分类号 TP000.0 学号 11060142

ＵＤＣ　 000 密级 公 开

工学硕士学位论文

**基于两级存储的正则表达式匹配技术研究**

|  |  |  |
| --- | --- | --- |
| 硕士生姓名 |  | 徐成成 |
| 学科专业 |  | 计算机科学与技术 |
| 研究方向 |  | 网络安全 |
| 指导教师 |  | 陈曙晖 副研究员 |

**国防科学技术大学研究生院**

**二〇一三年十一月**

**论文书脊**

（此页只是书脊样式，学位论文不需要印刷本页。）

（硕士学位论文题目） 国防科学技术大学研究生院

**Research of Regular Expression Matching Technology based on Two-Stage Memory**

**Candidate：Xu-chengcheng**

**Advisor：Asso.Prof. Chen-shuhui**

**A thesis**

**Submitted in partial fulfillment of the requirements**

**for the degree of Master of Engineering**

**in Computer Science and Technology**

**Graduate School of National University of Defense Technology**

**Changsha, Hunan, P.R.China**

**November, 2013**

E:\work\mydesk\page-0001.tif

目 录

[表 目 录 IV](#_Toc371582997)

[图 目 录 V](#_Toc371582998)

[摘 要 i](#_Toc371582999)

[ABSTRACT ii](#_Toc371583000)

[第一章 绪论 1](#_Toc371583001)

[1.1 研究背景 1](#_Toc371583002)

[1.2 正则表达式基础 2](#_Toc371583003)

[1.2.1 正则表达式的历史 2](#_Toc371583004)

[1.2.2 正则表达式的形式化描述 3](#_Toc371583005)

[1.2.3 正则表达式的扩展 3](#_Toc371583006)

[1.3 本文主要工作 4](#_Toc371583007)

[1.4 论文结构 5](#_Toc371583008)

[第二章 相关技术研究 7](#_Toc371583009)

[2.1 正则表达式和有限状态自动机 7](#_Toc371583010)

[2.1.1 非确定有限状态自动机 7](#_Toc371583011)

[2.1.2 确定性有限状态自动机 9](#_Toc371583012)

[2.1.3 基于FSM的正则表达式匹配 11](#_Toc371583013)

[2.2 提升匹配速度 13](#_Toc371583014)

[2.2.1 基于FPGA的实现技术 13](#_Toc371583015)

[2.2.2 基于TCAM的匹配技术 14](#_Toc371583016)

[2.2.3 基于GPU的匹配技术 15](#_Toc371583017)

[2.3 压缩存储空间 15](#_Toc371583018)

[2.3.1 合并输入字符 16](#_Toc371583019)

[2.3.2 合并单个状态的转换边 16](#_Toc371583020)

[2.3.3 合并状态间的转换边 17](#_Toc371583021)

[2.4 新型结构自动机 18](#_Toc371583022)

[2.5 本章小结 20](#_Toc371583023)

[第三章 基于两级存储的匹配思想 21](#_Toc371583024)

[3.1 匹配引擎性能分析 21](#_Toc371583025)

[3.2 状态访问概率的统计分析 23](#_Toc371583026)

[3.3 理论性能分析 27](#_Toc371583027)

[3.4 小结 28](#_Toc371583028)

[第四章 报文匹配的马氏模型 29](#_Toc371583029)

[4.1 马尔可夫链模型 29](#_Toc371583030)

[4.1.1 基本概念 29](#_Toc371583031)

[4.1.2 转移转移概率矩阵 30](#_Toc371583032)

[4.1.3 状态分布向量 31](#_Toc371583033)

[4.1.4 周期性 32](#_Toc371583034)

[4.2 可约马尔可夫链 33](#_Toc371583035)

[4.3 稳态分布 35](#_Toc371583036)

[4.3.1 稳态分布的基本概念 35](#_Toc371583037)

[4.3.2 常用的求稳态矩阵方法 36](#_Toc371583038)

[4.3.3 可约马氏链的稳态向量求解 37](#_Toc371583039)

[4.4 模型评估 40](#_Toc371583040)

[4.5 小结 43](#_Toc371583041)

[第五章 基于NetMagic的系统实现 44](#_Toc371583042)

[5.1 NetMagic开发平台 44](#_Toc371583043)

[5.1.1 NetMagic体系结构 44](#_Toc371583044)

[5.1.2 UM总体结构 45](#_Toc371583045)

[5.1.3 UM报文处理流程 47](#_Toc371583046)

[5.1.4 报文格式及地址空间分配 48](#_Toc371583047)

[5.2 系统设计实现 49](#_Toc371583048)

[5.2.1 编译器 49](#_Toc371583049)

[5.2.2 控制端程序 50](#_Toc371583050)

[5.2.3 NetMagic硬件程序 51](#_Toc371583051)

[5.2.4 表项配置模块 52](#_Toc371583052)

[5.2.5 报文采集分发模块 53](#_Toc371583053)

[5.2.6 匹配处理模块 55](#_Toc371583054)

[5.2.7 规则采集处理模块 58](#_Toc371583055)

[5.3 实验与分析 59](#_Toc371583056)

[5.3.1 实验环境 59](#_Toc371583057)

[5.3.2 实验过程 60](#_Toc371583058)

[5.3.3 性能分析 62](#_Toc371583059)

[5.4 小结 63](#_Toc371583060)

[结 束 语 65](#_Toc371583061)

[致 谢 66](#_Toc371583062)

[参考文献 67](#_Toc371583063)

表 目 录

[表1.1 常用元字符及其含义 4](#_Toc371583080)

