充分利用网络带宽、不间断传输数据的目的。现有关于拟态表决方法的相关研究主要关注完整输出内容下表决的准确性，通常采用以连接为单位的“连接内数据传输结束后再表决、转发”的数据管理机制，即存储持久性连接中异构冗余执行体输出所有分块数据并拼接成完整的输出内容再对数据进行表决和转发处理。为方便表述，本文记这种数据管理机制为 CAC（collect all the transmission content in the connection）机制。CAC 机制在处理 HTTP 1.1 分块报文时存在严重的问题：一方面，拟态表决器存储异构冗余执行体在持久性连接中输出的所有分块报文需消耗大量内存资源；另一方面，拟态表决器对连续传输的分块报文的阻塞处理会破坏数据传输的连续性，增大系统的响应时延。

为了解决上述问题，本文从分块报文的动态性与拟态表决要求数据完整性的矛盾出发，通过自适应切分到达表决器的异构冗余执行体的分块报文得到多个待表决的分块报文块（定义为表决块），以滑动窗口的方式依次对表决块进行动态表决与输出，同时在数据持续传输过程中逐步释放完成处理的表决块，在持久性连接中保持数据传输的连续性并降低拟态表决器的内存消耗和处理时间。此外，本文基于分块报文的特征，通过策略性选取适配表决块长度范围和数据类型的表决算法，避免不完整的分块报文的表决不准确的问题。

本文的主要研究工作及贡献如下。

1) 分析拟态表决器处理持久性连接的分块报文流的过程及存在的问题，给出自适应拟态表决器的架构设计，提出同步开展数据存储与表决的自适应拟态表决算法。

2) 针对自适应拟态表决器切分的数据不完整性带来的表决准确性不高的问题，给出兼顾表决效率和准确性的表决算法选取策略。

3) 基于存贮理论[16]对自适应拟态表决器的数据收发过程进行建模分析，并提出基于存贮模型的自适应表决窗口控制策略，为分块报文的表决块切分操作提供最优方案。

4) 修改 Nginx 源代码实现自适应拟态表决器的原型系统，基于 Web 服务器的分块传输编码传输数据的场景验证所提方案的有效性。通过与传统的拟态表决器的对比实验可知，自适应拟态表决器在保证表决准确性的前提下可以降低内存开销并提高表决效率。

1 研究背景

1.1 场景分析

DHR 架构如图 1 所示，输入代理将输入数据复制 n 份，并且分发给 n 个异构执行体构成的执行体集同时处理。表决器接收 n 个异构执行体的输出数据，经表决得到相对正确的结果输出，同时将表决异常的执行体记录或上报。其中，执行体集由异构构件集合经动态选择算法生成，并根据运行时的反馈信息动态更新。

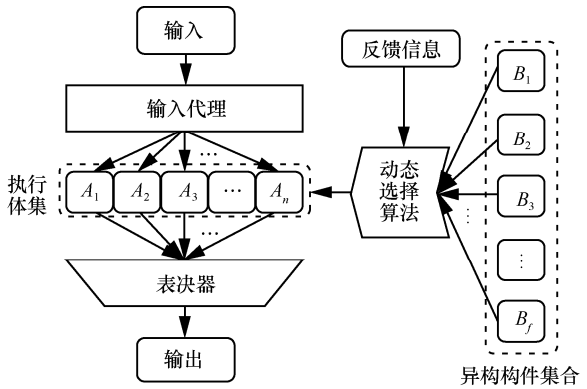


图 1　DHR 架构

由图 1 可知，表决器对异构执行体输出的数据进行表决处理，但是在持久性连接中传输分块报文的场景中存在挑战。本文搭建拟态 Web 服务器[5]的实验环境，配置 Web 执行体以分块传输编码来传输 20 MB 的页面资源。在通过拟态 Web 服务器访问该资源时，本文从表决器与 3 个 Web 执行体分别建立的持久性连接中分别抓取 Web 执行体输出的所有分块传输的报文，并解析其长度和传输的时刻，得到 3 个 Web 服务执行体的分块报文特征，如图 2 所示。从图 2 可以看到，执行体基于分块传输编码技术，将待传输的数据切分为一系列数据块，以分块报文的形式通过持久性连接依次发送出去。这一过程存在以下特点。

1) 传输时间长。由于传输的数据总量较大或者动态生成，其传输的时间较长，如图 2 中传输的总时长为 1.631 4~2.124 s。

2) 各个分块报文长度随机。分块传输编码切分数据块取决于网络带宽资源状态，其切分长度存在随机性，如图 2 中分块报文的长度在 1.5 KB 和 30 KB 之间随机分布。

3) 各个执行体输出分块报文的时刻随机。异构冗余执行体各自独立地对数据进行分块传输编码，其分块报文的传输完成时间和发送时间各不相同。

以上现象反映了在 HTTP 1.1 的持久性连接场景下，Web 执行体会将较大的页面资源经过分块传输编码切分为多个大小不一的分块报文并依次、陆续地输出，从而达到在持久性连接中接连不断地传输数据的目的。此外，由于各个 Web 执行体的分块传输编码相互独立，因此各自传输的分块报文在时间和空间上存在较大的差异，这给依赖完整语义的拟态表决算法带来较大的困难，主要表现在 CAC 机制下需要较长时间的等待和对齐数据才能对数据进行表决处理。表决器在等待的过程中存储陆续收到的分块报文，也造成了大量的内存占用的问题。图 3 是拟态 Web 服务的表决器与非拟态 Web 服务的反向代理服务器通过持久性连接传输分块报文的过程中采集到的空闲物理内存变化情况。持久性连接传输分块报文的场景给采用 CAC 机制的表决器带来的问题可通过图 3 所示的现象来直观说明，具体介绍如下。

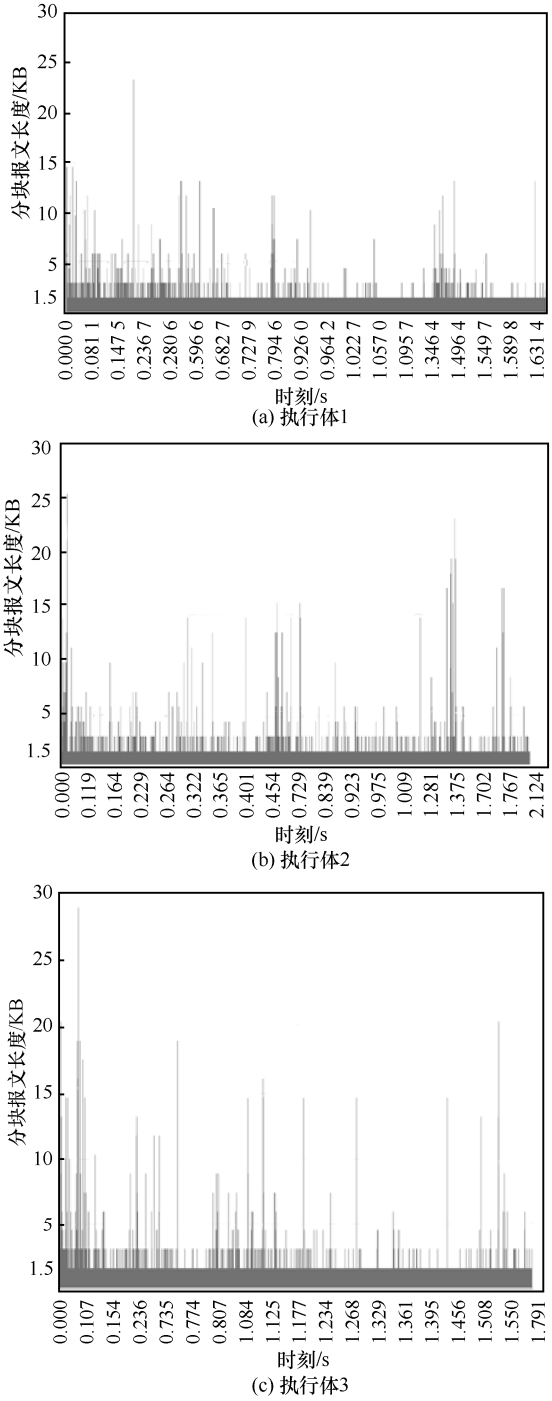


图 2　分块报文特征

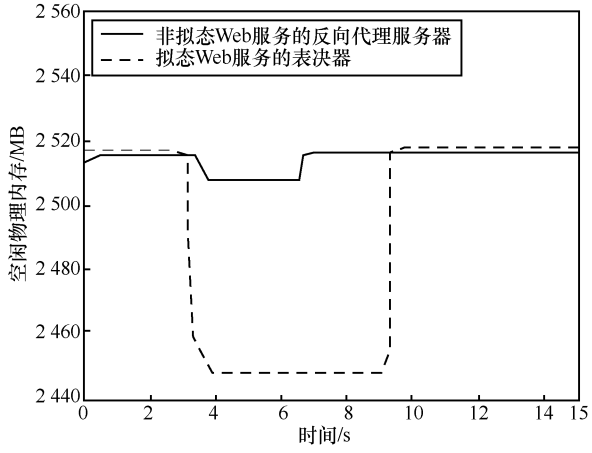


图 3　拟态表决器过度消耗内存资源

1) 长时间阻塞。由于持久性连接中报文传输时间长，采用 CAC 机制的表决器需要长时间阻塞接收执行体输出的所有分块报文，并且需要消耗几乎同等的时间将表决结果输出。此外，一次性表决持久性连接中传输的所有报文也需要较长的处理时间。如图 3 所示，Web 服务的表决器的内存占用时间几乎是非拟态 Web 服务的反向代理服务器的 2 倍。

