软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(7):1650−1665 [doi: 10.3724/SP.J.1001.2013.04314]

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

存储有效的多模式匹配算法和体系结构*

嵩 天[1], 李冬妮[1], 汪东升[2], 薛一波[2]

[1](北京理工大学 计算机学院 智能信息技术北京市重点实验室,北京 100081)

[2](清华大学 微处理器与片上系统技术研究中心,北京 100084)

通讯作者: 嵩天, E-mail: songtian@bit.edu.cn, http://cs.bit.edu.cn/~songtian

摘 要: 多模式匹配是基于内容检测的网络安全系统的重要功能,同时,它在很多领域具有广泛的应用.实际应用中,高速且性能稳定的大规模模式匹配方法需求迫切,尤其是能够在线实时处理网络包的匹配体系结构.介绍了一种存储有效的高速大规模模式匹配算法及相关体系结构.研究从算法所基于的理论入手,提出了缓存状态机模型,并结合状态机中转换规则分类,提出了交叉转换规则动态生成的匹配算法 ACC(Aho-Corasick-CDFA).该算法通过动态生成转换规则降低了生成状态机的规模,适用于大规模模式集.进一步提出了基于该算法的体系结构设计.采用网络安全系统中真实模式集进行的实验结果表明,该算法相比其他状态机类模式匹配算法,可以进一步减少 80%~95%的状态机规模,存储空间降低 40.7%,存储效率提高近 2 倍,算法单硬件结构实现可以达到 11Gbps 的匹配速度.

关键词: 模式匹配;网络安全;网络入侵检测;有限状态自动机;大规模

中图法分类号: TP393 **文献标识码**: A

中文引用格式: 嵩天,李冬妮,汪东升,薛一波.存储有效的多模式匹配算法和体系结构.软件学报,2013,24(7):1650−1665. http://www.jos.org.cn/1000-9825/4314.htm

英文引用格式: Song T, Li DN, Wang DS, Xue YB. Memory efficient algorithm and architecture for multi-pattern matching. Ruan Jian Xue Bao/Journal of Software, 2013,24(7):1650−1665 (in Chinese). http://www.jos.org.cn/1000-9825/4314.htm

Memory Efficient Algorithm and Architecture for Multi-Pattern Matching

SONG Tian[1], LI Dong-Ni[1], WANG Dong-Sheng[2], XUE Yi-Bo[2]

[1](Beijing Laboratory of Intelligent Information Technology, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

[2](Microprocessor and SOC R&D Center, Tsinghua University, Beijing 100084, China)

Corresponding author: SONG Tian, E-mail: songtian@bit.edu.cn, http://cs.bit.edu.cn/~songtian

Abstract: Pattern matching is the main part of content inspection based network security systems, and it is widely used in many other applications. In practice, pattern matching methods for large scale sets with stable performance are in great demand, especially the architecture for online real-time processing. This paper presents a memory efficient pattern matching algorithm and architecture for a large scale set. This paper first proposes cached deterministic finite automata, namely CDFA, in the view of basic theory. By classifying transitions in pattern DFA, a new algorithm, ACC, based on CDFA is addressed. This algorithm can dynamically generate cross transitions and save most of memory resources, so that it can support large scale pattern set. Further, an architecture based on this method is proposed. Experiments on real pattern sets show that the number of transition rules can be reduced 80%~95% than the current most optimized algorithms. At the same time, it can save 40.7% memory space, nearly 2 times memory efficiency. The corresponding architecture in single chip can achieve about 11Gbps matching performance.

Key words: pattern matching; network security; network intrusion detection; finite state automata; large scale

* 基金项目: 国家自然科学基金(60803002, 61272510, 60833004, 60970002); 国家高技术研究发展计划(863)(2012AA010905); 北京市重点学科建设项目; 北京市自然科学基金(4122069)

收稿时间: 2011-03-17; 修改时间: 2012-04-24; 定稿时间: 2012-08-17

随着网络技术的发展和应用的丰富,人们对网络依赖程度逐渐增加,网络安全问题日益突出.为了防范病毒、恶意代码、入侵、垃圾邮件等安全威胁的传播,需要在传统防火墙进行访问控制的同时增加对网络包有效载荷的检测,即深包检测(deep packet inspection).深包检测技术已成为很多网络安全系统的基础技术,这类系统包括反垃圾邮件系统、反病毒系统、反恶意代码系统、网络入侵检测和防御(NIDS/NIPS)系统、内容过滤系统、统一威胁管理(UTM)系统等.深包检测的提出和应用是网络安全领域的一次重要技术变革.深包检测技术所包含的最重要功能是多模式匹配技术,即根据大量预定义模式对输入内容进行检测.在网络安全应用中,多模式匹配主要是指针对字符串、正则表达式等字符类模式的匹配技术.实际应用中,多模式匹配可以部署于在线检测环境,也可以部署在离线分析环境.

网络安全问题的严重性,使得模式库规模日益增加.2005 年,ClamAV 病毒库的规模在 5 万条左右,到 2012 年 5 月,该病毒库规模已经超过 250 万.其他病毒库,如诺顿、瑞星、卡巴斯基等,规模也超过 100 万.与此同时,网络入侵检测系统 Snort 的入侵库也从 2001 年的 2 000 条增长到 2012 年的 1.8 万条以上.针对大规模模式集的模式匹配算法需求迫切.

大部分基于深包检测的网络安全设备作为网关放置在网络的入口处,如网络入侵防御系统、内容级防火墙、防毒墙、统一威胁管理系统等.由木桶原理可知,网络安全应用中,模式匹配在最差情况下的性能决定了设备的在线工作性能.为此,如何保证模式匹配在最差情况下性能有效,是一个关键问题.

针对多模式匹配,在技术的选择上,应该选取匹配性能稳定的算法.这里所说的稳定包含 3 层含义:匹配速度与被检测的内容无关、匹配速度与预定义模式的内容无关、匹配速度与预定义模式的数量无关.达到匹配性能的稳定,可以有效避免网络安全设备因为攻击而导致性能降低情况的发生,避免通过对安全设备的攻击造成的实质网络攻击,从而实现设备自我保护和网络的保障.

1 概 述

模式匹配问题广泛应用于信息检索、模式识别、入侵检测、内容过滤、病毒代码检测、基因匹配等众多领域,是一个基本的数学问题[1].模式匹配按照模式数目分为单模式匹配和多模式匹配,后者在各领域广泛应用,是本文研究的主要内容.网络安全应用具有实时扫描网络数据的特点,往往要求模式匹配算法能够经过单次扫描找到实时数据中可能的匹配,且高速稳定.

在实际的千兆网络环境中,基于通用处理器的网络安全系统一般采用基于哈希函数(基于跳跃)的多模式匹配方法,这类方法在没有入侵的时候匹配较快,但在大量攻击中性能却较差,网络设备本身很容易受到攻击.在电信级高端网络设备中,芯片级硬件类匹配方法应用广泛,对于这类系统,性能稳定的匹配算法至关重要.

基于状态机的模式匹配算法具备性能稳定的特点,该类算法以 Aho-Corasick 算法(简称 AC 算法)[2]为代表.AC 算法由贝尔实验室的 Aho 和 Corasick 于 1975 年提出,它采用有限状态自动机结构一次接收所有模式串,通过构造状态机来扫描匹配文本.在状态机结构中,无论某个前缀同时属于几个模式,它都有唯一的状态来表示.AC 算法的时间复杂度为 $O(n)$,其中,n 为输入串的长度.

加州大学圣迭戈的 Tuck 等人于 2004 年借鉴 IP 查找技术提出了一种改进的 AC 算法[3],采用节点压缩(bitmap 压缩)和路径压缩技术,在保证最坏性能的前提下大大压缩了算法的存储需求,使得 AC 算法可以适应片上 SRAM 或通用 CPU 缓存的容量配置,从而提高了整体性能.Nathan 等人的实验结果表明,在开源入侵检测系统 Snort 中,对于现有模式集,采用软件实现的 AC 算法所用存储空间可以减少为原来的 5%左右.

IBM 苏黎世研究院的 Lunteren 在 2006 年提出了带优先级的状态机类算法[4],它通过对转换规则进行优先级划分,从而进一步压缩了 AC 算法所需存储空间.尽管该算法具有很好的存储效率,并可以针对正则表达式进行匹配[5],但并没有进一步找到 AC 算法存储空间很大的本质原因.

Smith 等人在 2008 年提出了将每个状态结合一个扩展变量的扩展状态机 XFA[6,7],该方法可以通过定义变量进一步减少多模式产生的状态机规模,给出了一种灵活压缩状态机的办法.然而,该方法由于复杂的状态操作,更适用于软件实现,硬件实现复杂性较高.尽管 Smith 也给出了相关的硬件设计,但由于关键路径相关性很高,

无法进行有效的流水线设计,延长了最短路径时间,从而降低了频率影响整体性能.