[表2.1 DFA状态转换表 10](#_Toc371583081)

[表2.2 NFA和DFA处理的时间复杂度和空间复杂度 12](#_Toc371583082)

[表3.1不同数量规则集的自动机规模 25](#_Toc371583083)

[表3.2存储器件性能 28](#_Toc371583084)

[表5.1 NetMagic地址空间分配 49](#_Toc371583085)

[表5.2 UM内部地址空间分配 49](#_Toc371583086)

[表5.3 表项配置模块信号列表 53](#_Toc371583087)

[表5.4 报文采集分发模块信号列表 54](#_Toc371583088)

[表5.5 单个匹配处理模块的信号列表 56](#_Toc371583089)

[表5.6 转发规则定义 57](#_Toc371583090)

[表5.7 规则采集处理模块信号列表 59](#_Toc371583091)

[表5.8 部分规则集及自动机规模 61](#_Toc371583092)

[表5.9 DFA状态空间划分 61](#_Toc371583093)

[表5.10 报文转发时NetMagic端口丢包统计 62](#_Toc371583094)

[表5.11 与当前研究成果的性能对比 63](#_Toc371583095)

[表5.12 匹配性能统计（单位：Gbps） 63](#_Toc371583096)

图 目 录

[图2.1 Thompson构造法NFA生成规则 8](#_Toc371583139)

[图2.2 正则表达式r对应的NFA 9](#_Toc371583140)

[图2.3 子集合算法构造DFA 10](#_Toc371583141)

[图2.4 最小化后的DFA 11](#_Toc371583142)

[图2.5 四种基本正则表达式所对应的逻辑电路 14](#_Toc371583143)

[图2.6 索引表压缩方法示例 16](#_Toc371583144)

[图2.7 原始DFA及状态转换表 17](#_Toc371583145)

[图2.8 转换后的D2FA及对应的状态转换表 18](#_Toc371583146)

[图2.9 规则“.\*a.\*bcd”和“.\*bef”生成的NFA及相应的HFA 19](#_Toc371583147)

[图3.1 正则表达式ab+cd+对应的DFA 21](#_Toc371583148)

[图3.2 部分状态访问概率分布 24](#_Toc371583149)

[图3.3 部分状态访问概率函数 24](#_Toc371583150)

[图3.4 状态访问概率函数与自动机规模的关系 25](#_Toc371583151)

[图3.5 状态访问概率函数与报文关系 26](#_Toc371583152)

[图3.6 匹配不同报文时状态访问概率差异 26](#_Toc371583153)

[图3.7两级存储的架构图 27](#_Toc371583154)

[图4.1 规则“ab+ac”的DFA 31](#_Toc371583155)

[图4.2 式4-10的状态转换图 33](#_Toc371583156)

[图4.3 可约马尔可夫链和不可约马尔可夫链 33](#_Toc371583157)

[图4.4 可约马尔可夫链状态空间划分 34](#_Toc371583158)

[图4.5 含两个闭态集的马尔可夫链 34](#_Toc371583159)

[图4.6 调整后的转移概率矩阵 35](#_Toc371583160)

[图4.7 不可约马尔可夫链的转移概率矩阵及稳态矩阵 36](#_Toc371583161)

[图4.8 规则“^a.\*d”和“b?c”的DFA 38](#_Toc371583162)

[图4.9 部分DFA状态理论访问概率 41](#_Toc371583163)

[图4.10 匹配DARPA时DFA状态实际访问概率与理论访问概率差值 42](#_Toc371583164)

[图4.11 DFA命中率随一级存储器空间大小变化 42](#_Toc371583165)

[图5.1 NetMagic平台基本组成 45](#_Toc371583166)

[图5.2 NetMagic08硬件结构图 45](#_Toc371583167)

[图5.3 FPGA内部逻辑结构图 46](#_Toc371583168)

[图5.4 UM与CDP之间接口线路图 47](#_Toc371583169)

[图5.5 内部报文格式 48](#_Toc371583170)

[图5.6 DFA重命名算法 50](#_Toc371583171)

[图5.7 控制端程序界面 50](#_Toc371583172)

[图5.8 UM内部逻辑结构图 52](#_Toc371583173)

[图5.9 表项配置模块结构图 52](#_Toc371583174)

[图5.10 报文采集分发模块结构图 54](#_Toc371583175)

[图5.11单个匹配处理模块的结构图 55](#_Toc371583176)

[图5.12 报文匹配处理逻辑 57](#_Toc371583177)

[图5.13 规则采集处理模块结构图 58](#_Toc371583178)

[图5.14 实验环境整体部署 59](#_Toc371583179)

摘 要

在网络高速发展的同时，网络的开放性导致的安全问题也日益严峻。深度报文检测是网络安全的核心技术，深度报文检测利用预定义的规则集对报文内容进行匹配，从而识别出隐藏于报文内容中的恶意信息或协议特征。正则表达式匹配是深度报文检测的主要手段，在正则表达式匹配技术中，匹配性能和存储需求是一对相互制约的因素。吉比特网络的快速发展要求骨干网必须具备线速匹配能力，而越来越复杂的规则要求存储器必须具备足够大的容量，但存储器件一般都不同时具备大容量和高吞吐量的特点。这给正则表达式的匹配带来了巨大的挑战，必须探索新的技术以从根本上解决性能与存储的矛盾问题。

本文首次提出基于两级存储的匹配技术，一级存储器采用高速的小容量存储器解决性能问题，二级存储器采用大容量的低速存储器解决存储空间需求。通过结合使用两种存储器件，可以用较低的存储代价获得较高的吞吐量。