2) 大量内存资源消耗。如图 3 所示，拟态 Web 服务的表决器的内存占用量远大于非拟态 Web 服务的反向代理服务器。这是因为非拟态 Web 服务的反向代理服务器透明转发 Web 服务的表决器输出的分块报文，即依次存储、释放若干个长度较小的分块报文，占用的内存较少；而采用 CAC 机制的拟态表决器进行表决处理的前提是需要存储异构冗余执行体的一次连接传输的所有分块报文，占用的内存成倍增加。

总之，现有的拟态表决器由于要求数据完整，需存储大量的待表决数据，这不仅占用了大量内存空间，而且丢弃了持久性连接中传输分块报文带来的性能优势。

1.2　相关研究

拟态表决以全体一致表决算法[3]、多数表决算法[4]等为基础，基于语义相似度、历史统计、多要素分析开展研究，克服由于数据异构、偶发性虚警、多要素相互矛盾等问题带来的表决准确度问题。

基于语义相似度的研究集中在解决异构输出数据的可表决性问题。仝青等[5]在拟态 Web 服务器中应用基于语义特征的抽取方法对执行体输出内容进行表决，解决不同排列格式的数据表决问题。在语义解析方面，张文建等[6]提出可编程语义解析方法来解决拟态表决过程中不同协议的解析问题，并引入算法提高解析效率。马博林等[7]基于字符串编辑距离和最长公共子串求解文本相似度，解决动态异构 Web 服务系统中出现的网页篡改的表决问题。Qi 等[8]对拟态软件定义网络（SDN, software defined network）中的流表进行语义抽样和分段打分，给出异构 SDN 控制器下流表信息归一化的方法。

基于历史统计的研究致力于解决单次表决中偶然出现的虚警带来的表决准确性问题。欧阳城添等[9]基于统计历史表决结果，构建表决历史置信度，在表决过程中侧重于输出历史执行度较高的执行体所输出的数据，避免频繁的单次表决虚警给系统带来扰动。Hu 等[10]通过对软硬件的漏洞后门的数量进行统计给出执行体可信度评分的方法，并基于该评分在表决过程中给对应执行体赋予相应的权重。

基于多要素分析的研究主要解决表决过程中多要素存在矛盾的问题。陆以勤等[11]基于多指标决策提出一种改进层次分析法的表决算法，将拟态表决转化为模糊评价过程，通过多个指标之间的共同校验提高表决的正确率。Zhou 等[12]提出基于关键特征统计和层次分析法的拟态表决方法，通过综合考虑 Web 场景下的状态码、报文关键字、威胁关键字等要素，构建面向威胁特征的层次分析的评价方法。

在拟态表决效率方面，林森杰等[13]基于多数一致表决，提出一种竞赛式的仲裁模型，通过增加异构执行体的任务副本并选择领先的任务输出进行一致性表决，从而提高表决效率。Wang 等[14]提出一种延迟决策机制来检查任务执行结果，通过多阶段检查任务执行结果，保证满足多数一致的数据尽早输出，从而提高系统的性能。王禛鹏等[17]使用对服务器输出内容直接投票的方法提高拟态 DNS 服务器的效率。但是，以上方法没有关注持久性连接内的数据表决中的表决效率问题。张铮等[18]对拟态 Web 服务器在 HTTP 1.1 协议下的数据传输进行了功能和性能测试，但未考虑持久性连接内分块报文的传输连续性问题。综上，现有研究中缺乏在持久性连接传输分块报文的场景下设计拟态表决器的研究。

2　自适应拟态表决器架构设计

拟态表决器在处理持久性连接中分块报文的表决时面临的核心问题是动态连续的分块报文序列难以满足现有拟态表决方法对报文的完整性要

求。为了解决这一问题，本文通过将拟态表决过程动态化来适应碎片化的分块报文，设计在接收分块报文的同时自适应地对已有数据进行表决输出的拟态表决器，即自适应拟态表决器。自适应拟态表决器在接收异构冗余执行体输出的分块报文的同时对现存数据进行拟态表决和输出，减少待处理数据在拟态表决器上的驻留，从而降低内存消耗。在此过程中，自适应拟态表决器对现存的分块报文进行切分，并以滑动窗口的方式进行拟态表决处理和输出，在数据持续传输过程中逐步释放已表决分块报文，从而降低拟态表决的时间消耗。为达到这一目的，自适应拟态表决器主要面临以下 2 个挑战。

1) 需要保证不完整的数据报文的拟态表决准确性。异构的服务执行体的输出数据具有异构同义的特点，表决器将分块报文序列切分为表决块进一步加剧了表决过程中的报文差异性，影响表决准确性。

2) 需要确定单次表决数据的长度，即表决块大小。从持续增长的待表决数据中切分表决块是自适应拟态表决器的核心流程，关系着表决的速率和表决器的内存占用量。过长的表决块切分的前提是收集更多的分块报文，这会占用过多的存储空间同时也会增加表决的处理时间；过短的表决块切分会进一步破坏报文的完整性，影响拟态表决的准确性。

针对第一个挑战，本文基于不同的表决算法在不同场景下表决准确性和表决速率各不相同的特点，通过策略性选择表决算法来适配不同长度及数据类型的表决块。为此，本文提出了表决算法选取策略集构建方法，给出了不完整数据的表决准确性解决方案。

针对第二个挑战，本文基于存贮理论对分块报文的表决过程进行建模分析，并基于此模型协同拟态表决速率、拟态表决准确性和内存占用量，提出表决窗口控制算法来控制自适应拟态表决器的表决块切分长度。

基于以上分析，本文给出了自适应拟态表决器架构，如图 4 所示。当拟态表决器从执行体侧的持久性连接中接收到异构冗余执行体源源不断输出的分块报文时，首先，按连接分类存储，统计分块报文的关键特征。在拟态表决之前，自适应表决窗口控制模块首先分析可表决的数据，记录包括每一个分块报文的长度、到达时刻和数据类型的流特征，根据数据类型 ς 和分块报文传输吞吐量 V（定义为分块报文的平均长度与平均到达时间间隔的比值）选取局部最优的表决窗口 ξ，并根据分块报文长度 l 和数据类型 ς 通过表决算法选取策略集 G 选取最优的表决算法 λ；其次基于存贮模型评估该表决窗口下的存贮成本，通过优化存贮成本自适应调整得到当前最佳的表决窗口；最后以当前最佳表决窗口为依据对存储的异构冗余执行体输出的分块报文做切分处理。在此过程中，自适应表决窗口控制模块建立流特征与最佳表决窗口的映射表 I，避免同一流特征的分块报文反复搜索最佳表决窗口。然后，表决输出模块根据选取的表决算法对表决块进行拟态表决处理，将不满足要求的执行体的信息记录或上报，并将通过表决的表决块数据通过客户端侧持久性连接源源不断地输出。