Yang 等人提出了针对多正则表达式匹配的时空可切换半确定状态机 SFA[8],该状态机是 DFA 和 NFA 之间的一种结构,由确定数量并行 DFA 组成,但由该方法构造的状态机十分复杂,不适合大规模模式使用.

此外,GPU 作为高速处理引擎,近年来也被引入多模式匹配领域[9,10].由于 GPU 具有高度并行性,且编程容易,已有方法都取得了较好效果.由于网络处理具有较为固定的处理模式,不需要经常改变,定制电路设计相比 GPU,在芯片使用效率和成本上更有优势.

随着对模式匹配性能需求的增加,基于状态机类的模式匹配算法成为高性能模式匹配体系结构设计的基础.但是,这类算法生成的 DFA 规模较大,尤其是对于大规模模式集(10 万以上规模的模式集),所生成状态机规模对存储空间需求很大,既不利于软件算法快速实现,也无法适合硬件实现.为此,国际上相关研究主要围绕如何降低状态机类算法所需的规模和存储空间展开[11−20].应该说,简洁且有效的算法是解决该领域问题的首选,过分追求存储空间最优但造成结构复杂的方法生命力有限.

本文第 2 节对基本的状态机类算法、带优先级的状态机类算法进行回顾,并对这些算法中的转换规则进行分类.第 3 节给出算法设计,提出缓存状态机理论和基于该理论的 ACC 算法,并给出算法的理论证明和扩展的探讨.第 4 节给出该算法的体系结构设计.第 5 节给出实验结果和代价分析.最后总结全文.

2 状态机类算法分析

2.1 基本的状态机类算法

基于状态机的模式匹配算法以 AC 算法为代表.AC 算法包括 3 个函数:goto(转移)函数、failure(失效)函数和 output(输出)函数.在文献[1]中,AC 算法通过将 failure 函数和 goto 函数合并,进行了优化.优化后的 AC 算法以 DFA 为基本模型构建,利用 DFA 的状态转移函数 δ 进行函数跳转.

以模式集{PAT,PPT}为例,基本的状态机类算法构造的 DFA 如图 1 所示.图中圆圈表示状态,编号为 S_i (i=0,…,5).圆圈间带箭头的曲线表示转换规则(跳转规则),每条规则上的方块表示规则编号.该模式集生成的状态机共包含 6 个状态和 16 个转换规则,转换规则见表 1.

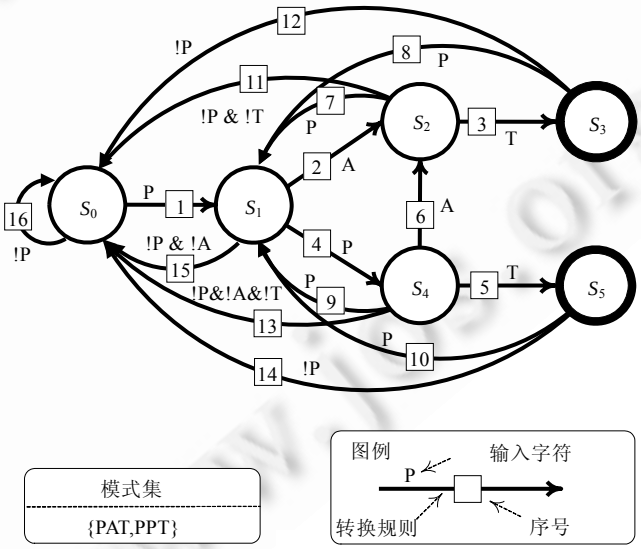


Fig.1 DFA built from basic AC algorithm

图 1 基本的状态机类算法构造的 DFA

Table 1 Transitional rules table in DFA built from AC algorithm

表 1 基本的状态机类算法中转换规则列表

规则编号	当前状态	输入字符		后状态
R1	S_0	P	→	S_1
R2	S_1	A	→	S_2
R3	S_2	T	→	S_3
R4	S_1	P	→	S_4
R5	S_4	T	→	S_5
R6	S_4	A	→	S_2
R7	S_2	P	→	S_1
R8	S_3	P	→	S_1
R9	S_4	P	→	S_1
R10	S_5	P	→	S_1
R11	S_2	!P & !T	→	S_0
R12	S_3	!P	→	S_0
R13	S_4	!P&!T&!A	→	S_0
R14	S_5	!P	→	S_0
R15	S_1	!P & !A	→	S_0
R16	S_0	!P	→	S_0

2.2 转换规则分类

基本的状态机类模式匹配算法基于DFA构造,而构建DFA所产生的转换规则有着不同的类型.传统模式匹配算法对转换规则没有任何区分,采用相同的策略进行状态转换.我们认为,从转换规则功能角度来看,模式匹配算法所构造的状态机中的转换规则可以分成4类:基本转换规则、交叉转换规则、失败转换规则和重启转换规则.以模式集{PAT,PPT}产生的状态机为例,各类转换规则定义如下:

- 基本转换规则.这是指从DFA初始状态开始直接接收模式所包含字符串的转换规则,如图1所示中的转换规则R1~R5.基本转换规则生成DFA所需的所有状态并将它们连接起来,形成了该DFA的基本结构.
- 交叉转换规则.表示状态机中某个状态代表的模式后缀与其他模式或该模式的前缀相同.图1中的转换规则R6为交叉转换规则,该规则表示状态S_4所代表的字符后缀P与模式PAT的前缀相同.因此,在状态S_4位置,当接收字符A时,将转移到代表模式PAT路径的S_2状态.交叉转换规则产生的实质是模式中某一段子模式与其他模式或者该模式的前缀相同.在本例中,由于PPT的子串(第2个字符)P与PAT的前缀(第1个字符)P相同,因此产生了交叉转换规则.
- 重启转换规则.定义为DFA中从其他状态转换到初始状态的后一个状态的转换规则,如图1所示中的转换规则 R7~R10.这类转换规则在模式匹配问题中表示输入字符无法正确匹配当前的模式路径,但可以从初始状态重新开始一次模式匹配过程,并已经接收了第1个字符.例如,状态S_2在接收字符P时无法完成模式PAT的匹配,但该输入字符可以重新开始模式PAT或者PPT的匹配过程,因此,状态S_2在接收字符P时根据重启转换规则跳转到状态S_1.
- 失败转换规则.定义为DFA中从其他状态转换到初始状态的转换规则,如图1所示中的转换规则R11~R16.这类转换规则代表输入字符无法正确地匹配任何模式及其子模式,因此,状态机回到初始位置.

2.3 带优先级的状态机类算法

2006年,IBM苏黎世研究院学者Lunteren也对AC算法的转换规则进行了分类[4].他将DFA中转换规则分成3类,并赋予不同的优先级,其主要贡献是区分了失败转换规则和重启转换规则,并利用规则优先级方法,采用无关操作符,将全部的失败转换规则和重启转换规则合并,并控制在256条之内.

以模式集{PAT,PPT}为例,带优先级的状态机类算法生成的DFA如图2所示,表2给出了转换规则列表(其中,*为无关操作符).

Lunteren将DFA中基本转换规则和交叉转换规则(他没有区分这两类规则)设为最高优先级2,将重启转换规则设为优先级 1,失败转换规则设为优先级 0.状态机的状态转换根据转换规则优先级进行,对于当前状态,首

先判断输入字符是否符合优先级为 2 的转换规则,如果符合,则跳转到后状态;否则,检查低优先级规则.

无关操作符的存在使得优先级为 1 的转换规则与当前状态无关,而是根据输入字符来判断需要跳转的状态.这类转换规则对应于重启转换规则,标识重新启动一次模式匹配.优先级为 0 的转换规则与当前状态和输入字符都无关,直接跳转到初始状态,对应于失败转换规则.

对于带优先级的状态机类算法,优先级为 1 的转换规则数量是初始状态 S_0 后状态的数量(图 2 所示例子中数量为 1),优先级为 0 的转换规则只有 1 条.由于初始状态 S_0 后状态的数量最多为 256(8 位输入字符对应的字母表最大为 256),因此,优先级为 1 的转换规则数量最多是 256.如果优先级为 1 的转换规则数量为 256,则不存在优先级为 0 的转换规则,因为对于字母表中的任何字符,都存在一个优先级为 1 的转换规则与其对应.因此,带优先级的状态机类算法中全部的失败转换规则和重启转换规则数量最多为 256 条.