论文主要完成的工作有：

1. 阐明吞吐量和存储是正则表达式匹配技术的主要矛盾，进而提出基于两级存储的匹配引擎思想。通过仿真匹配实验对状态访问概率进行统计分析，实验表明状态访问概率呈Zipf分布，非常有利于两级存储的架构。
2. 利用马尔可夫链理论对报文匹配中状态迁移过程进行建模，把稳态向量作为状态理论访问概率。讨论了稳态向量的计算方法，并进行编码实现。实验数据表明该模型与状态访问概率分布特性基本吻和。
3. 基于开放式网络实验平台NetMagic，实现了本文提出的两级存储匹配引擎，并充分利用FPGA内部多RAM块的特性，实例化多个匹配线程，使系统性能线速提升。实验结果表明，该方法能在保证一定吞吐量的情况下，使存储代价大幅降低。

主题词：正则表达式；访问概率；Zipf分布；两级存储；马尔可夫链；稳态向量；NetMagic

ABSTRACT

With the high development of the Internet, the security problems caused by the Internet’s openness have become more and more serious. Deep packet inspection is one of the key technologies of network security, in order to detect the features of malicious information or protocols, it uses predefined rule sets to match with the payloads of packets. Regular expression matching is the primary means of deep packet inspection, but in which the matching performance and storage requirement contradict each other. The Gigabit Networks require line-speed matching, and the increasingly complex rule sets require memory with large capacity. However, most of the memories can not have both large capacity and high throughput. This contradiction brings a huge challenge to regular expression matching technology, new ideas and methods are needed to solve this problem basically.

This paper proposes a matching engine with two-stage memory model by the first time. The high speed of the first-level memories can guarantee a high performance, and the big capacity of second-level memories can solve the storage problem. By combining the two kinds of memories, the engine obtains a high throughput at a low cost.

The main work and contributions of this dissertation are shown as follows:

1. This thesis clarifies that the throughput and capacity requirements are the main contradiction in matching, and proposes a matching method with two-stage memory by the first time. A series of simulation experiments of matching reveals that the access probabilities of states follow the Zipf distribution, which was very opportune for two-stage memory structure.
2. The state migration process in packet matching is modeled with Markov chain, and the Steady-State vector is treated as states access probabilities in theory. Furthermore, we discuss the method to compute Steady-State vector and implement it in coding. Simulation experiments show that Steady-State vector of this model can accurately describe the state access probabilities.
3. Based on the open network platform of NetMagic, a regular expression matching system with two-stage memory architecture is realized. The multi-block RAM in FPGA is configured as multi-engine, As a result, the matching speed is accelerated multi-times. The results show that the system can reach a high throughput and the storage cost reduces significantly at the same time.

**Key Words：Regular expression matching; access probabilities; Zipf distribution; two-stage memory; Markov chain; Steady-State vector; NetMagic**

第一章 绪论

1.1 研究背景

网络技术的迅猛发展在给人们生活带来便利的同时，网络的开放性和复杂性带来的安全挑战也日益严峻。针对各种网络应用的多样化攻击层出不穷，网络安全的主要任务是阻止入侵指令、病毒、木马等有害信息在网络上蔓延，并防止机密信息通过网络泄露。入侵检测可对网络提供有效的安全防护，传统的检测方法是分析报文的结构化头部，通过分析源地址、目标地址、端口号等首部字段的信息，判断报文是否含有恶意信息。然而如今病毒、木马、垃圾邮件等信息更多地是隐藏在报文内容中，因此还需要对报文的内容进行检测，即深度报文检测(Deep Packet Inspection, DPI)。深度报文检测技术广泛应用于入侵检测与防御、病毒检测、协议识别、网络取证等系统中。

检测的方法是使用预定义的模式集对报文的负载逐个字节匹配，模式即对攻击特性或协议特征的描述。如果报文负载匹配上某条模式，就说明该报文具有相应特征。早期描述模式的方法是采用精确字符串，在过去几十年中，精确字符串的匹配从理论到实现都已经有了深入的研究，一些经典的匹配算法如AC、WU-MANBER、SBOM可实现高速的报文匹配。但是精确字符串的表达能力非常有限，攻击者也在不断地针对各种安全检测技术进行躲避和隐藏，这使得网络中的攻击和恶意代码变得越来越复杂，简单的字符串已远远不能满足描述的需求。正则表达式表达能力强、表达方式灵活，能够描述更为广泛的负载特征，已取代精确字符串成为深度报文检测的主要手段。研究领域和商业界都开始广泛采用正则表达式实现特征描述[1]。如Linux应用层协议分类器L7-filter[2]完全采用正则表达式描述协议特征，入侵检测系统Snort[3]、Bro[4]中正则表达式描述的规则已经占全部规则的一半，3Com的Tipping Point X505[5]和Cisco的IOS[6]也都开始使用正则表达式。另外在数字取证领域，正则表达式也用于解决大规模数据的快速检索问题[7-9]。正则表达式匹配已经成为信息安全领域不可或缺的重要支撑技术。

匹配之前首先要将正则表达式编译成有限状态自动机（Finite State Machine, FSM）的形式，并得到状态转换表。匹配的过程实质上就是自动机上的状态迁移的过程。每处理一个字节的报文，都需要至少一次查表，以获取下一次要访问的状态。由于需要逐个字节查表匹配，因此它是深度报文检测的瓶颈。一般网络安全设备都处于骨干链路上，因此要求匹配过程能达到线速。目前主要有两种类型的FSM：确定性有限自动机（DFA, deterministic finite automata）和非确定性有限自动机（NFA, nondeterministic finite automata）。