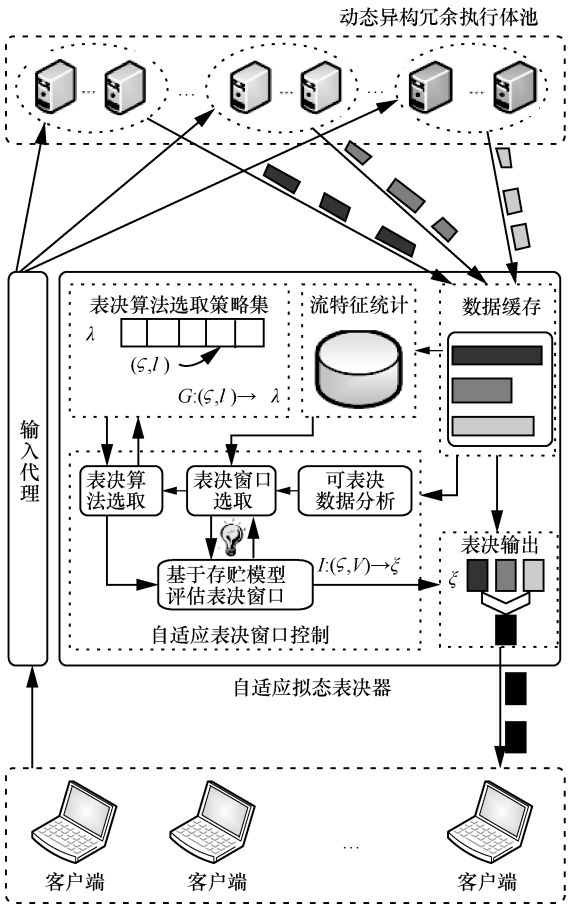


图 4　自适应拟态表决器架构

为详细说明以上工作流程，假设执行体集合为 $E=\{e_1,e_2,\cdots,e_n\}$，执行体 $e_i \in E$ 在 $\{t_{i,1},t_{i,2},\cdots,t_{i,m^i}\}$ 时刻分别输出长度为 $\{l_{i,1},l_{i,2},\cdots,l_{i,m^i}\}$ 的分块报文 $\{r_{i,1},r_{i,2},\cdots,r_{i,m^i}\}$。自适应拟态表决算法如算法 1 所

示。步骤 1)和步骤 2)为初始化，包括初始化流特征统计集 $\{(t_{i,k},l_{i,k})\}$、执行体 $\forall e_i \in E$ 的已存储数据长度 l_i 和存储队列 S_i、当前可参与表决的数据长度 R_i。步骤 3)~步骤 16)的循环语句为处理持续到达的分块报文。步骤 4)以三元组的形式记录分块报文 $r_{i,k}$ 的流特征，包括到达时刻 $t_{i,k}$、长度 $l_{i,k}$ 和数据类型 ς。步骤 5)存储分块报文并计算已存储执行体 $e_i \in E$ 的数据长度及当前可参与表决块切分的数据长度 R_i。步骤 6)粗略分析可表决数据，若满足条件则通过步骤 7)的 VoteWindowControl() 自适应表决窗口控制函数获取当前最佳表决窗口 ξ 和表决算法 λ_φ，进而由步骤 8)和步骤 9)的表决输出模块对待表决分块报文做表决块切分、表决和输出。接着，步骤 10)~步骤 11)释放已表决输出的分块报文所占用的内存空间并更新系统状态。最后，由步骤 13)~步骤 15)判断持久性连接传输分块报文的结束条件并在满足结束条件时终止表决过程。

算法 1 自适应拟态表决算法

输入 接收分块报文 $r_{i,k},\forall i\in[1,n],\forall k\in[1,m^i]$

输出 表决结果 Out，流特征集 $\{<t_{i,k},l_{i,k},\varsigma>\}$

1) 初始化流特征集 $\{<t_{i,k},l_{i,k},\varsigma>\}$

2) $\forall i\in[1,n],\text{DataInit}(S_i);l_i=0;R_i=0$

3) while 分块报文 $r_{i,k}$ 输入 do

4) $\{<t_{i,k},l_{i,k},\varsigma>\}.\text{append}(r_{i,k}.t,r_{i,k}.l,r_{i,k}.\varsigma)$

5) $l_i+=l_{ik};R_i\leftarrow l_i-d;\text{DataIn}(S_i,r_{i,k})$

6) if 分块报文可以被表决 then

7) $<\xi,\lambda_\varphi>\leftarrow$ VoteWindowControl()

8) $\text{Out}=\text{Vote}(\lambda_\varphi,\text{DataOut}(S_i,\xi,\forall i\in[1,n]))$

9) 发送表决结果 Out 至客户端

10) $\forall i\in[1,n]$ 数据队列 S_i 释放已表决数据

11) $R_i-=\xi$

12) end if

13) if $\forall i\in[1,n],S_i==\text{Empty}$ or 断连接 then

14) 结束表决进程

15) end if

16)end while

如图 5 所示，本文以 2 个执行体利用持久性连接输出分块报文的场景为例，直观分析算法 1 的有效性。自适应表决器在 t_1 时刻收到 2 个执行体的第一个分块报文时，就开始对可表决数据段 1 进行自适应表决和转发处理，之后释放表决器中存储的数据段 1。随后在 t_2、t_3、t_4 时刻，表决器根据算法 1 陆续处理数据段 2、3、4，并且在 t_5 时刻处理完成并释放所有数据。而现有的采用 CAC 机制的拟态表决算法需要在 t_4 时刻才能接收到 2 个执行体的完整数据，并通过一次性对所有数据进行表决处理才能在 t_6 时刻处理完毕。

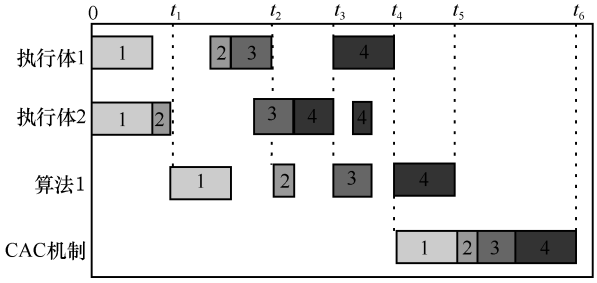


图 5 自适应表决算法数据管理机制有效性分析

3 表决算法选择策略集构建

为有效组合各类表决算法在不同数据场景下表决准确性和表决速率的优势，本节给出表决算法处理不同长度和不同数据类型的处理效率和准确度评价指标，并以该指标为参考构建分块报文的匹配表决块长度和数据类型的拟态表决算法选择策略集 $G:(\varsigma_h,l)\rightarrow\lambda_\varphi$，为自适应拟态表决器工作场景提供不完整数据的表决准确性解决方案。

由于异构服务执行体输出的报文存在固有的格式差异，拟态表决算法针对不同数据类型及不同数据长度的报文表决准确性存在差异。例如，图片数据一般只能通过 CRC 或 MD5 特征值比较方法，较长的 HTML 数据可以采用语义特征比较方法，而较短的 HTML 数据只能采用字符串编辑距离等相似度比较方法。为量化评估拟态表决准确度，本文基于混淆矩阵给出拟态表决算法的准确性评估方法。由于网络攻击行为难以穷举，拟态表决器通过表决检测攻击的准确性难以直接评估，即混淆矩阵中的真阳性（TP, true positive）和假阳性（FP, false positive）等指标难以直接获取。因此，本文通过统计无攻击条件下拟态表决的虚警概率来评价拟态表决的准确度，即通过线下实验统计拟态表决算法在处理不同长度的各类数据时表现出的真阴性（TN, true negative）和假阴性（FN, false negative）的次数。假设持久性连接中传输的分块报文的数据类型集为 $Z=\{\varsigma_1,\varsigma_2,\cdots,\varsigma_H\}$，可供选择的拟态表决算法集为 $\Lambda=\{\lambda_1,\lambda_2,\cdots,\lambda_\Phi\}$，算法 $\lambda_\varphi\in\Lambda$ 对数据类型为 $\varsigma_h\in Z$、长度为 l 的表决块的表决准确度可表示为

$$p(\lambda_\varphi,\varsigma_h,l)=\frac{\mathrm{TP}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}}{\mathrm{TP}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{FP}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{FN}_{\lambda_\varphi,\varsigma_h,l}}\approx \frac{\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}}{\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{FN}_{\lambda_\varphi,\varsigma_h,l}} \tag{1}$$