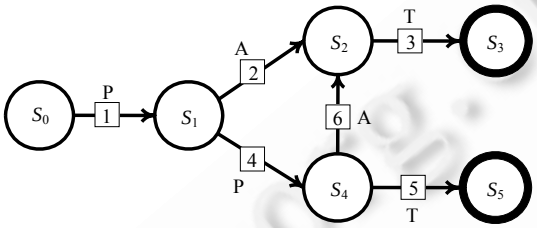


Fig.2 DFA built from priority based AC algorithm

图 2 带优先级的状态机类算法生成的 DFA

Table 2 Transitional rules table in DFA built from priority based AC algorithm

表 2 带优先级的状态机类算法中转换规则列表

规则编号	当前状态	输入字符		后状态	优先级
R1	S_0	P	→	S_1	2
R2	S_1	A	→	S_2	2
R3	S_2	T	→	S_3	2
R4	S_1	P	→	S_4	2
R5	S_4	T	→	S_5	2
R6	S_4	A	→	S_2	2
R7	*	P	→	S_1	1
R8	*	*	→	S_0	0

3 基于转换规则动态生成的匹配算法 ACC(Aho-Corasick-CDFA)

3.1 算法动机

根据上述转换规则分类,DFA 包含 4 类转换规则,通过优先级方法可以将重启转换规则和失败转换规则合并为 256 条.本节通过实际数据分析基本转换规则和交叉转换规则的数量,从而阐述 ACC 的动机.

所有实验数据基于两个模式库:网络入侵检测系统 Snort 的模式库和反病毒系统 ClamAV 的病毒特征库.

Snort[21]是一个开源的网络入侵检测系统,其规则对模式匹配研究具有重要意义,国际上的同类研究以 Snort 模式库作为标准测试集.本文使用 Snort 2.3.3 版本的一个模式集,包含 3 000 多条规则.提取所有规则中模式后组成一个模式集合,其中包含 1 785 条不同模式.

ClamAV[22]是一个开源的防病毒系统,其病毒特征库每天更新.本文采用的模式库包含约 50 000 条模式.去掉重复模式,组成模式库集,共包含 49 644 条模式.

我们实现了带优先级的状态机类算法,对两个模式库进行分析获得实验数据.为了便于统计不同规模模式集对 4 类转换规则数量的影响,以上述两个模式集为蓝本,对模式集进行了数量上的平均分割.本文引入“模式子集”的概念表示这种分割.以 Snort 模式集为例,模式子集数量为 1 表示 Snort 模式集本身,包括 1 785 条模式;模式子集数量为 2 表示将 Snort 模式集尽可能地在数量上平均分成两份,即生成两个模式集,数量分别是 892 和

893.在模式集分割过程中,没有采用任何优化分割方法.

Snort 模式集的各类转换规则统计如图 3 所示.由于采用优先级策略之后,重启转换规则和失败转换规则数量之和不大于 256,上述各类统计均不包含这两类规则.图中横坐标是模式子集的数量,纵坐标是对应转换规则的数量.对于模式子集大于 1 的情况,对应转换规则的数量为每个模式子集对应转换规则的数量之和.

由图 3 可以看出,随着模式子集数量的减少,即每个模式集数量的增加(从右向左),转换规则总数快速增加.这种转换规则数量的增加主要来源于交叉转换规则的增加,而基本转换规则数量没有发生变化.

对于 ClamAV 模式集,各类转换规则的统计如图 4 所示.其中,各类转换规则数量的变化规律与 Snort 模式集类似.由于模式集规模数量较大,转换规则数量增加十分明显.

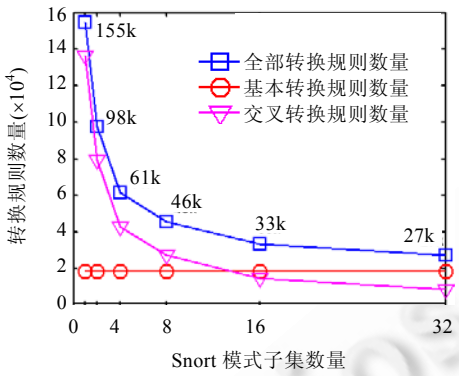


Fig.3 Statistics of transitional rules in Snort set

图 3 Snort 模式集中各类转换规则的统计

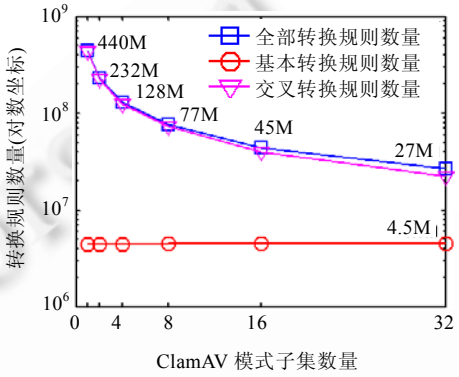


Fig.4 Statistics of transitional rules in ClamAV set

图 4 ClamAV 模式集中各类转换规则的统计

综合 Snort 和 ClamAV 模式集转换规则的统计可以看到,交叉转换规则数量是带优先级状态机类算法生成转换规则数量的重要组成部分,而且随着模式集包含的模式数量的增加,这类转换规则增加迅速.

根据分类原则,交叉转换规则来源于状态机中某状态代表的子串与其他模式或该模式前缀的同一性.基于这种相同性的程度,可以将交叉转换规则进一步加以分类:1 步交叉转换规则和 n 步交叉转换规则(n>1).1 步交叉转换规则定义为交叉转换规则中所相同的子串长度等于 1 的交叉转换规则;n 步交叉转换规则定义为交叉转换规则中相同的子串长度大于 1 的交叉转换规则.除第 3.6 节外,本文其他地方约定 n>1.

以模式集{pattern,testing}为例,生成的状态机如图 5 所示(图中未画出重启和失败转换规则),其中包含 3 条交叉转换规则,具体见表 3.pattern 中子串 t 与模式 testing 的前缀 t 相同,且此相同子串长度等于 1,因此,转换规则 R1 为 1 步交叉转换规则.模式 testing 中子串 t 与该模式前缀 t 相同,且长度等于 1,因此,转换规则 R2 也为 1 步交叉转换规则.模式 pattern 中,子串 te 与模式 testing 前缀 te 相同,子串长度等于 2,产生转换规则 R3,为 n 步交叉转换规则.

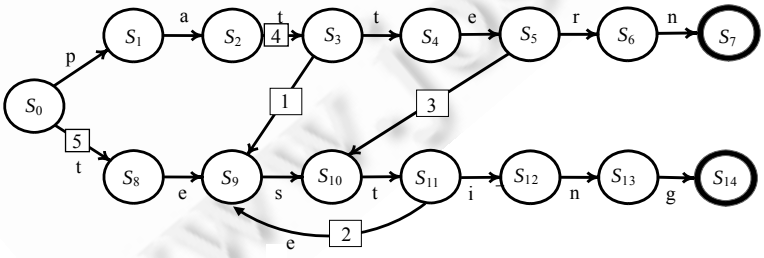


Fig.5 Example of 1-step cross transition and n-step cross transition

图 5 1 步交叉转换规则和 n 步交叉转换规则

Table 3 Cross transitional rules table

表 3 交叉转换规则列表

规则编号	当前状态	输入字符		后状态	优先级	分类
R1	S_3	e	→	S_9	2	1 步
R2	S_{11}	e	→	S_9	2	1 步
R3	S_5	s	→	S_{10}	2	n 步

经过上述分类,交叉转换规则可由 1 步交叉转换规则和 n 步交叉转换规则组成.我们可以进一步分析这两类交叉转换规则数量随着模式集变化的规律.图 6 和图 7 分别给出了 Snort 模式集和 ClamAV 模式集在不同模式子集情况下各类转换规则数量的比例(数据中不包含失败和重启转换规则的数量).