正则表达式的匹配需要使用有穷自动机来完成，但是NFA和DFA在实际应用中却有不同的优缺点。NFA的优点是空间复杂度低，因为NFA的状态数与正则表达式的长度成线性关系。但是NFA可能同时有多个活跃状态，每处理一个字符可能需要多次访存，因此效率较低。而DFA处理一个字符只需要访存一次，匹配效率很高。但若将每条规则编译成单独的DFA，其时间复杂度会随规则数的增加而线性提高,匹配性能也会线性下降；而将所有规则编译成一个DFA时，可能会发生状态爆炸，导致内存需求大大增加，有时甚至无法编译成完整的自动机。因此在实用中必须降低NFA处理一个字符的时间复杂度或者对DFA的内存需求进行压缩。NFA的时间复杂度由其理论模型决定，在不改变系统结构的情况下很难对其改进。因此目前的研究大多集中于对DFA的内存需求进行压缩。

目前有一些较为成熟的压缩技术，压缩率可达到90%以上。但是这些压缩方法的本质都是以时间换空间，通过增加计算量或者访存次数来减少内存空间的使用。随着网络带宽的高速增加，这种以时间换空间的方法可发挥的空间越来越小。在报文匹配中，每处理报文的一个字节都需要一次查表操作，以获取下一次要访问的状态地址。匹配的速度取决于访存次数和每次访存的时延，而对于一个给定的报文，访存次数等于报文负载长度。因此要提高匹配的效率就需要尽量减少每次访存的时延，而访存时延由具体存储器件的性能决定。所以要保证高速的匹配性能，必须使用高速的存储器件。而通常情况下，高速存储器件如FPGA片上内存块、TCAM等的容量又非常小，只能存储数千个状态。目前状态表的规模远超过高速存储器的容量，即使使用已有的压缩技术，压缩后的状态表的空间仍然远大于目前高速存储器的容量。状态表只能配置在大容量的低速存储器中，访存时延大大提高，匹配速度严重受限于低速存储器的访存时延。内存需求和匹配性能是一对相互制约的因素。另外，随着网络带宽的快速增加，10G比特的网络已经开始应用于园区网络中，若无法解决内存需求与匹配性能的矛盾，网络安全设备将成为整个网络的瓶颈。

1.2 正则表达式基础

模式匹配有单模匹配和多模匹配，单模匹配中一个模式只能匹配一个字符串，多模匹配中一个模式可以匹配多个字符串，正则表达式是一种多模匹配。正则表达式表达能力强、表达方式灵活，在文本编辑器、程序设计语言、网络安全设备等领域获得广泛应用。

1.2.1 正则表达式的历史

正则表达式最初出现于自动控制理论和形式化语言理论中。在这些领域中有对计算的模型和对形式化语言描述和分类的研究。1940年，美国神经生理科学家Warren McCulloch和Walter Pitts[10]研究出一种新的描述神经网络的数学方法，创造性地将神经系统中的神经元描述为小而简单的控制单元。20世纪50年代，数学家Stephen Kleene在此基础上，利用称之为“正则集合”的数学符号来描述此模型，引入正则表达式的概念。60代末，Ken Thompson将正则表达式应用于搜索算法的一些早期研究[11]。他将此符号系统引入编辑器[QED](http://zh.wikipedia.org/w/index.php?title=QED_(%E6%96%87%E5%AD%97%E7%B7%A8%E8%BC%AF%E5%99%A8)&action=edit&redlink=1)，然后是[Unix](http://zh.wikipedia.org/wiki/Unix)上的编辑器[ed](http://zh.wikipedia.org/w/index.php?title=Ed_(%E6%96%87%E5%AD%97%E7%B7%A8%E8%BC%AF%E5%99%A8)&action=edit&redlink=1)，并最终引入[grep](http://zh.wikipedia.org/wiki/Grep)。自此自则表达式被广泛应用到UNIX系统和类UNIX系统的各类工具中，如Perl等。

目前，正则表达式在程序语言、文本编辑、安全设备等领域有着广泛的应用。如vi、grep、sed、word等文本编辑器都提供了利用正则表达式查找替换字符串的能力，主流程序语言（如C++、Java、Perl、Python等）也都支持基于正则表达式的字符串操作，许多网络设备如防火墙、入侵检测系统等也大都采用正则表达式进行内容匹配[12]。

1.2.2 正则表达式的形式化描述

正则表达式可以用形式化语言理论的方式来描述。正则表达式由常量和算子组成，常量表示字符串集合，算子表示这些集合上的运算，算子主要包括选择、连接、闭包三种运算。给定有限字母表*Σ*，可定义常量如下：

* *Σ*上的空集Φ是常量
* *Σ*上的空串ε是常量
* 对任意字符a，若a∈*Σ*，则a是常量

*Σ*上的正则表达式的定义可递归定义如下：

* 常量是正则表达式
* 连接运算：若R、S是正则表达式集合，则RS表示{αβ| α∈R,β∈S}
* 选择运算：若R、S是正则表达式集合，则R|S表示{α| α∈R或α∈S}
* 闭包运算：若R是正则表达式集合，则R\*表示包含ε的R的最小超集，并且R\*对R上的字符串连接操作闭合。