由于表决块的长度 l 为连续变量，因此式(1)的策略集构建过程无法在有限次的线下实验中完成。为简化处理，本文按照等长区间 2σ 对数据长度进行分区间处理，因此长度在区间 $[x-\sigma,x+\sigma]$ 内的 $\varsigma_h\in Z$ 类型数据在算法 $\lambda_\varphi\in\Lambda$ 处理下的表决准确度可表示为

$$p(\lambda_\varphi,\varsigma_h,x,\sigma)=\frac{1}{N}\sum_{l\in[x-\sigma,x+\sigma]}\frac{\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}}{\mathrm{TN}_{\lambda_\varphi,\varsigma_h,l}+\mathrm{FN}_{\lambda_\varphi,\varsigma_h,l}} \tag{2}$$

其中， N 为表决块长度在区间 $[x-\sigma,x+\sigma]$ 的样本个数。

假设算法 $\lambda_\varphi\in\Lambda$ 处理单位长度的数据类型为 $\varsigma_h\in Z$ 的待表决数据的时间消耗为 $\mu(\lambda_\varphi,\varsigma_h)$，可通过线下实验测试表决算法的响应时延求解线性方程得出。表决处理时间与当前选定表决算法的准确性呈负相关，即采用准确度较低的拟态表决算法进行表决处理需要消耗更多的时间来补偿准确性损失，因此，本文记 n 个服务执行体的数据类型为 $\varsigma_h\in Z$ 、长度为 l （ $l\in[x-\sigma,x+\sigma]$ ）的表决块经算法 $\lambda_\varphi\in\Lambda$ 表决处理的时间消耗为

$$\tau(\lambda_\varphi,\varsigma_h,l)=\mathrm{C}_n^2 l\mu(\lambda_\varphi,\varsigma_h)\frac{1}{p(\lambda_\varphi,\varsigma_h,x,\sigma)} \tag{3}$$

其中，组合数 C_n^2 的物理含义是 n 个服务执行体的输出数据在拟态表决过程中两两相互比较，该方法是基于大数判决的一般方法[1]；表决准确度的倒数的物理含义是表决时间与表决准确度呈负相关。当可选的表决算法的准确性均较低时，拟态表决器需要更多的时间来保证表决的可靠性，即拟态表决器需要多次重复或交叉表决才能得到可靠的结果。

基于该定义，本节将算法的准确性判断融入算法执行效率的排序中，以简化工程实现中对表决准确性这一因素的处理。因此，本节的目标转化为给数据类型为 $\varsigma_h\in Z$ 、长度为 l 的表决块选取对应的拟态表决算法 $\lambda_\varphi\in\Lambda$，使 $\tau(\lambda_\varphi,\varsigma_h,l)$ 最小，从而添加键值对 $(\varsigma_h,l)\to\lambda_\varphi$ 至表决算法选取策略集 G 中。就目前的拟态表决算法的发展现状而言，可选的表决算法集 Λ 和待处理数据类型集 Z 的维度均较小，该优化目标函数的搜索空间较小，通过线下实验测试的数据类型、表决长度与表决算法的映射表，可得到自适应拟态表决器工作场景下表决算法选取策略集 G 。

4　自适应表决窗口控制

为了确定持久性连接中分块报文拟态表决场景下表决块的最佳切分方案，本节基于存贮模型对分块报文的拟态表决过程进行建模分析，给出分块报文的拟态表决在该场景下的成本评估方法，并提出基于存贮模型的自适应表决窗口（IMAVW, inventory model based adaptable voting window）控制策略。

4.1　存贮模型建立

存贮模型是基于存贮理论[16]的运筹学理论模型，是仓储管理领域的经典分析模型[20]。本文将拟态表决器的收发分块报文建模为仓库转运问题：拟态表决器从持久性连接中接收异构冗余执行体输出的分块报文为进货过程；表决结束后向客户端发送表决通过的表决块数据为出货过程；存储持久性连接中输出的分块报文所占用的内存空间为存贮量；拟态表决器处理一个持久性连接中所有分块报文的全部数据的总时长 T 为一个表决事务的存贮周期；系统的存储资源占用量的成本为存贮成本，本文将存贮成本定义为内存空间占用量与占用时间的乘积。

记 $l_i(t)$ 表示 t 时刻执行体 $e_i\in E$ 在拟态表决器中存储的数据量， $d(t)$ 表示 t 时刻拟态表决器已经完成表决且输出的数据量，那么在一次持久性连接的拟态表决的事务中的存贮成本为

$$C_Q=\int_0^T\sum_{i\in[1,n]}[l_i(t)-d(t)]\mathrm{d}t \tag{4}$$

该积分项无法直接求解，本文基于递归方法给出可计算的表达式。假设以 ξ 为步长并将其作为表决窗口，记 $l_{i,k}$ 进入表决器时是否具备参与表决条件的指示函数为

$$\phi(i,k)=\begin{cases}1,\forall i\in[1,n],R_i\geqslant\xi\\0,\exists i\in[1,n],R_i<\xi\end{cases} \tag{5}$$

其中， R_i 表示执行体 $e_i\in E$ 当前可参与拟态表决处理的分块报文的数据长度，即已接收数据的长度与已表决输出数据的长度之差。 R_i 的初始值为 0，其在迭代计算过程中的变化包括非表决时刻的累加

$l_{i,k}$ 和表决时刻减少至整除 ξ 的余数，即

$$R_i(t_{i,k+1})=\begin{cases}R_i(t_{i,k})+l_{i,k},\forall i\in[1,n],\forall k\in[1,m^i]\\ R_i(t_{i,k})\bmod\xi,\phi(t_{i,k})=1\end{cases} \tag{6}$$

为了给出一个持久性连接传输分块报文的全时段分析，将所有分块报文到达表决器的时刻值按序排列，记为 $\Gamma=\{t_1,t_2,\cdots,t_{i_1,k_1},t_{i_2,k_2},\cdots\}$，其中 $t_{i_1,k_1}<t_{i_2,k_2}(\forall i_1,i_2\in[1,n],\forall k_1\in[1,m^{i_1}],\forall k_2\in[1,m^{i_2}])$。存贮成本可分为表决等待期间（VWP, vote wait period）的存贮成本和表决忙碌期间（VBP, vote busy period）的存贮成本两部分。一个 VWP 的时间间隔可表示为

$$t_w=\left|t_{i_2,k_2}-t_{i_1,k_1}\right| \tag{7}$$

其中，$\phi(t_{i_1,k_1})=\phi(t_{i_2,k_2})=1$，$t_{i_1,k_1}$ 和 t_{i_2,k_2} 为相邻的 2 个可表决时刻，即 $\forall t_{i,k}\in(t_{i_1,k_1},t_{i_2,k_2})$，$\phi(t_{i,k})=0$，因此一个 VWP 的存贮成本可表示为

$$C_w=\sum_{i\in[1,n]}\sum_{t_{i_1,k_1}<t_{i,k}<t_{i_2,k_2}}(t_{i,k+1}-t_{i,k})R_i(t_{i,k}) \tag{8}$$

记一个 VWP 从 t_{i_2,k_2} 开始，表决器以长度为 ξ 的表决块为单位，经过 $\Omega=\left\lfloor\dfrac{\min\limits_{i\in[1,n]}\left\{R_i\left(t_{i_2,k_2}\right)\right\}}{\xi}\right\rfloor$ 次的表决处理完成一个 VBP，其中表决处理的时间消耗可由式(3)计算得出，因此，每个 VBP 的结束时刻可表示为

$$t_v=t_{i_2,k_2}+\Omega\tau(\lambda_\varphi^{\xi},\varsigma_h^{\xi},\xi) \tag{9}$$

该表决期间的存贮成本可表示为

$$C_v=\sum_{i\in[1,n]}\sum_{r\in[1,\Omega]}\left(R_i(t_{i_2,k_2})-r\xi\right)\tau(\lambda_\varphi^{\xi},\varsigma_h^{\xi},\xi) \tag{10}$$

注意到，在一个处理线程中，存在上一个 VBP 的结束时刻 t_v 可能会晚于下一个 VBP 的开始时刻 $t_{i_2',k_2'}$ 的情况，即上一个 VBP 消耗时间过长。当该情况出现时，下一个 VBP 需要等待前一个 VBP 结束才能开始，以免抢占。因此，补充式(6)中表决条件指示函数的修正条件 $\phi(i,k)=0,\forall t_{i,k}\leqslant t_v$。

基于以上讨论，存贮成本可表示为

$$C_Q=\sum_{t_{i,k}\leqslant\max\limits_{i\in[1,n]}}C_w+\sum_{t_{i_2,k_2}\leqslant\max\limits_{i\in[1,n]}}C_v \tag{11}$$