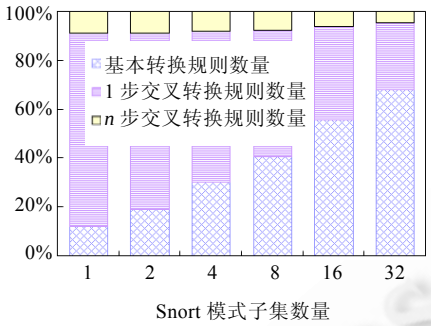


Fig.6 Proportion of different types of transitional rules in Snort set

图 6 Snort 模式集中各类转换规则数量比例

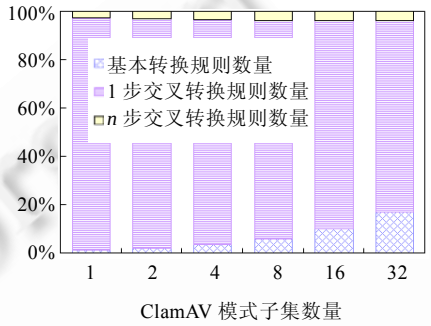


Fig.7 Proportion of different types of transitional rules in ClamAV set

图 7 ClamAV 模式集中各类转换规则数量比例

由图 6 可以看出,1 步交叉转换在全部转换规则数量中占很大比例,而且这种比例随着模式子集数量的减少(即模式集规模的增加)而增加.对于 Snort 模式全集(模式子集数量为 1),1 步交叉转换规则占整个转换规则数量的 79.2%.其中,1 步交叉转换规则占全部交叉转换规则数量的 89.9%.对于 n 步交叉转换规则,其数量所占比例及变化都很小.对于基本转换规则,尽管比例有所变化,但绝对数量基本未变.

图 7 中,大规模 ClamAV 模式集结果与 Snort 模式集类似.对于 ClamAV 全集,1 步交叉转换规则占全部规则数量的 95.9%,1 步交叉转换规则占全部交叉转换规则数量的 96.9%.n 步交叉转换规则所占据比例很小.

1 步交叉转换规则是导致优化 AC 算法状态机转换规则数量增加的主要原因.随着模式集规模的扩大,这类转换规则所占比例显著增加.对于大规模模式集,所占比例甚至超过 95%.1 步交叉转换规则的产生来源于模式集中某模式的非首字符与其他模式或者该模式首字符的相同性.显然,对于大规模模式集,这种情况十分普遍.甚至当模式集中所有模式首字符覆盖字母表时,每条模式的非首字符会产生一个 1 步转换规则.

本文方法的设计思想是:在模式匹配过程中,动态生成 1 步交叉转换规则,而不是静态地存储它们.从而在状态机中消除全部的 1 步转换规则,有效地减少了状态机中转换规则的数量.为了动态生成 1 步交叉转换规则,本文扩展了传统有限状态自动机模型,提出了缓存状态机.

3.2 缓存状态机模型

传统的状态机模型是图灵机模型的一种简化形式,无论确定性有限状态自动机(DFA)还是不确定性有限状态自动机(NFA),下一个状态仅由当前状态和当前输入决定,如图 8 所示.NFA 可以等价转化为等价的 DFA.

确定性有限状态自动机(DFA)定义为一个五元组,$M=\{K,\Sigma,s_0,F,\delta\}$,包括:

- 有限状态集合,记作 K,即所有状态的集合;
- 字母表集合,记作Σ,即状态机接收字符的集合;
- 状态机的开始状态,记作 s_0;
- 接收状态集合,记作 F;

- 状态转换函数,δ:$K\times\Sigma\to K$.

其中,状态转换函数是一个二元函数,它根据状态机所处的当前状态和接收字符决定下一个状态.基本的状态机类算法和带优先级的状态机类算法都是基于 DFA 模型的模式匹配算法.

缓存状态机(cached deterministic finite automata,简称 CDFA)是本文提出的一种状态机模型.该模型的初衷是,希望能够在原有状态机功能中增加对历史信息的记录,并由当前状态、当前输入和状态机的历史信息一起决定下一个状态.缓存状态机打破了传统状态机"下一个状态仅由当前状态和当前输入决定"的特点,通过对历史信息的记录,增加了状态机在决定后状态时操作的丰富性.缓存状态机通过在状态机中增加状态缓存功能达到上述的设计目的,如图 9 所示.从外部接口来看,缓存状态机与传统状态机一样,仅接收输入字符,输出状态机的判断结果.所不同之处在于内部增加了缓存寄存器(cache),能够对状态进行一定策略的缓存.

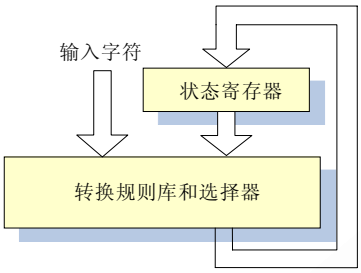


Fig.8 DFA model

图 8 DFA 模型

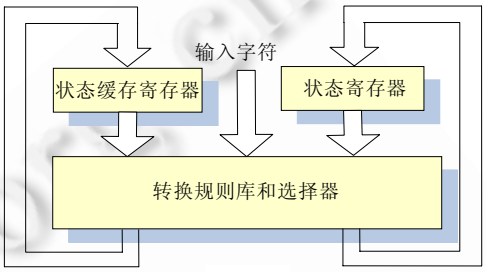


Fig.9 Cached DFA model

图 9 缓存状态机模型

缓存状态机(CDFA)可以定义为一个七元组,$M'=\{K,\Sigma,s_0,F,N,\delta,\theta\}$,包括:

- 有限状态集合,记作 K,即所有状态的集合;
- 字母表集合,记作Σ,即状态机接收字符的集合;
- 状态机的开始状态,记作 s_0;
- 接收状态集合,记作 F;
- 状态机中包含的缓存数量,记作 N;
- 状态转换函数,δ:$K\times K^N\times\Sigma\to K$;
- 策略转换函数,θ:$K\times\Sigma\to K^N$.

其中,缓存策略函数θ根据当前状态和当前输入决定要缓存的状态,状态转换函数δ根据当前状态、被缓存状态和输入字符决定下一个状态.由于新增加的寄存器由状态机根据缓存策略函数控制,对外不可见,这与计算机存储系统中 Cache 的设计原理相似,因此,命名这种新的状态机模型为缓存状态机模型.

3.3 ACC算法思想和描述

ACC 算法基于缓存状态机设计,通过动态方法在匹配过程中生成 1 步交叉转换规则,从而显著减少状态机中转换规则的数量.根据图 6 和图 7 的数据统计可知,1 步交叉转换规则数量很大,对于大规模模式集甚至超过总数的 95%.ACC 算法可以将这类超过 95%的转换规则全部去掉.

1 步交叉转换规则之所以可以动态生成,是因为 1 步交叉转换规则产生所需要的信息已经保存在状态机的基本转换规则中.如图 5 所示,1 步交叉转换规则 R1 的产生信息来源于转换规则 R4 和转换规则 R5.正是因为转换规则 R4 接收字符 t 与初始状态 S_0 出发的转换规则 R5 接收字符相同,因此,S_3 状态接收字符 e 会产生交叉转换规则跳到 S_9.n 步交叉转换规则与 1 步交叉转换规则类似,可以通过基本转换规则判断生成.

基于交叉转换规则从基本转换规则生成的特点,我们利用缓存状态机模型设计了基于转换规则动态生成的模式匹配算法 ACC.ACC 算法采用带有一个缓存寄存器的缓存状态机可以生成全部 1 步交叉转换规则.

3.4 ACC算法细节和实例

ACC 算法中采用的缓存状态机 CDFA 是一个七元组$\{K,\Sigma,s_0,F,1,\delta,\theta\}$,需要的缓存状态数 N=1,即只需要 1 个寄存器进行状态缓存.状态转换函数δ可以分为以下两类:

- δ_{basic}为基本转换规则的状态转换函数,δ_{basic}:$K\times\Sigma\to K$,ACC 算法δ_{basic}的定义与 AC 算法中的相同;
- δ_{ncross}为 n 步交叉转换规则的状态转换函数,δ_{ncross}:$K\times\Sigma\to K$,该定义与 AC 算法中定义相同.

状态转换函数δ的定义为

$$\delta(S_i,\mathrm{c},S_k)=\begin{cases}S_j, \text{ if } (\delta_{basic}(S_i,\mathrm{c})=S_j \parallel \delta_{ncross}(S_i,\mathrm{c})=S_j), & priority=3\\ S_j, \text{ if } (\delta_{basic}(S_k,\mathrm{c})=S_j \parallel \delta_{ncross}(S_k,\mathrm{c})=S_j), & priority=2\\ S_j, \text{ if } (\delta_{basic}(S_0,\mathrm{c})=S_j), & priority=1\\ S_0, \text{ if } (\delta_{basic}(S_0,\mathrm{c})=\varnothing), & priority=0\end{cases} \tag{3-1}$$

其中,$\varnothing$为“空”,表示不存在对应的转换规则;*priority* 为优先级标识,3 为最高优先级,0 为最低优先级.如果高优先级的结果有效(非空),则该结果被优先采纳;如果优先级高的结果无效,则低优先级结果被采纳.结果无效是指某一状态 S_i 在δ_{basic}和δ_{ncross}转换函数中没有接收字符 c 的转换规则.