1.2.3 正则表达式的扩展

通过常量和基本的算子可以得到所有的正则表达式，但通常为表达方便我们会

对正则表达式进行扩展。扩展后的正则表达式由普通字符和一些元字符组成，元字符具有特殊的含义。在不同的扩展中，正则表达式的语法并不完全相同，其中比较通用的是PCRE(Perl Compatible Regular Expressions，Perl兼容正则表达式)标准[13] 和POSIX标准[14]。表1.1列出了常用了部分元字符及其含义。

表1.1 常用元字符及其含义

|  |  |
| --- | --- |
| 元字符 | 含义 |
| . | 匹配任意的非换行符 |
| \* | 匹配该元字符之前的子表达式至少零次 |
| ? | 匹配前面的子表达式至少零次至多一次 |
| + | 匹配前面的子表达式至少一次 |
| ^ | 从字符串首个字符开始匹配 |
| $ | 匹配字符串的尾部字符 |
| {n} | 匹配该元字符之前的子表达式n次 |
| {n,} | 匹配该元字符之前的子表达式至少n次 |
| {n,m} | 匹配该元字符之前的子表达式至少n次，至多m次 |
| [xyz] | 字符集合，匹配括号内字符集中任意字符一次 |
| [^xyz\] | 负字符集合，匹配除括号内字符集的任意字符一次 |
| \ | 转义字符，将该元字符之后的字符转换为特定字符或者原义字符 |

例如，正则表达式^[ab](cd)\*(e|f)$表示所有以字符’a’或’b’开头，中间有0个或多个cd，最后以字符’e’或’f’结尾的字符串。包括ae、acde、acdcde… af、acdf、acdcdf… be、bcde、bcdcde… bf、bcdf、bcdcdf……

1.3 本文主要工作

DPI是网络安全领域的关键技术，广泛应用于入侵检测与防御、病毒检测、协议识别、网络取证等系统中。正则表达式匹配是DPI的核心，随着网络应用的拓展和网络链路速率的飞速提升，正则表达式正面临着日趋严峻的挑战。匹配性能和存储需求是正则表达式匹配中最重要的两个指标，但这两个指标又是相互矛盾的。模式集复杂度的增加导致状态机规模呈指数增长，进而存储需求也呈指数增长。要保证高速的匹配性能必须对状态表进行压缩，状态表压缩本质上都是以时间换空间，必然导致匹配性能的下降。当前的研究都只侧重于某一方面，提升匹配性能或压缩存储空间。正则表达式的匹配迫切需要新的技术来解决性能和存储的矛盾问题。

基于此，本文提出了基于两级存储的匹配思想，结合使用小容量的高速存储器和大容量的低速存储器，在保证一定吞吐量的性能下使存储代价大幅降低。本文具体工作如下：

1. 对正则表达式匹配的相关技术进行深入研究。深入研究当前正则表达式匹配领域的主流技术，并分类阐述。主要包括提升匹配性能、压缩存储需求、新型自动机结构三个方面，并对各类技术的优缺点进行了分析比较。
2. 首次提出两级存储的匹配思想。结合底层硬件的访存查表操作，深入研究了正则表达式匹配处理过程。为解决正则表达式匹配中匹配性能与存储需求的矛盾，首次提出了两级存储的匹配技术，并通过软件对这种技术进行仿真实验，实验表明该技术能以较低的存储代价获得较高的匹配性能。
3. 针对正则表达式匹配中状态访问概率分布的建模研究。用通用的规则集对实际网络报文进行匹配，统计匹配过程中状态访问概率的分布情况。统计结果表明状态访问概率的分布服从Zipf分布，并且与具体的规则集和所匹配的报文无关，这将有助于指导状态表项在两级存储器的部署。用随机过程中的马尔可夫链理论对正则表达式的匹配过程建模，深入研究了在大规模状态机的情况下，理论上状态访问概率的计算方法。实验表明，理论预测概率与实际状态访问概率基本吻合，说明该模型能准确地反映状态访问的分布特征。
4. 基于两级存储的正则表达式匹配技术实现。软件部分实现包括正则表达式编译器、马尔可夫链模型的计算、仿真报文的匹配处理。在软件部分的基础上，基于国防科学技术大学计算机学院的开放式网络实验平台NetMagic实现了两级存储匹配技术的原型系统，包括软件端部分和硬件端部分。软件端可使用户通过PC机将状态表配置到NetMagic的FPGA片内存储块和片外DDR2 SDRAM上，并且提供查询和修改功能。硬件端是该系统核心，提供报文接收、查表匹配、转发功能，对符合匹配条件的报文将保存报文结果，同时响应软件端的配置、查询、修改请求。

1.4 论文结构

全文内容分为五章，各章内容安排如下：

第一章介绍正则表达式匹配技术的研究背景，并简要地介绍正则表达式的基础知识。

第二章是相关技术部分，首先介绍正则表达式与有限状态自动机的关系，指出不同自动机的优缺点。然后从三个方向对当前研究技术进行归类分析：提升匹配速度、压缩存储空间、新型结构自动机。

第三章结合底层硬件的访存查询操作，对基于DFA的匹配引擎进行分析。指出单一的存储器件无法同时满足较大规模DFA的容量需求和吞吐量需求，进而提出基于两级存储的匹配引擎。仿真实验发现，在报文匹配中状态访问概率呈Zipf分布。在此实验基础上，结合具体存储器件的参数，对两级存储的匹配引擎进行定量的理论分析。

第四章是理论建模部分，通过马尔可夫链对报文匹配中状态迁移过程进行建模。使用稳态向量作为对状态访问概率的预测，重点讨论了大规模马尔可夫链的稳态向量计算方法，并进行编码实现。最后，通过仿真实验对模型进行评估，实验表明，马尔可夫链模型的预测准确性很高。