以上求解存贮成本算法的具体实现逻辑如算法 2 所示。其中，步骤 2)和步骤 3)为初始化过程；步骤 4)~步骤 20)的循环逻辑根据所有执行体的所有分块报文的到达时刻和数据长度计算存贮成本。该循环逻辑按照时间顺序在每一个分块报文到达时刻开始处理，包括每一个时间间隔的等待存贮成本 C_w 计算（步骤 6)~步骤 8)）、根据当前时刻与上一个 VBP 的结束时刻的关系更新每一个分块报文到达时的指示函数 $\phi(i,k)$（步骤 9)~步骤 11)）、依据当前各个执行体的分块数据的存储状态计算表决存贮成本（步骤 12)~步骤 16)）以及最后更新当前时刻与当前 VBP 结束时刻的关系，以准备下一分块数据的表决处理（步骤 17)~步骤 19)）。

算法 2 计算存贮成本算法

输入 $\{(t_{i,k},l_{i,k})\}_{\varsigma_h},i\in[1,n],k\in[1,m^i],\varsigma_h\in Z,\xi$

输出 C_Q

1)function CalCost$\left(\{(t_{i,k},l_{i,k})\},\xi\right)$

2) 初始化 $C_Q=0;t_{\text{prev}}=0;R_i=0,\phi(i,k)=0$

3) 添加 $t_{i,k}$ 到 Γ，$t_{\text{curr}}=t_{i,k}$

4) for each $t_{i,k}$ in Γ do

5) $t_{\text{curr}}=t_{i,k};R_i+=l_{i,k};$

6) if Γ 中元素数量多于 2 个 do

7) $C_w=\sum\limits_{i\in[1,n]}R_i(t_{\text{curr}}-t_{\text{prev}})$//根据式(8)

8) end if

9) if $t_{\text{curr}}<t_{\text{prev}}$ do

10) $\phi(i,k)=0;$ continue;

11) end if

12) if $\forall i\in[1,n],R_i\geqslant\xi$ do

13) $i_2=i,k_2=k;\phi(i_2,k_2)=1;$

14) 根据式(9)和式(10)分别计算 t_v 和 C_v

15) $R_i=R_i\bmod\xi;$

16) else

17) $t_v=t_{i,k};\phi(i,k)=0;$

18) end if

19) $t_{\text{prev}}=t_v;C_Q+=C_w+C_v;$

20) end for

21) return C_Q

22)end function

4.2 基于存贮模型的自适应表决窗口控制策略

获取最优表决窗口。即搜索使 C_Q 最小的最佳表决窗口 ξ^*。

$$\xi^*=\min_{\xi}\left\{C_Q\right\},0<\xi<\sum_{k\in[1,m^i]}l_{i,k} \tag{12}$$

由于式(12)中优化问题的主要约束条件与分块报文的特征有关，包括持久性连接中分块报文的数据类型ς_h和传输吞吐量V，在实际系统的运行中均需要通过流特征统计模块不断丰富。因此，本文基于哈希表构建最佳表决窗口的策略集$I:(\varsigma_h,V)\to\xi$，即给出流特征与最优表决窗口的哈希映射表，并且随着应用过程中流特征的丰富而不断更新该策略集。

策略集更新的主要思路是根据接收到的分块报文的数据类型ς_h和传输吞吐量V选择或计算对应的最佳表决窗口，若最佳表决窗口策略集I中包含该特征，则直接选取ξ；若不包含该特征，则通过搜索求解该特征下的最佳表决窗口并存入最佳表决步长策略集。为减少寻优的搜索次数，本文使用模拟退火算法求解最佳表决窗口。基于以上分析，本文提出基于存贮模型的自适应表决窗口控制策略，具体过程如算法 3 所示。其中，步骤 1)给出计算平均流传输速率的计算方法，步骤 2)搜索最佳表决窗口策略集中是否存在该流特征的历史记录：若存在则直接得到最佳表决窗口（步骤 3)）；否则通过模拟退火算法求取该流特征下的最佳表决窗口并且存入最佳表决窗口策略集I中（步骤 5)~步骤 15)）。

算法 3　基于存贮模型的自适应表决窗口控制算法

输入　$\{(t_{ik},l_{ik})\}_{\varsigma_h},i\in[1,n],k\in[1,m^i],\varsigma_h\in Z$

输出　ξ^*

1) $V=\dfrac{\underset{i\in[1,n]}{\text{mean}}\left(\sum\limits_{k\in[1,m^i]}l_{ik}\right)}{\underset{i\in[1,n]}{\text{mean}}\left(\sum\limits_{k\in[1,m^i]}t_{ik}\right)}$

2) if　find　(ς_h,V)　in　I　do

3)　$\xi^*\leftarrow I(\varsigma_h,V)$

4) else

5)　在 0 和 $\sum\limits_{k\in[1,m^i]}l_{ik}$ 之间初始化ξ值

6)　$T=100,\lambda=0.05$

7)　while　$T>0.001$　do

8)　$\xi'\leftarrow\xi+\dfrac{T}{100}\underset{i\in[1,n]}{\text{mean}}\left(\sum\limits_{k\in[1,m^i]}l_{ik}\right)\text{rand}(-1,1)$

9)　$\Delta E=\text{CalCost}\left(\left\{(t_{ik},l_{ik})\right\},\xi'\right)-\text{CalCost}\left(\left\{(t_{ik},l_{ik})\right\},\xi\right)$

10)　if　$\Delta E<0$ or $\exp\left(-\dfrac{\Delta\text{E}}{T}\right)>\text{rand}(0,1)$ then

11)　$\xi\leftarrow\xi'$

12)　end if

13)　$T\leftarrow T-\lambda T$

14)　end while

15)　$\xi^*=\xi,I(\varsigma_h,V)\leftarrow\xi$

16)end if

4.3　算法仿真分析

为了分析存贮模型下的关键参数，本节通过仿真实验评估前文所述的分块报文的传输吞吐量和数据类型对自适应拟态表决器的存贮成本的影响。仿真实验假设分块报文输出服从泊松分布，每个分块报文长度正比于输出时间间隔，其比值即分块报文传输吞吐量。为对比 IMAVW 控制策略在表决器中的表现，本节同时分析现有表决器常用的 CAC 机制和根据当前已有分块报文，最大限度切分表决块的“尽力而为”表决窗口（BVW, best-effort voting window）控制策略。其中，BVW 控制策略为功能弱化的自适应表决窗口控制策略，其在接收到分块报文时检查数据存储模块中异构冗余执行体的待表决数据，若所有执行体都存在待表决数据，则以待表决数据最少的执行体的长度作为表决窗口对所有执行体的待表决数据做表决块切分，使所有能参与表决的数据都参与表决。

持久性连接中分块报文的传输吞吐量是执行体服务性能的重要指标，本节定义该指标为单位时间传输的数据长度。仿真实验假设分块报文输出服从泊松分布，每个分块报文长度正比于输出时间间隔，其比例系数即分块报文传输吞吐量。存贮成本与分块报文的传输吞吐量的关系曲线如图 6 所示。由图 6 可知，随着分块报文的传输吞吐量的增大，表决器的存贮成本增大。IMAVW 策略明显优于 BVW 策略和 CAC 机制，且随着分块报文的传输吞吐量的增大，其优势愈发明显。

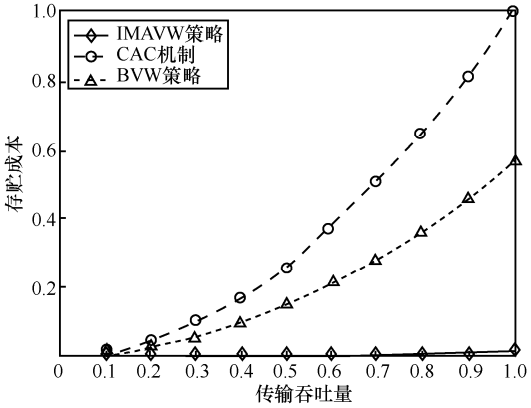


图 6　存贮成本与分块报文的传输吞吐量的关系曲线

分块报文的数据类型是影响分块报文拟态表决准确度的重要原因。基于式(3)的定义，分块报文的不同数据类型的最优表决算法的表决准确度间接影响表决算法的运行速率，进而对存贮成本产生影响。因此，本文在仿真实验中通过设置最大表决准确度的变化，分析分块报文的数据类型对存贮成本的影响，结果如图 7 所示。从图 7 可以看到，随着表决准确度的上升，表决器的存贮成本不断下降。IMAVW 策略明显优于 BVW 策略和 CAC 机制，且在表决准确度较低的情况下，其优势更加明显。这间接说明了 IMAVW 策略可有效缓解某些数据类型的分块报文在切分表决中表决准确度低的问题。