状态转换函数δ的含义是,对于 ACC 中 CDFA 的状态转换,首先判断当前状态 S_i 在基本转换规则和 n 步交叉转换规则中是否存在接收当前字符 c 的转换规则:

- 如果存在,则应用该规则,跳到下一个状态 S_j,实施基本转换规则或 n 步交叉转换规则;
- 如果不存在对应的转换规则,则将被缓存的状态 S_k 取出,并以 S_k 状态为当前状态在基本转换规则和 n 步交叉转换规则中寻找接收当前字符 c 的转换规则:
  - ✧ 如果存在,则跳转到对应的下一个状态 S_j,实施 1 步交叉转换规则;
  - ✧ 如果不存在对应的转换规则,则判断初始状态 S_0 是否接收字符 c:如果接收,则跳到相应状态,实施重启转换规则;否则,跳转到初始状态 S_0,实施失败转换规则.

上述 4 个优先级操作之间无关,可以采用并行操作,不影响最终性能.

缓存策略函数θ定义为

$$\theta(S_i,\mathrm{c})=\begin{cases}S_j, \text{ if } (\delta_{basic}(S_0,\mathrm{c})=S_j)\\ S_0, \text{ if } (\delta_{basic}(S_0,\mathrm{c})=\varnothing)\end{cases} \tag{3-2}$$

缓存策略函数θ的含义是,对于 ACC 中 CDFA 的缓存空间(只有 1 个),每周期进行缓存,被缓存的内容是初始状态 S_0 接收当前输入字符 c 对应的下一个状态;如果在基本转换规则中不包含该对应转换规则,则缓存初始状态 S_0.可以看到,缓存策略函数与当前状态 S_i 无关.

ACC 算法基于上述缓存状态机,该算法主要包含两个步骤:预处理和匹配.预处理阶段读入模式集,构造缓存状态机;匹配阶段读入待匹配文本,进行状态机转换,并在特定状态报告匹配.

预处理阶段与 AC 算法类似,但仅需要构造基本转换规则和 n 步交叉转换规则,对应的构造算法伪代码如图 10 和图 11 所示[20].ACC 算法与 AC 算法不同,它只需构造基本转换规则和 n 步交叉转换规则.

为了输出模式匹配信息,需要在缓存状态机中特定状态输出模式结束(匹配成功).这需要在构造状态机时能够标记哪些状态需要输出匹配信息,这种标记算法只是一种线性查找算法,比较简单,本文略去.

ACC 算法的匹配阶段利用构造的缓存状态机逐个接收输入文本的字符,同时进行状态转换.状态转换根据缓存状态机的状态转换函数δ进行.下面通过一个实例说明 ACC 算法的具体匹配过程.

模式集{pattern,testing}采用带优先级的状态机类算法所构造的 DFA 如图 5 所示.采用 ACC 算法构造的 CDFA 去掉了其中的 1 步交叉转换规则(R1 和 R2),如图 12 所示,其中,n 步交叉转换规则保留.

待匹配输入流 patesting 从第 1 周期输入到缓存状态机中.我们以周期 3 接收字符 t、周期 4 接收字符 e 为例说明该缓存状态机的匹配流程.

周期 3,待匹配文本输入字符 t,此时的当前状态为 S_2.由基本转换规则可知,$\delta_{basic}(S_2,\mathrm{t})=S_3$,所以后状态跳转到 S_3.同时,根据缓存策略,$\theta(S_2,\mathrm{t})=\delta_{basic}(S_0,\mathrm{t})=S_8$,因此,状态 S_8 被缓存.

```
Create_Basic_Trans()
1:   for each pattern(i) in PatternSet do
2:     for each char(j) in pattern(i) do
3:       j==0 ? CurState=S0: CurState=LastState
4:       if basic_trans(CurState,c) do
5:           CurState=basic_trans(CurState,c)
6:       else
7:           NewState=CreateState()
8:           CreateTrans(CurState,c,NewState)
9:           CurState=NewState
10:      end if
11:   LastState=CurState
12:   end for
13: end for
```

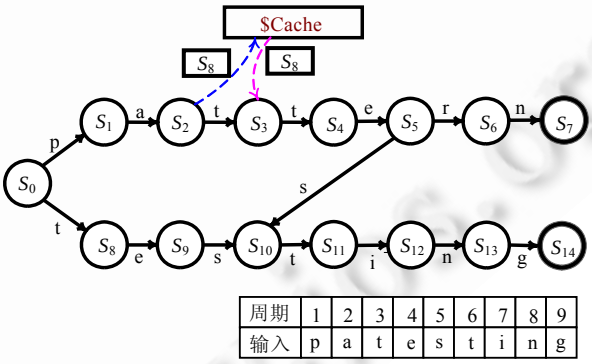
Fig.10 The building algorithm of basic transitions in ACC algorithm

图 10 ACC 算法中基本转换规则构造算法

```
Create_NStep_Trans()
1:   for each pattern(i) in PatternSet do
2:     for each char(j) in pattern(i) do
3:       j==0 ? CurState=S0: CurState=LastState
4:       if basic_trans(CurState,c) && i!=0 do
5:           CrossState=basic_trans(S0,c)
6:           k=0; TmpState=CurState
7:           do
8:              t=pattern[j+k]
9:              if basic_trans(CrossState,c) && (j+k=plen−1) do
10                CrossState=basic_trans(CrossState,t)
11                TmpState=basic_trans(TmpState,t)
12             else if k>1
13                for each q in basic_trans(CrossState,[0,255]) do
14:          CreateTrans(TmpState,q,basic_trans(CrossState,q))
15:            end for; end if; end if
16:          end do while (basic_trans(CrossState,t))
17:       end if
```

Fig.11 The building algorithm of *n*-step cross transitions in ACC algorithm

图 11 ACC 算法中 *n* 步交叉转换规则构造算法



周期	1	2	3	4	5	6	7	8	9
输入	p	a	t	e	s	t	i	n	g

Fig.12 The matching process in ACC algorithm for an example set {pattern, testing}

图 12 ACC 算法的匹配过程,模式集{pattern,testing}

周期 4,待匹配文本输入字符 e,此时的当前状态为 S_3,被缓存的状态为 S_8.根据转换规则函数δ的处理方法可知,首先判断$\delta_{basic}(S_3,e)=NULL$ 且$\delta_{ncross}(S_3,e)=NULL$,即不存在对应转换规则;然后将缓存状态 S_8 取出,由基本转换规则可知,$\delta_{basic}(S_8,e)=S_9$,因此,后状态跳转到 S_9.同时,根据缓存策略,$\theta(S_3,e)=\delta_{basic}(S_0,e)=NULL$,因此,状态 S_0 被缓存.在该周期,当 CDFA 接收字符 e 时,实现了从 S_3 状态跳转到 S_9 状态.这个状态转换在图 5 中是通过 1 步交叉

转换规则来实现的.通过动态生成 1 步交叉转换规则而不是静态存储这些规则,ACC 算法比 AC 算法可生成更少的转换规则.

3.5 算法理论证明

定理 1. 基于 CDFA 的 ACC 算法与基于 DFA 的 AC 算法等价.

证明:为了证明 ACC 和 AC 算法等价,只需证明两种算法生成的状态机等价即可.两个状态机等价当且仅当两个状态机在接收相同输入时产生相同输出.

DFA 是一个五元组$\{K,\Sigma,s_0,F,\delta^{DFA}\}$,根据第 2.2 节对转换规则的分类可知,对于给定模式集,状态机的转换规δ^{DFA}可以分为以下 5 个子集:δ_{basic}为基本转换规则,δ_{1cross}为 1 步交叉转换规则,δ_{ncross}为 n 步交叉转换规则,$\delta_{failure}$为失败转换规则,$\delta_{restart}$为重启转换规则.

CDFA 是一个七元组$\{K,\Sigma,s_0,F,1,\delta^{CDFA},\theta\}$,对于给定模式集,缓存状态机的转换规则$\delta^{CDFA}$可以分为以下两个子集:$\delta_{basic}$为基本转换规则,$\delta_{ncross}$为 n 步交叉转换规则.这两个子集的生成与 AC 算法相同.

下面,我们证明 DFA 中δ_{1cross},$\delta_{failure}$和$\delta_{restart}$能够被 CDFA 等价表示.

对于 1 步交叉转换规则,如图 13 所示,当前状态为 S_i,输入字符为 b,在 DFA 中存在$\delta_{1cross}(S_i,b)=S_j$.根据 1 步交叉转换规则定义,状态 S_h 接收的字符 a 与模式 ab 的前缀 a 相同.因此,此交叉转换规则等价于$\delta_{basic}(S_k,b)=S_j$.