第五章是原型系统实现部分。首先介绍开发平台NetMagic的相关知识，然后从编译器、控制端程序、NetMagic硬件部分程序几个方面说明实现过程，重点介绍硬件平台各模块的实现。最后，用实际的报文流对系统的性能进行评价。

第二章 相关技术研究

正则表达式匹配是DPI的核心技术，在实际应用中为了能让机器自动进行匹配，还需要进一步将正则表达式编译成有限状态自动机的形式。有限状态自动机的基本形式有两种，非确定性有限状态自动机(nondeterministic finite automata, NFA)和确定性有限状态自动机(deterministic finite automata, DFA)。当前许多正则表达式匹配技术的研究都是围绕这两种形式的自动机进行的。

2.1 正则表达式和有限状态自动机

有限状态自动机(Finite State Machine, FSM或者Finite State Automaton, FSA)是为研究有限内存的计算过程和某些语言类而抽象出来的模型。FSM含有限数量的状态，每个状态在输入串的作用下，迁移到零个或多个状态。FSM可表示为一个有向图，是自动机理论的研究对象。

根据自动机和文法的相关理论[15]，FSM和正则表达式有等价的描述能力。通常可以将正则表达式编译成等价的FSM形式，进而执行匹配过程。FSM可形式化描述为一个五元组的形式：M=(, *Σ*,δ,q0,)。其中，

* 表示状态的非空有穷集合，中任意元素称为FSM的一个状态。
* *Σ*表示输入字母表，通常情况下为256个ASCII字符。
* δ表示状态转移函数，即。
* q0为FSM的初始状态，也称为初态，q0。
* F为FSM的终止状态集合，F是的子集，到达终止状态说明匹配成功。

自动机初始状态为q0，逐字节读入输入字符串中的每一个字符，转移函数δ

根据当前状态及读入的字符确定下一个迁移的状态。如果输入字符串读入结束时自动机的状态属于终止状态集合F，说明该自动机接受该字符串，否则为不接受。

2.1.1 非确定有限状态自动机

非确定有限状态自动机也可由五元组(,*Σ*,δ, q0,)表示，但是NFA的状态转移函数δ是多值映射的，即一个状态在接收一个输入字符时可能迁移到多个状态，NFA的非确定性就体现在这里。另外，NFA对空串输入ε也可能发生状态转移，即从一个状态出发可以不消耗任何输入字符而进行状态迁移。

将正则表达式转换成FSM的过程称为编译，将正则表达式编译成NFA的经典算法是Thompson构造算法[16]。其基本思想是：为每个基本的子正则表达式构造一个NFA，然后连接各子表达式的NFA得到整个正则表达式对应的NFA。Thompson算法构造规则如图2.1所示。

算法1：Thompson构造法

输入：字母表*Σ*上的正则表达式r。

输出：能够接受r所代表字符串的NFA。

方法：首先将构成r的每个元素分解，对每个元素，按照规则1和规则2生成NFA。之后根据r的组成规则，将生成的NFA按照规则3进行组合。

规则1 对于ε，构造NFA如图2.1(a)：

规则2 对于*Σ*的每个字符a，构造NFA如图2.1(b)：

规则3 若N(s)和N(t)为正则表达式s和t所对应的NFA，则

1. s|t的NFA的构造如图2.1(c)：
2. st的NFA的构造如图2.1(d)：
3. s\*的NFA的构造如图2.1(e)：
4. 对于(s)，使用s本身的NFA N(s)。

(a) (b)

(c) (d)



(e)

图2.1 Thompson构造法NFA生成规则

利用以上基本的Thompson构造算法得到的NFA具有以下性质：

1. NFA的状态数最多为r中出现的字符和运算符个数之和的两倍。
2. NFA的开始状态和结束状态有且仅有一个。
3. NFA中各状态对字符表中一个字符，或者有一个状态迁移，或者最多有两个ε迁移。

以正则表达式r=(a|b)\*abb为例，可构造NFA如图2.2所示：



图2.2 正则表达式r对应的NFA

2.1.2 确定性有限状态自动机

五元组(,*Σ*,δ, q0,)也可用来描述确定性有限状态自动机。与NFA不同的是，DFA的转移函数δ是一个单值函数，即DFA一个状态对于一个输入字符只能转移到一个确定的状态。另外，DFA对于空串ε输入不发生状态迁移。在形式语言与自动机理论中，正则表达式、NFA、DFA是等价的，即对任意的正则表达式都有相应的NFA和DFA与之等价。DFA一般是使用经典的子集合构造算法[17]在NFA的基础上生成的，进一步可通过最小化操作对生成的DFA进行精简。

2.1.2.1 子集合算法构造DFA

子集合构造算法的基本思想是将每一步状态转移得到的NFA状态集合视为一个DFA状态。为描述子集合构造算法，首先做以下定义：

* ε-clousure(I)：状态集合I的ε闭包，即I中任意状态s经过任意步ε转移所能到达的NFA状态集合。
* Jump(I,a)：状态集合I中任意状态s接受输入字符a时，所能跳转到的状态集合。
* Ia=ε-clousure(Jump(I,a))，即Ia定义为从I中任意状态出发，接受输入字符a，再经过任意步ε转换，所能到达的状态集合。