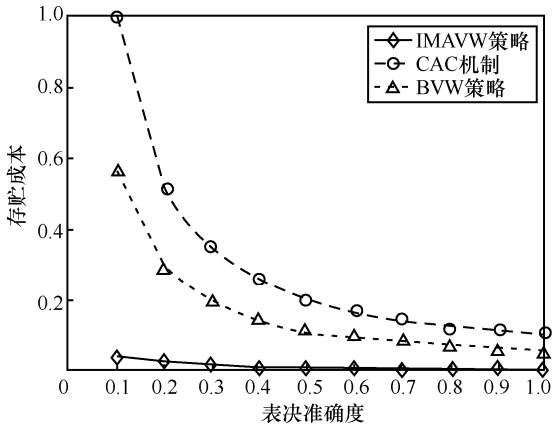


图 7 分块报文的数据类型对存贮成本的影响

5 系统评估

5.1 系统实现

本文基于 Nginx 实现自适应拟态表决器的原型系统。首先，本文基于 Nginx 的子请求处理模块开发 HTTP 请求分发模块，复制客户端发出的 HTTP 请求报文，并且在反向代理的业务逻辑中将复制的 HTTP 请求转发给后端异构冗余执行体。其次，针对异构冗余执行体通过持久性连接输出的 HTTP 响应的分块报文，本文在输出过滤模块回调链中利用多级存储队列来存储持续到达的分块报文。再次，本文实现待表决数据的状态判断函数和自适应表决窗口选取函数，并基于选定的表决窗口，通过指针偏移的方式实现零复制操作的表决块切分。最后，本文根据调用选定的表决算法的实现函数完成对表决块的表决，并将表决选出的表决块通过 Nginx 的过滤回调链表发送至下游连接写操作接口，进而发送至客户端。此外，为了保证基于 Nginx 的拟态表决器在持久性连接场景下的稳定性，本文对 Nginx 的连接管理机制做了部分改动。

5.2 实验配置

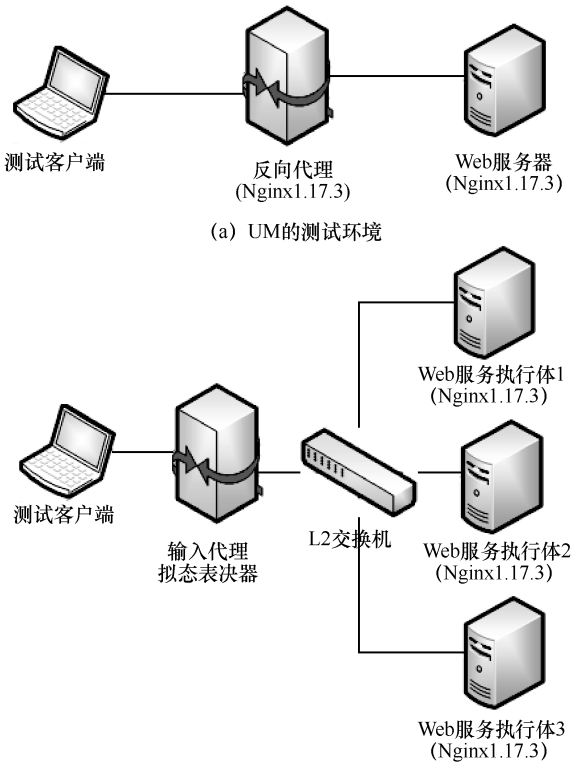
由于拟态表决器的网络位置与反向代理服务器相同，本文选取非拟态（UM, unused mimic）Web 服务的反向代理作为自适应拟态表决器的实验性能参考基准。在同等资源配置下，具有安全增益的拟态系统的性能越接近非拟态系统则越优。其次，CAC 机制是当前拟态表决算法常用的数据管理机制，用来评估自适应拟态表决器相较于拟态表决器的表决效率是否有所提升。此外，BVW 策略是不含基于存贮模型优化表决窗口的弱化策略，用来评估 IMAVW 策略的有效性。为方便后文表述，本节将采用 CAC 机制的传统拟态表决器、采用 BVW 策略和 IMAVW 策略的自适应拟态表决器分别简述为 CAC、BVW 和 IMAVW。

自适应拟态表决器的可选表决算法集包括基于语义特征（SF, semantic feature）抽取[5]、基于字符串相似度（SS, string similarity）表决[6]、基于层次分析法（AHP, analytic hierarchy process）的多维表决[10]。BVW 与 IMAVW 根据表决算法选择策略集在每个表决块的表决中动态选择表决算法。CAC 在表决效率与资源开销实验中选择复杂度最低的 SS 算法，而在表决准确性评估实验中分别选择 SF、SS 与 AHP 算法。

本文基于 6 台配置如表 1 所示的虚拟机搭建实验环境，其实验拓扑及环境组件如图 8 所示。图 8(a)包含反向代理服务器和 Web 服务器，为 UM 的测试环境。图 8(b)包含具有输入代理和拟态表决器以及冗余的 Web 服务器，为 CAC、BVW、IMAVW 的测试环境。各个 Web 服务器均由 Nginx 搭建，配置其 Keep-alive 模式，以持久性连接传输分块报文的方式输出 Web 页面资源。

表 1 实验主机配置

指标	参数
操作系统	CentOS 7.6 64 位
CPU	8 核 2.5 GHz
内存/GB	4
硬盘/GB	32
网络带宽/(Mbit·s^{-1})	1 000



(a) UM的测试环境

(b) CAC、BVW、IMAVW的测试环境

图 8　实验拓扑及环境组件

5.3　内存资源占用量评估

本文提出的自适应拟态表决器通过对持久性连接中的分块报文动态表决输出，在表决的过程中动态释放已表决的数据，从而降低表决器的内存占用量。为了验证自适应表决器是否可有效降低内存占用量，本节通过采集持久性连接传输分块报文过程中的拟态表决器空闲物理内存量，分析比较各实验组的内存资源占用情况。

本节实验配置执行体传输 20 MB 的页面资源，并且在拟态表决器或反向代理服务器中运行基于 Linux 系统的 free 命令的测试脚本，记录它们的空闲物理内存变化情况，如图 9 所示。从图 9 可以看到，相较于传统表决器（CAC），自适应拟态表决器（BVW、IMAVW）占用的内存较少且占用时间较短。具体来讲，UM 占用内存的极值为 8 MB，工作持续时间为 3.35 s，而 CAC 占用内存的极值为 70 MB，工作持续时间为 6.3 s，远逊于 UM 的工作性能。BVW 占用内存的极值为 39 MB，工作持续时间为 3.5 s，显著降低了拟态表决对服务性能的影响。本文提出的 IMAVW 占用内存的极值为 36 MB，工作持续时间为 3.4 s，进一步优化了拟态表决器的内存开销和响应时延。

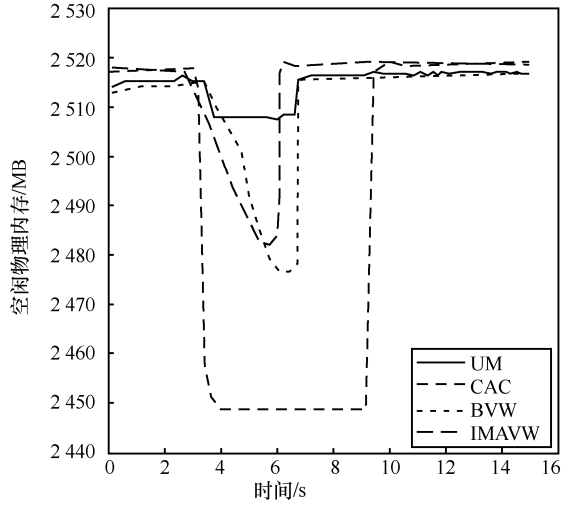


图 9　内存资源占用量对比

5.4　传输时延评估

为了验证本文提出的自适应表决器通过最优表决窗口控制能否有效降低表决器的表决处理时间，本节实验通过在持久性连接中传输不同数据规模的分块报文，测试分析各实验组的响应时延差异。本节实验配置 Web 服务执行体分别传输总长度为 10 MB、20 MB、40 MB、80 MB、160 MB、320 MB 的页面资源，在测试客户端上分别请求这些页面资源并记录响应时延，实验结果如图 10 所示。在 Web 服务执行体任一数据规模的响应资源中，UM 响应时延最小，CAC 最大，BVW 次之，IMAVW 第三。此外，随着响应资源的数据规模的增大，BVW 和 IMAVW 相对于 CAC 的性能优化存在增强的效果。这说明本文提出的自适应拟态表决器在持久性连接中传输不同的网页资源的场景下均具有降低响应时延的效果。