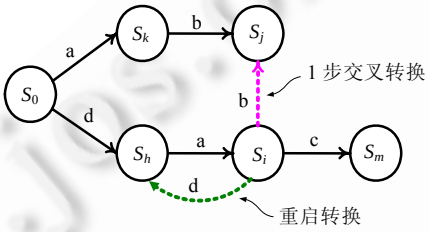


Fig.13 Proof that ACC algorithm is equivalent to AC algorithm

图 13 证明:ACC 算法与 AC 算法等价

CDFA 不存在 1 步交叉转换规则,$\delta_{basic}(S_i,b)=NULL$,$\delta_{ncross}(S_i,b)=NULL$,因此采用缓存状态进行转换,后状态来源于$\delta_{basic}(\theta(S_h,a),b)$,缓存定义推导过程如公式(3-3)所示.后状态跳转到 S_j,与 DFA 状态转换相同,即 ACC 算法动态生成的 1 步交叉转换规则与 DFA 静态生成的等价.

$$\delta_{basic}(\theta(S_h,a),b)=\delta_{basic}(\delta_{basic}(S_0,a),b)=\delta_{basic}(S_k,b)=S_j \tag{3-3}$$

对于 DFA 中的失败转换规则,例如$\delta_{failure}(S_i,k)=S_0$,由失败转换规则定义,当前状态 S_i 接收字符 k 不满足基本转换规则和交叉转换规则,它可以被替换为δ^{CDFA}中优先级为 0 的跳转方式,即$\delta^{CDFA}(S_i,k)=S_0$.

对于 DFA 中重启转换规则,如图 13 所示,$\delta_{restart}(S_i,d)=S_h$.由重启转换规则定义,当前状态 S_i 接收字符 d 不满足基本转换规则和交叉转换规则,它可以被替换为δ^{CDFA}中优先级为 1 的跳转方式,如公式(3-4)所示.

$$\delta^{CDFA}(S_i,d)=\delta_{basic}(S_0,d)=S_h \tag{3-4}$$

至此,证明了 AC 算法中 5 类转换规则完全可以由 ACC 中 CDFA 表示,ACC 算法与 AC 算法等价. □

3.6 算法扩展

ACC 算法基于如下缓存状态机$\{K,\Sigma,s_0,F,1,\delta,\theta\}$,其中,缓存寄存器的数量 N 为 1.通过该缓存状态机,ACC 算法可以动态生成全部的 1 步交叉转换规则.实际上,ACC 算法可以进一步扩展为采用包含多个缓存寄存器的 CDFA.

理论上,如果采用 k 个单位的缓存寄存器,CDFA 可以动态生成 k 步以内的全部交叉转换规则.从算法扩展角度来看,将本文提出的动态生成 1 步交叉转换规则的算法命名为 ACC-1 算法,而将能够消除 k 步以下交叉转换规则的算法命名为 ACC-k 算法,在预处理阶段,该算法生成基本转换规则和 n 步交叉转换规则($n>k$).

ACC 算法扩展的额外代价是进一步增加了状态转换的复杂性.本文选择了 N=1,因为 1 步交叉转换数量在

全部转换规则中占主导地位,将其去掉可以显著减少转换规则数量,而且状态转换复杂性较低,相当于取了一个合理的折中.对于超大规模模式集,2 步甚至多步交叉转换规则数量会显著增加,适当增大 N 值,ACC 算法可以进一步优化存储空间.对于以中文字符为主的关键词库模式匹配问题,设置 N 为 2 的倍数比较合适.

以模式集{pattern,testing}为例,图 14 给出了 ACC-2 算法的具体过程.与图 12 相比,缓存状态机去掉了全部 1 步和 2 步交叉转换规则(即,S_5~S_{10} 的转换规则)(图中虚线为说明方法过程而用,非实际需要存储的转换规则).ACC-2 需要 2 个单位缓存寄存器,Cache-1 与 ACC-1 算法相同,用于存储生成 1 步交叉转换规则生成所需要的状态, Cache-2 用于存储 2 步交叉转换规则生成所需要的状态.第 5 个周期,输入字符 e,当前状态为 S_4,ACC-2 算法首先根据转换规则跳转到 S_5 状态,同时,将 Cache-1 中缓存状态 S_8 取出后也进行了状态转换.由于 S_8 状态接收字符 e 到 S_9 状态,S_9 状态被缓存到 Cache-2 中.第 6 个周期,当前 S_5 状态不接收字符 s,优先判断 Cache-2 中状态 S_9,它接收字符 s 到状态 S_{10},则状态机当前状态更新为 S_{10}(如图 14 中虚线所示).假如 Cache-2 中状态不接收当前字符,进而判断 Cache-1 中状态是否接收当前字符.

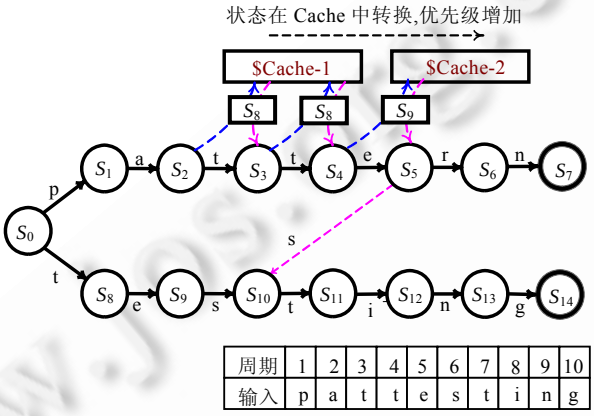


Fig.14 An exmaple for ACC-2 algorithm

图 14 ACC-2 算法举例

ACC-k 算法扩展了 ACC-1 算法的核心思想,利用缓存状态机,可以去掉全部的 1 到 k 步交叉转换,对应的缓存状态机 CDFA 是一个七元组$\{K,\Sigma,s_0,F,k,\delta,\theta\}$,$k$ 个缓存状态分别是$\{S_{c1},S_{c2},\ldots,S_{ck}\}$,$S_{ci}$ 代表第 i 个缓存状态,用于消除第 i 步交叉转换规则.CDFA 中剩余的 n 步交叉转换规则表示为$\delta_{ncross},n>k$.

状态转换函数δ的定义为

$$\delta(S_i,\mathrm{c},S_k)=\begin{cases}S_j,\ \text{if}\ (\delta_{basic}(S_i,\mathrm{c})=S_j\ \|\ \delta_{ncross}(S_i,\mathrm{c})=S_j), & priority=k+2\\ S_j,\ \text{if}\ (\delta_{basic}(S_i,\mathrm{c})=S_j\ \|\ \delta_{ncross}(S_i,\mathrm{c})=S_j), & priority=k+1\\ \ldots & \\ S_j,\ \text{if}\ (\delta_{basic}(S_k,\mathrm{c})=S_j\ \|\ \delta_{ncross}(S_{c1},\mathrm{c})=S_j), & priority=2\\ S_j,\ \text{if}\ (\delta_{basic}(S_0,\mathrm{c})=S_j), & priority=1\\ S_0,\ \text{if}\ (\delta_{basic}(S_0,\mathrm{c})=\varnothing), & priority=0\end{cases} \tag{3-5}$$

缓存策略函数$\theta=\{\theta_1,\theta_2,\ldots,\theta_k\}$,其中,第 1 个缓存状态的缓存策略函数$\theta_1$为公式(3-2),第 m+1 个缓存状态的缓存策略函数θ_{m+1}定义为

$$\theta_{m+1}(S_i,\mathrm{c},S_{cm})=\begin{cases}S_j,\ \text{if}\ (\delta_{basic}(S_{cm},\mathrm{c})=S_j\ \|\ \delta_{ncross}(S_{cm},\mathrm{c})=S_j)\\ \varnothing,\ \text{others}\end{cases} \tag{3-6}$$

ACC-k 算法与 ACC 算法的等效性证明与第 3.5 节内容相似,这里略去.

4 体系结构设计

ACC 算法可以应用于软件和硬件实现,硬件实现中,ACC 算法需要结合特定的模式匹配体系结构.实时性要求较高的千兆级网络入侵检测/防御系统对算法的硬件实现需求迫切,而对于百兆网络环境,基于嵌入式工业平台的网络安全系统则需要算法的软件实现.这里,给出 ACC 算法的一种硬件体系结构设计.

图 15 是 ACC 算法的一种直观体系结构实现.该结构中,状态寄存器保存当前状态,状态缓存寄存器保存当前的缓存状态,所有的转换规则存储在片上或者片外的存储器中,包含两部分:S_0 后状态表和其他规则表.其中,S_0 后状态表由于表示的是从 S_0 开始的转换规则,且最多为 256 条,所以,可以根据输入字符直接存储转换规则所跳到的状态,形成一个状态表.