DFA实质上是一个二维矩阵，行代表状态，列代表输入字符，矩阵中任意一个元素dij代表DFA状态i在接受输入字符j时迁移到DFA状态。子集合构造过程如下：取NFA状态集合的链表list，初始为空。首先计算初始状态s0的ε闭包，即ε-clousure({s0})，记为N0，这个状态集合对应的就是DFA状态0，记为d0，将该N0加入到list中，并且标记为已访问。对于每一个输入字符a，计算N0a，若N0a在list中出现，则将d0a赋值为该状态集合对应的DFA状态标识；否则将该状态集加入到list中，标记为未访问，并且为该状态集映射到一个新的DFA状态。如此DFA状态表中第一行值已完全确定。依次取list中下一个未被访问的状态集Ni，标记为已访问，按照上述方法计算该状态集对应每个输入字符a的Nia，并且将新生成的但未出现的状态集标记为未访问，并且加入到list中。如此进行下去，直至list中不再有未被访问的NFA状态集。

以图2.2所示的NFA为例，初始状态s0的ε闭包为N0={0,1,2,4,7}，该集合对应的DFA状态记为d0。对于输入符号a，N0a={1,2,3,4,6,7,8}，这是一个新的NFA状态集合，记为N1，对应的DFA状态记为d1，则d0a=d1；对于输入符号b，N0b={1,2,4,5,6,7}，这也是一个新的NFA状态集合，记为N2，对应的DFA状态记为d2，则d0b=d2。以此类推，可依次得到N3={1,2,4,5,6,7,9}，N4={1,2,4,5,6,7,10}。进而可得到DFA状态转换表如表2.1所示。

表2.1 DFA状态转换表

|  |  |  |
| --- | --- | --- |
| 状态 | 输入字符 | |
|  | a | b |
| *d*0 | *d*1 | *d*2 |
| *d*1 | *d*1 | *d*3 |
| *d*2 | *d*1 | *d*2 |
| *d*3 | *d*1 | *d*4 |
| *d*4 | *d*1 | *d*2 |

DFA状态机如图2.3所示。



图2.3 子集合算法构造DFA

2.1.2.2 DFA的最小化

对于一个NFA，进行确定化后得到的DFA不是唯一的，因此所得到的DFA状态数可能不是最少的。因为在子集合构造算法中，未考虑到DFA中某些具有同样性质的可合并的DFA状态。DFA状态最小化是指构造一个等价的DFA，使其具有最小的DFA状态数。在介绍最小化算法之前，先引入等价状态的概念。在DFA中，两个状态ｓ和ｔ等价一致性和蔓延性两个条件，一致性指状态s和状态t必须同时为终态或非终态；蔓延性指对于所有的符号，状态s和状态t必须转换到等价的状态里。最小化的具体过程如下：

1. 先将所有状态划分成两个集合：接受状态集A和非接受状态集B。
2. 对集合A中的每一个状态进行状态转换，即确定集合A中每一个状态在经过符号表中同一个符号转化后的状态在哪个集合中，如果有的状态经过转换后和别的状态不在同一个集合，那么就将其从原来的集合中移除，建立一个新的集合。这个新集合在不断地加入状态，如果加入的状态经过符号表中同一个字符转换后所得到的状态和新集合中已有元素通过同一个符号转换后得到的状态在相同的集合中，再用其它符号用相同的方法划分集合。
3. 经过步骤b)后，对任意一个集合，对符号表中每一个符号进行与步骤b)相同的操作，不断地将集合划分成更小的集合，直到任意集合中每一个状态经过符号表中任意相同符号转换后得到的状态在相同的集合中。

最小化的过程就是一个集合由大到小不断的划分过程，直至每一个集合中各状态是等价的。以图2.3所示的DFA为例，初始划分由两个子集组成，即{*d*0, *d*1, *d*2, *d*3}、{ *d*4}。对于字符a，，但是对于字符b，，而，即该状态集合对字符b会转移到不同的状态集合。因此，可将状态集继续划分为{*d*0, *d*1, *d*2}、{*d*3}、{d4}。进一步考察{*d*0, *d*1, *d*2}，，但对于字符b，有，而，因此可将{*d*0, *d*1, *d*2}进一步划分为{*d*0,*d*2}、{*d*1}。*d*0和*d*2是等价状态，不可再划分。最后得到的状态集为{*d*0, *d*2}、{*d*1}、{*d*3}、{*d*4}。现在将状态*d*0作为{*d*0, *d*2}的代表，将状态*d*2删除，并将指向*d*2的转换都指向*d*0,由此得到最小化后的DFA如2.4所示。



图2.4 最小化后的DFA

2.1.3 基于FSM的正则表达式匹配

高速的正则表达式匹配系统一般都基于NFA或DFA，自动机的相关理论可以证明正则表达式、NFA、DFA具有等价性。为实现正则表达式的匹配，需要将自动机配置在实际的硬件匹配引擎中，因此实际应用中需要考虑自动机数据结构的硬件资源需求和匹配吞吐量的问题。匹配引擎通常有两种类型的体系架构，以内存为中心的架构和基于FPGA的架构。在以内存为中心的架构中，需要考虑的问题主要是自动机的存储需求及匹配处理过程中的带宽需求。在基于FPGA的实现中，主要关注时钟频率的问题。两种类型的体系架构中，硬件资源的需求都依赖于自动机规模的大小。