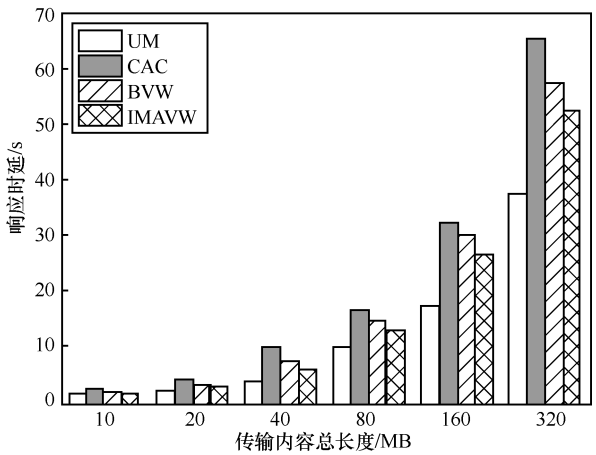


图 10　持久性连接内传输内容总长度对系统响应时延的影响

5.5　并发性能评估

为了验证自适应拟态表决器在高并发场景下是否具有较好的响应时延表现和降低内存占用量之后是否会提高表决器的吞吐量，本节实验测试了各实验组在不同并发连接下的性能表现。在测试客户端上，使用 Apache Benchmark 测试工具测试拟态表决器在不同并发压力下的响应时延和吞吐量。在Web服务器配置64 B的较小页面资源，使用Apache Benchmark 分别以并发量为 1 000、2 000、3 000、4 000、5 000 给 Web 服务系统发送 100 000 个请求报文。在此过程中，拟态表决器依次配置 CAC、BVW、IMAVW。根据 Apache Benchmark 生成的测试报告，响应时延和吞吐量的变化趋势分别如图 11 和图 12 所示，其中吞吐量的单位为每秒查询率（QPS, query-per-second）。

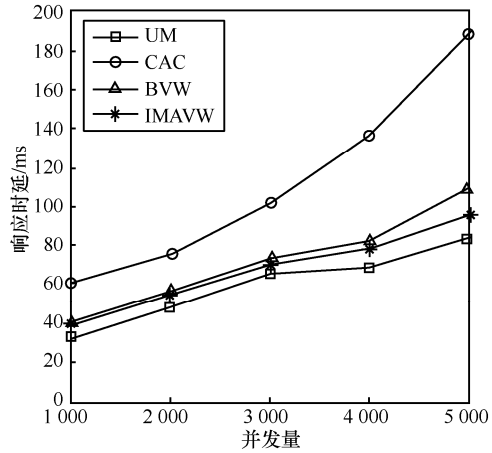


图 11　连接并发量对系统响应时延的影响

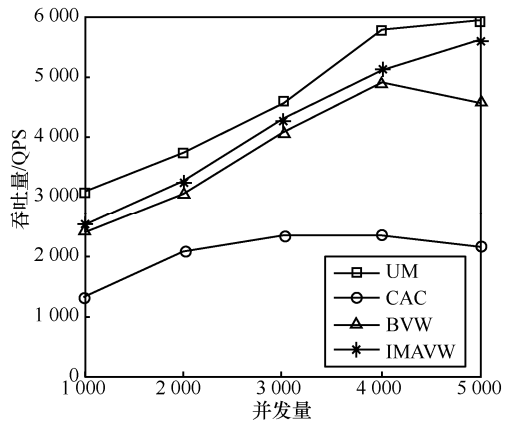


图 12　连接并发量对系统吞吐量的影响

由图 11 可知，随着并发量的增大，响应时延呈上升趋势。CAC 相对于 UM 的响应时延较长，且随着并发量的增大而逐渐加剧，而 BVW 和 IMAVW 接近 UM 的响应时延；通过原始数据计算可得 UM、CAC、BVW、IMAVW 的所有实验组的平均响应时延分别为 59.8 ms、112.8 ms、72.5 ms、67.9 ms，说明 BVW 和 IMAVW 在不同的服务压力下均可大幅降低拟态系统的响应时延，且本文提出的 IMAVW 效果最好。

由图 12 可知，随着并发量的增大，吞吐量的总体趋势呈先增大后减小趋势，CAC 的吞吐量曲线在并发量为 3 000 时出现拐点，BVW 在并发量为 4 000 时出现拐点，而 IMAVW 在并发量为 5 000 时尚未出现明显拐点，说明自适应拟态表决器的并发能力明显优于现有表决器，且本文提出的 IMAVW 的并发性能优于BVW。通过原始数据计算可得UM、CAC、BVW、IMAVW 的所有实验组的平均吞吐量分别为 4 628.8、2 040.6、3 799.9、4 141.1，说明 BVW 和 IMAVW 在不同的服务压力下对拟态表决器的吞吐量均有较好的改善效果，且 IMAVW 改善效果最好。

5.6　表决准确性评估

为了验证自适应拟态表决器在降低开销、提高表决效率的同时能否维持可接受的表决准确度，本节实验通过与 CAC+SF、CAC+SS、CAC+AHP 比较评估自适应表决器的表决准确性。

本节实验配置 HTML 文件作为执行体传输的 Web 页面，其原因是 HTML 格式的数据构成的网页框架具有多标签和文本内容动态生成的特点，可不失一般性地代表拟态表决器在生产环境下处理的数据。通过脚本程序分别或组合修改 HTML 文件的文本内容、标签顺序，各个 Web 执行体的修改相互独立，来模拟拟态表决器对报文做比较的表决处理过程。其中，文本内容修改用来模拟恶意篡改内容的行为，标签顺序修改用来模拟动态网页生成时产生的正常的内容差异。因此在理想状态下，文本内容不一致会被判定为非法，而标签顺序不一致则被判定为合法。各种算法的判决阈值为 90%，即两执行体的数据相似度低于 90%认为是不一致。本节实验通过多次随机修改重复实验来计算表决算法的准确度，其中合法报文的表决准确度基于式(1)计算 TN 出现的比例，非法报文的表决准确度由 TP 出现的比例计算可得。

表决准确度分析如图 13 所示，CAC+SF 算法、CAC+SS 算法分别在标签顺序修改、文本内容修改的实验中表现最好，CAC+AHP 算法则整体表现最优，IMAVW 整体次优，BVW 整体最差。分析可知，CAC+SF 算法通过解析文本特征可识别顺序颠倒的标签，且应对内容修改的表决也具有较高的准确率；

CAC+SS 算法通过字符比较可准确判断文本内容修改，但是容易将标签顺序修改判定为不一致，因此对于标签顺序修改和多点组合修改表决准确度较差；CAC+AHP 算法通过综合分析多个指标可以有效应对各种类型的数据变化，因此整体的表现最优，表决准确率达 94.64%。自适应拟态表决器由于对整体数据进行切分表决，存在将一处文本内容修改或标签顺序修改切分到 2 个表决块的概率，会造成表决算法无法判断的问题，因此 IMAVW 整体相对 CAC+AHP 算法表决准确率降低 8.41%；BVW 由于对报文切分具有盲目性，其表决准确率降低更多，达到 34.84%。此外，由于本文提出的表决算法选择策略集通过动态选择不同的表决算法，可组合不同表决算法的优势，因此在文本内容修改、标签顺序修改和多点组合修改的实验中表决准确率相对 SS 算法平均提高 14.67%。总体而言，本文提出的 IMAVW 相对于传统拟态表决器的表决维持了可接受的准确度。

5.7　结果分析

自适应拟态表决器的设计初衷是解决分块报文表决过程中存储资源过度占用带来的服务性能下降问题，资源开销实验直观地说明了自适应拟态表决器对该问题的改善效果。针对不同数据规模和不同服务压力下的实验，以微基准的方式给出了自适应拟态表决器在一般应用场景下的可行性分析。表决准确度评估实验说明了自适应拟态表决器在提高表决效率的同时维护了可接受的表决准确性。采用 CAC 机制的传统拟态表决器在表决效率的实验中表现最差，这说明当前的拟态表决方法在面对 HTTP 1.1 协议的持久性连接、分块传输编码的应用场景中存在较大问题。本文提出的基于 IMAVW 策略的自适应拟态表决器在面对大规模数据和高并发场景时，表决效率较高且表决准确度在可接受的范围内。