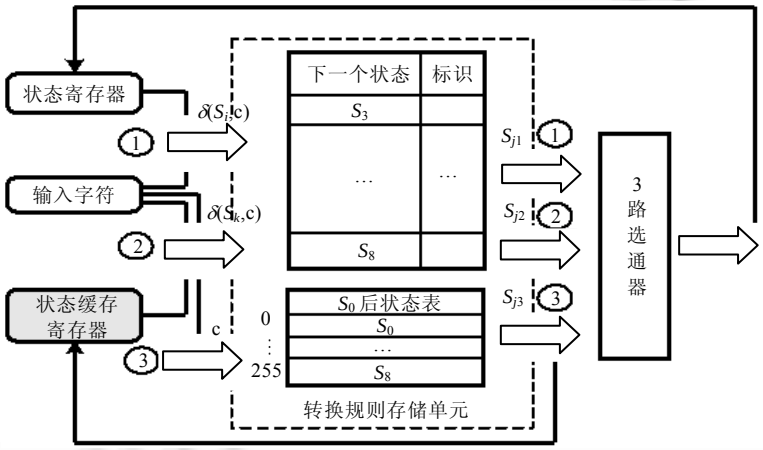


Fig.15 The hardware architecture and implementation for ACC algorithm

图 15 ACC 算法的体系结构实现

在硬件结构中,数据通路主要有 3 条,汇聚在 3 路选通器上.第 1 条数据通路是当前状态 S_i 和当前输入字符 c 根据转换规则 $\delta_{basic}(S_i,c)$ 形成的下一个状态 S_{j1};第 2 条数据通路是当前缓存状态 S_k 和当前输入字符 c 根据转换规则 $\delta_{basic}(S_k,c)$ 形成的下一个状态 S_{j2};第 3 条数据通路是根据当前字符 c 在 S_0 后状态表中查找的状态,对应于失败和重启转换规则.由于硬件结构的并行性特点,3 条数据通路采取并行访问和执行的方式运行,并汇聚在 3 路选通器数据最终结果.3 路选通器根据公式(3-1)对结果 S_{j1},S_{j1},S_{j1} 进行选择输出.

转换规则存储和选择单元中存储着状态机中所有的转换规则,并根据状态和输入字符进行选择,这部分结构的主要功能是实现状态机的硬件转换,目前主要的方法有 3 种,包括 TCAM 方式[23]、哈希方式[4]和后状态查找方式[15].由于这部分内容相对独立,不是本文关注的重点,这里不再详细阐述.结合图 15 的结构,本文实现的具体硬件结构采用了后状态查找方式.

5 实验结果和分析

ACC 算法的硬件实现需要结合特定的模式匹配体系结构.这里,我们分别给出 ACC 算法软件实现中的存储结果和硬件实现中的匹配速度结果,并分析算法代价.

5.1 存储空间结果分析

ACC 算法最大的贡献在于将模式集状态机中的 1 步交叉转换规则全部去掉,并改为动态生成.对于 Snort 模式集,79.2%的转换规则被去掉,占全部交叉转换规则的 89.9%,如图 6 所示.对于 ClamAV 模式集,95.9%的转换规则被去掉,占全部交叉转换规则的 96.9%,如图 7 所示.

ACC 算法软件版本中缓存状态机状态的数据结构如图 16 所示.其中,该数据结构使用了 AC 优化方法之一

的节点压缩方法[3].该方法使用一个 256 位的 bitmap 来标识对应字符是否存在下一个状态,如果存在,则将 bitmap 中该位之前的所有位求和,作为偏移量加到 *p_next_state* 指针上.针对 Snort 模式集,采用各种算法在软件实现中所需的存储需求见表 4.

```
Structure acc_cdma_state
1:  struct acc_cdma_state
2:      integer state_value
3:      bool    report_match
4:      char*   to_report_rule
5:      bitmap next_state_valid   //256 bits
6:      struct acc_cdma_state*p_next_state
7:  end struct
```

Fig.16 The data structure of ACC algorithm in software implementation

图 16 ACC 算法软件实现数据结构

Table 4 Memory size comparison of different methods

表 4 各类算法实现的存储需求比较

算法	存储空间/模式字符
Aho-Corasick[2,4]	2.8KB
Wu-Manber[4,24]	1.6KB
节点压缩的 AC 算法[3,4]	154B
路径压缩的 AC 算法[3,4]	60B
带优先级的 AC 算法[4]	41B
XFA 算法[6,7]	67B (1.05KB/Snort 规则)
本文 ACC 算法软件实现	24.3B
CompactDFA[16]硬件实现	9.5B~14.5B
B-FSM[4,5]硬件实现	7.4B
本文 ACC 算法硬件实现	3.3B~7.4B

由于不同算法在不同文献中采用的模式库大小不尽相同,这里采用每个模式字符对应的存储空间作为衡量标准.可以看到,本文提出的 ACC 方法在存储空间上比已有的几种 AC 优化方法都更为有效,比带优先级的 AC 算法存储空间降低 40.7%,存储效率提高近 2 倍.对于各种算法的硬件实现,ACC 所占用的存储空间大约在每字符 3.3B 左右,比其他几种方法更为有效,存储效率提高 2 倍左右.

5.2 性能分析

我们使用 verilog 语言实现 ACC 算法的硬件结构,在验证正确性后,使用 0.18μm 工艺标准单元库进行综合.在综合过程中,为了缩短最短路径延时,对体系结构进行了流水线划分.

在状态机转换过程中,判断后一个状态时需要知道当前状态,因此对于单数据流输入很难分割流水线.我们借鉴了细粒度流水线[25]方式组织数据流设计流水线.这种方式的基本思想是,以周期为单位从不同数据流中取得数据,每个流水段中只处理一个数据流数据,不同流水段之间处理不同数据流数据.多数据流是网络流量固有的特点,因此,这种流水线分割方法十分自然,同时也能为 ACC 结构带来较高的频率和匹配吞吐率.

表 5 给出了 ACC 硬件实现与其他方法的匹配速度比较.其中,ACC 结构的“多级流水线”是指该结构可能的最短关键路径,这条关键路径由其中的加法(ADD)模块产生.实验数据表明,ACC 算法的单个结构具有超过 10Gbps 的匹配速度,与其他研究相比,具有较好的速度提高.

Table 5 The performance comparison of hardware implementations for different methods, including ACC

表 5 ACC 硬件实现与其他方法的匹配速度比较

结构	关键路径延时(ns)	匹配速度(Gpbs)	说明(1 字节/周期)
ACC-NSA,无流水线	1.65	6.1	0.18μm 工艺综合
ACC-NSA,2 级流水线	1.18	8.5	
ACC-NSA,多级流水线	0.85	11.7	
Bit split FSM[12]	N/A	8.4~11.0	使用模拟器计算
BFSM[4]	N/A	0.8~1.0	FPGA
Cho-MSmith[26]	1.12	7.14	0.18μm 工艺综合
Predec CAMs[23]	N/A	2.68	FPGA
CompactDFA[16]	N/A	2~10	估算

5.3 代价分析

由上述研究可以看到,ACC 算法有一些额外代价,主要来源于如下两个方面:第一,增加了 1 个单位的缓存寄存器,该寄存器大小与状态机中状态寄存器大小相同,这个代价在实际情况中可以忽略不计;第二,增加了状态转换的复杂性.根据公式(3-1)可以看到,在进行状态转换时,需要进行 4 次判断,相比较,带有优先级的 AC 算法需要进行 3 次判断.

如果 ACC 算法应用于软件实现,则状态转换的复杂性会对匹配速度带来一定影响.但由于生成的转换规则集很小,有利于有效利用系统的高速缓存,因此速度影响并不明显.

如果 ACC 算法应用于硬件实现,由于状态转换中 4 次判断(优先级为 1 和 0 的 2 次判断可以合并)之间并不相关,因此可以采取并行访问方式,有效屏蔽 4 次判断带来的复杂性.因此,ACC 算法可以获得与 AC 算法几乎相同的匹配速度.ACC 算法的存储有效性能够使硬件设计利用片上 SRAM 成为可能,为大规模模式匹配提供更高的性能.