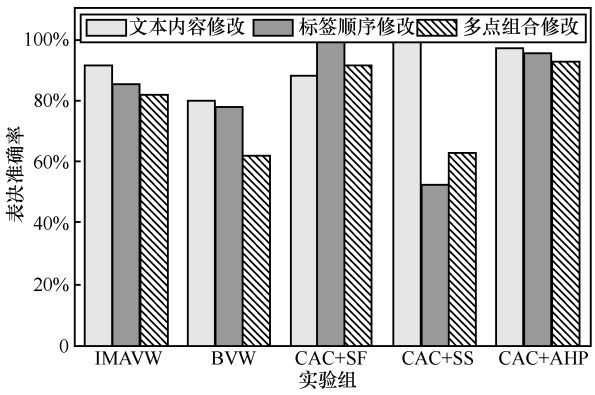


图 13　表决准确度分析

6　结束语

为解决现有拟态表决器在 HTTP 1.1 持久性连接传输分块报文的业务场景下存储大量报文带来的内存资源过度占用及时延增加的问题，本文提出了一种自适应的拟态表决器设计方案，在接收异构冗余执行体输出分块报文的同时，对现存的分块报文数据进行表决块切分和表决输出，在接收分块报文的过程中逐步将已表决的数据释放，从而降低该场景下的内存开销。首先给出自适应拟态表决器的工作机制和实现架构；接着针对自适应拟态表决器实现过程中的表决准确度和表决效率的矛盾，给出自适应拟态表决器工作场景下的表决算法选择策略和基于存贮模型的表决窗口控制策略；最后通过原型系统实现并基于此开展实验分析，说明本文提出的自适应拟态表决器相对于传统表决器具有更好的性能表现，可以在同等资源限制下有效提高分块报文传输场景下拟态系统的性能表现。

今后的工作将围绕自适应拟态表决器的适应性，一方面，开展自适应学习的流特征表决窗口策略集生成方法的研究，实现更方便拟态系统适配的拟态表决器；另一方面，扩展持久性连接的场景，针对 HTTP 2.0 的二进制分帧的持久性连接数据传输场景开展自适应拟态表决器的升级工作。

参考文献：

[1] 邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016, 1(4): 1-10.
WU J X. Research on cyber mimic defense[J]. Journal of Cyber Security, 2016, 1(4): 1-10.

[2] 吴铤, 胡程楠, 陈庆南, 等. 基于执行体划分的防御增强型动态异构冗余架构[J]. 通信学报, 2021, 42(3): 122-134.
WU T, HU C N, CHEN Q N, et al. Defense-enhanced dynamic heterogeneous redundancy architecture based on executor partition[J]. Journal on Communications, 2021, 42(3): 122-134.

[3] MCALLISTER D F, SUN C, VOUK M A. Reliability of voting in fault-tolerant software systems for small output-spaces[J]. IEEE Transactions on Reliability, 1990, 39(5): 524-534.

[4] JAMALI N, SAMMUT C. Majority voting: material classification by tactile sensing using surface texture[J]. IEEE Transactions on Robotics, 2011, 27(3): 508-521.

[5] 仝青, 张铮, 张为华, 等. 拟态防御 Web 服务器设计与实现[J]. 软件学报, 2017, 28(4): 883-897.
TONG Q, ZHANG Z, ZHANG W H, et al. Design and implementation of mimic defense Web server[J]. Journal of Software, 2017, 28(4): 883-897.

[6] 张文建, 宋克, 谭力波, 等. 面向拟态判决的可编程语义解析方法[J]. 通信学报, 2020, 41(4): 62-69.

ZHANG W J, SONG K, TAN L B, et al. Programmable semantic parsing approach for mimic arbitration[J]. Journal on Communications, 2020, 41(4): 62-69.

[7] 马博林, 张铮, 刘健雄. 应用于动态异构 Web 服务器的相似度求解方法[J]. 计算机工程与设计, 2018, 39(1): 282-287.
MA B L, ZHANG Z, LIU J X. Similarity calculation method applied to dynamic heterogeneous Web server system[J]. Computer Engineering and Design, 2018, 39(1): 282-287.

[8] QI C, WU J X, HU H C, et al. An intensive security architecture with multi-controller for SDN[C]//Proceedings of 2016 IEEE Conference on Computer Communications Workshops. Piscataway: IEEE Press, 2016: 401-402.

[9] 欧阳城添, 王曦, 郑剑. 自适应一致表决算法[J]. 计算机科学, 2011, 38(7): 130-133.
OUYANG C T, WANG X, ZHENG J. Adaptive consensus voting algorithm[J]. Computer Science, 2011, 38(7): 130-133.

[10] HU H C, WU J X, WANG Z P, et al. Mimic defense: a designed-in cybersecurity defense framework[J]. IET Information Security, 2018, 12(3): 226-237.

[11] 陆以勤, 黄俊贤, 程喆, 等. 基于改进 AHP-FCE 模型的多指标拟态表决算法[J]. 北京邮电大学学报, 2021, 44(2): 8-13.
LU Y Q, HUANG J X, CHENG Z, et al. A multi-index mimic voting algorithm based on improved AHP-FCE model[J]. Journal of Beijing University of Posts and Telecommunications, 2021, 44(2): 8-13.

[12] ZHOU D C, CHEN H C, CHENG G Z, et al. SecIngress: an API gateway framework to secure cloud applications based on N-variant system[J]. China Communications, 2021, 18(8): 17-34.

[13] 林森杰, 刘勤让, 王孝龙. 面向拟态防御系统的竞赛式仲裁模型[J]. 计算机工程, 2018, 44(4): 193-198.
LIN S J, LIU Q R, WANG X L. Competitive arbitration model for mimic defense system[J]. Computer Engineering, 2018, 44(4): 193-198.

[14] WANG Y W, WU J X, GUO Y F, et al. Scientific workflow execution system based on mimic defense in the cloud environment[J]. Frontiers of Information Technology & Electronic Engineering, 2018, 19(12): 1522-1536.

[15] THÖNES J. Microservices[J]. IEEE Software, 2015, 32(1): 116.

[16] AKHSHABI S, NARAYANASWAMY S, BEGEN A C, et al. An experimental evaluation of rate-adaptive video players over HTTP[J]. Signal Processing: Image Communication, 2012, 27(4): 271-287.

[17] 王禛鹏, 扈红超, 程国振. 一种基于拟态安全防御的 DNS 框架设计[J]. 电子学报, 2017, 45(11): 2705-2714.
WANG Z P, HU H C, CHENG G Z. A DNS architecture based on mimic security defense[J]. Acta Electronica Sinica, 2017, 45(11): 2705-2714.

[18] 张铮, 马博林, 邬江兴. Web 服务器拟态防御原理验证系统测试与分析[J]. 信息安全学报, 2017, 2(1): 13-28.
ZHANG Z, MA B L, WU J X. The test and analysis of prototype of mimic defense in Web servers[J]. Journal of Cyber Security, 2017, 2(1): 13-28.

[19] SCARF H E. Inventory theory[J]. Operations Research, 2002, 50(1): 186-191.

[20] 杨益民, 付必胜. 仓库容量有限条件下的生产销售存贮模型[J]. 系统工程, 2001, 19(1): 18-23.
YANG Y M, FU B S. The storage model about production and sale under the condition of limited-space storehouse[J]. Systems Engineering, 2001, 19(1): 18-23.

[作者简介]

周大成（1995− ），男，河南息县人，信息工程大学博士生，主要研究方向为网络空间安全、云计算等。

陈鸿昶（1964− ），男，河南新密人，博士，信息工程大学教授、博士生导师，主要研究方向为网络空间安全、数据分析等。

程国振（1986− ），男，山东菏泽人，博士，信息工程大学副教授、硕士生导师，主要研究方向为网络空间安全、软件定义网络等。

何威振（1996− ），男，安徽亳州人，信息工程大学博士生，主要研究方向为网络空间安全、云计算等。

商珂（1995− ），女，河南郑州人，信息工程大学助理研究员，主要研究方向为网络空间安全、云计算等。

扈红超（1982− ），男，河南商丘人，博士，信息工程大学教授、博士生导师，主要研究方向为网络空间安全、拟态防御等。