6 结束语

本文提出了一种新的状态机模型——缓存状态机,并基于该模型设计了存储有效的多模式匹配算法 ACC 及其体系结构.理论证明,该算法与 AC 算法等价.ACC 算法基于带优先级的状态机类模式匹配算法,但本质上所不同的是,ACC 算法基于缓存状态机设计,利用其缓存机制,在匹配过程中动态生成而不是存储 1 步交叉转换规则.实验结果表明,ACC 算法生成的状态机模型,比已有优化方法构造的 DFA 模型多使用 1 个单位的缓存(存储)空间,但可以消除原有状态机中 80%~95%的转换规则,效果明显.该算法的软件实现比现有最优的模式匹配算法降低存储空间达 40.7%,适合大规模模式匹配的应用.ACC 算法的硬件实现可以达到单芯片超过 10Gbps 的匹配速度,并可以有效利用片上 RAM.对于含有 1.5k 条模式的真实 Snort 模式库,采用本文方法实现的 ACC 硬件结构仅需要 83KB 的存储空间.

随着网络带宽超越摩尔定律的高速增长,高速模式匹配算法和体系结构相关研究对提高多种网络安全应用性能起着至关重要的作用.ACC 算法适合作为模式匹配体系结构设计的基本算法,同时也可以作为软件的基本算法而实现.但无论采用哪种实现方式,都会带来存储方面的较大优势,具有良好的应用前景.

References:

[1] Navarro G, Raffinot M. Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge: Cambridge University Press, 2002. 41−75.

[2] Aho AV, Corasick MJ. Efficient string matching: An aid to bibliographic search. Communications of the ACM, 1975,18(6): 333−340. [doi: 10.1145/360825.360855]

[3] Tuck N, Sherwood T, Calder B, Varghese G. Deterministic memory-efficient string matching algorithms for intrusion detection. In: Li VOK, ed. Proc. of the IEEE Infocom 2004. Piscataway: IEEE, 2004. 333−340. [doi: 10.1109/INFCOM.2004.1354682]

[4] Lunteren JV. High-Performance pattern-matching for intrusion detection. In: Li VOK, ed. Proc. of the IEEE Infocom 2006. Piscataway: IEEE, 2006. 1−13. [doi: 10.1109/INFOCOM.2006.204]

[5] Lunteren JV, Guanella A. Hardware-Accelerated regular expression matching at multiple tens of Gb/s. In: Li VOK, ed. Proc. of the IEEE Infocom 2012. Piscataway: IEEE, 2012. 1737−1745. [doi: 10.1109/INFCOM.2012.6195546]

[6] Smith R, Estan C, Jha S, Kong S. Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. In: Bahl V, Wetherall D, eds. Proc. of the ACM SIGCOMM 2008. New York: ACM Press, 2008. 207−218. [doi: 10.1145/1402958.1402983]

[7] Smith R, Estan C, Jha S, Siahaan I. Faster signature matching with extended automata (XFA). In: Sekar R, Pujari AK, eds. Proc. of the ICISS 2008. Heidelberg, Berlin: Springer-Verlag, 2008. 158−172. [doi: 10.1007/978-3-540-89862-7_15]

[8] Yang YH, Prasanna VK. Space-Time tradeoff in regular expression matching with semi-deterministic finite automata. In: Singh JR, Ren K, eds. Proc. of the IEEE INFOCOM 2011. Piscataway: IEEE, 2011. 1853−1861. [doi: 10.1109/INFCOM.2011.5934986]

[9] Lin CH, Liu CH, Chang SC. Accelerating regular expression matching using hierarchical parallel machines on GPU. In: Du Q, ed. Proc. of the Globecom 2011. Piscataway: IEEE, 2011. 1−5. [doi: 10.1109/GLOCOM.2011.6133663]

[10] Vasiliadis G, Polychronakis M, Antonatos S, Markatos EP, Ioannidis S. Regular expression matching on graphics hardware for intrusion detection. In: Kirda E, Jha S, Balzarotti D, eds. Proc. of RAID 2009. Heidelberg, Berlin: Springer-Verlag, 2009. 265−283. [doi: 10.1007/978-3-642-04342-0_14]

[11] Becchi M, Cadambi S. Memory-Efficient regular expression search using state merging. In: Ramasubramanian S, ed. Proc. of the Infocom 2007. Piscataway: IEEE, 2007. 1064−1072. [doi: 10.1109/INFCOM.2007.128]

[12] Tan L, Sherwood T. A high throughput string matching architecture for intrusion detection and prevention. In: Zilles C, ed. Proc. of

the ISCA 2005. Washington: IEEE Computer Society, 2005. 112−122. [doi: 10.1109/ISCA.2005.5]

[13] Lu H, Zheng K, Liu B, Zhang X, Liu Y. A memory-efficient parallel string matching architecture for high speed intrusion detection. IEEE Journal on Selected Areas in Communications, 2006,24(12):1793−1804. [doi: 10.1109/JSAC.2006.877221]

[14] Yu F, Chen Z, Diao Y, Lakshman TV, Katz RH. Fast and memory-efficient regular expression matching for deep packet inspection. In: Berenbaum A, ed. Proc. of the ANCS 2005. New York: ACM Press, 2006. 93−102. [doi: 10.1145/1185347.1185360]

[15] Song T, Zhang W, Wang DS, Xue YB. A memory efficient multiple pattern matching architecture for network security. In: Gorinsky S, Kuzmanovic A, eds. Proc. of the IEEE Infocom 2008. Piscataway: IEEE, 2008. 166−170. [doi: 10.1109/INFOCOM.2008.42]

[16] Bremler-Barr A, Hay D, Koral Y. CompactDFA: Generic state machine compression for scalable pattern matching. In: Boutaba R, Chatterjee M, eds. Proc. of the IEEE Infocom 2010. Piscataway: IEEE, 2010. 659−667. [doi: 10.1109/INFCOM.2010.5462160]

[17] Zheng K, Zhang X, Cai ZP, Wang ZJ, Yang BH. Scalable NIDS via negative pattern matching and exclusive pattern matching. In: Boutaba R, Chatterjee M, eds. Proc. of the IEEE Infocom 2010. Piscataway: IEEE, 2010. 731−739. [doi: 10.1109/INFCOM.2010.5462152]

[18] Lin PC, Lin YD, Lai YC. A hybrid algorithm of backward hashing and automaton tracking for virus scanning. IEEE Trans. on Computers, 2011,60(4):594−601. [doi: 10.1109/TC.2010.95]

[19] Li WN, E YP, Ge JG, Qian HL. Multi-Pattern matching algorithms and hardware based implementation. Ruan Jian Xue Bao/Journal of Software, 2006,17(2):2403−2415 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20061201.htm [doi: 10.1360/jos172403]

[20] Song T. Research on high performance pattern matching techniques for applications of network security [Ph.D. Thesis]. Beijing: Tsinghua University, 2008 (in Chinese).

[21] Roesch M. SNORT—Lightweight intrusion detection for networks. In: Simon J, ed. Proc. of the LISA'99. Berkeley: USENIX Association, 1999. 229−238.

[22] ClamAV open source anti-virus system. http://www.clamav.org

[23] Sourdis I, Pnevmatikatos D. Pre-Decoded CAMs for efficient and high-speed NIDS pattern matching. In: Pocek KL, ed. Proc. of the IEEE Symp. on Field-Programmable Custom Computing Machines. Los Alamitos: IEEE Computer Society, 2004. 258−267. [doi: 10.1109/FCCM.2004.46]

[24] Wu S, Manber U. A fast algorithm for multi-pattern searching. Technical Report, TR 94-17, University of Arizona at Tuscon, 1994.

[25] Loikkanen M, Bagherzadeh N. A fine-grain multithreading superscalar architecture. In: Bothe B, ed. Proc. of the PACT'96. Washington: IEEE Computer Society, 1996. 163−168.

[26] Cho YH, Mangione-Smith WH. A pattern matching coprocessor for network security. In: Joyner WH, ed. Proc. of the 42nd Annual Conf. on Design Automation. New York: ACM Press, 2005. 234−239. [doi: 10.1145/1065579.1065641]

附中文参考文献:

[19] 李伟男,鄂跃鹏,葛敬国,钱华林.多模式匹配算法及硬件实现.软件学报,2006,17(12):2403−2415. http://www.jos.org.cn/1000-9825/20061201.htm [doi: 10.1360/jos172403]

[20] 嵩天.网络安全应用中高性能特征匹配技术研究[博士学位论文].北京:清华大学,2008.

嵩天(1980－),男,辽宁沈阳人,博士,副教授,CCF 会员,主要研究领域为网络内容安全,下一代互联网,计算机体系结构.
E-mail: songtian@bit.edu.cn

汪东升(1966－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为微处理器设计,海量存储.
E-mail: wds@tsinghua.edu.cn

李冬妮(1978－),女,博士,副教授,CCF 会员,主要研究领域为高性能网络技术,先进制造.
E-mail: ldn@bit.edu.cn

薛一波(1967－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络信息安全.
E-mail: yiboxue@tsinghua.edu.cn