从2.1.1中NFA的生成规则可以看出，NFA状态数与正则表达式中字符数成正比，通常情况下NFA的状态数都很少，因此NFA的空间复杂度很低。在匹配处理中，每当访问到一个终态时就说明匹配成功。由于每个状态对同一个输入字符可能有多个状态迁移，因此可能同时有多个状态被同时激活，我们称其为活跃状态集。每处理一个字符可能需要多次访存操作，在顺序访问存储器中，活跃状态集的大小反映了存储带宽的需求。在最极端的情况下，NFA中所有的状态都处于活跃状态，处理一个字符需要访问所有的状态表项，因此NFA处理的时间复杂度很高。所以基于NFA的匹配通常要考虑降低处理一个字符的时间复杂度。通常一个正表达式集合对应的NFA并不是唯一的，对于一个给定的NFA，可以通过压缩公共前缀来构建最优NFA。可以将NFA生成DFA的子集合算法应用于最优DFA的构建[18]。压缩后的NFA主要有两个优点，一是自动机规模更小，NFA状态数和转换边数同时变少；二是活跃状态集的上界变小，因此带宽需求也随之减少。

在DFA中，任意一个状态对字符中每一个字符都只有一个状态迁移。因此，基于DFA的匹配过程中，活跃状态数始终是1，带宽需求也是固定的。通常，基于NFA的匹配可以在基于FPGA的架构上得到很高的吞吐量，基于DFA的匹配更适合配置在以内存为中心的架构上。但另一方面，由于重复限定字符、通配符、多条正则表达式的交互作用等影响，NFA生成DFA的过程可能发生状态爆炸，导致DFA的状态数远远多于相应NFA状态数。对于一个含有*n*个状态的NFA，其DFA理论最大状态数可达2*n*，因此，基于DFA的匹配需要重点考虑DFA的存储需求问题。将*m*条长度为*n*的正则表达式对应的NFA和DFA在匹配处理时所需的空间复杂度和处理一个字符所需的时间复杂度如表2.2所示。

表2.2 NFA和DFA处理的时间复杂度和空间复杂度

|  |  |  |
| --- | --- | --- |
|  | 时间复杂度 | 空间复杂度 |
| NFA |  |  |
| DFA |  |  |

正则表达式匹配引擎的实现主要分为基于NFA和基于DFA两大类。基于NFA和DFA实现各有优缺点，基于NFA的匹配时间复杂度高，空间复杂度低，因此基于NFA的实现主要解决提升匹配性能的问题。而基于DFA的实现中，状态规模的指数增长使存储需求也呈指数增长，因此基于DFA的实现主要解决压缩存储需求的问题。另外，针对DFA状态爆炸带来的挑战，很多学者提出了新型的自动机结构，并且在硬件上设计相应的体系结构以加速匹配过程。下面从提升匹配速度、压缩存储需求、新型结构自动机三个方向对当前研究成果进行分析。

2.2 提升匹配速度

正则表达式的匹配过程实质上是一系列的查表操作过程。在FPGA的架构中，提升匹配速度需要减少存储空间及优化电路，在以内存为中心的架构中，提升处理速度主要靠使用高速的存储器件如TCAM来加速查表过程或使用加速引擎GPU来加速查找过程。

2.2.1 基于FPGA的实现技术

现场可编程门阵列(Field Programmable Gate Array, FPGA)在集成电路领域是一种半定制电路形式。FPGA内部逻辑资源丰富，且内部逻辑功能可通过编程来定制，并且具有天然的并发性。非常适宜实现基于NFA的正则表达式的匹配，在基于FPGA的匹配中，研究人员围绕提升匹配速度和节省硬件资源等方面做了大量研究。

Sidhu R等在2001年首次提出了基于FPGA的NFA映射算法[19]，主要思想是用NFA的逻辑电路来表示正则表达式的四种基本操作，如图2.5所示。该方法用一个触发器来表示一个NFA状态，并且用一个字符比较器的输出表示当前状态的输入字符，两者输出作“与”操作完成转移功能。该方法并未考虑到正则表达式复杂语法（如数量限定符）等的有效实现，另外，匹配引擎采用离散的字符比较器，硬件资源利用率过低。

在此基础之上，Cho Y H[20]、Clark C R[21]、Badran T F[22]等都分别提出每个时钟周期可处理多个输入字符的并行处理自动机。Korenek J从节省FPGA资源的角度出发，提出了一种新的架构：NFA-Split[23]，该方法将被同一输入字符所激活的状态的集合称为有冲突的状态集，进而将NFA所有状态分有有冲突状态集和无冲突状态集，这两类状态集分别映射到NFA单元和DFA单元。该架构可将NFA转移表中49%以上的部分转存到内存中，从而节省大量FPGA中查找表与触发器资源。

基于FPGA的技术尽管可以修改特征库，但是FPGA内部电路的更新需要编译、综合、布局、烧入过程，更新十分缓慢。因此基于FPGA的实现不适宜使用在需频繁更新特征库的应用场合。

   
 (a) 单字节匹配逻辑 (b) N1|N2匹配逻辑

(c) N1N2匹配逻辑 (d) N1\*匹配逻辑

图2.5 四种基本正则表达式所对应的逻辑电路

2.2.2 基于TCAM的匹配技术

TCAM (ternary content addressable memory)是一种三态内容寻址的[存储器](http://baike.baidu.com/view/87697.htm)，主要用于快速查找ACL、路由等表项。

TCAM表内所有条目都可以并行访问，并且访问性能不会随条目数增多而减弱。考虑到TCAM在并行访问方面的优点，研究人员开始将TCAM技术应用于正则表达式匹配中[24-26]。基于TCAM的匹配技术将TCAM和SRAM相结合，将转换边存储在TCAM，而将下一跳状态存储在SRAM中。TCAM的位掩码功能使得一个TCAM的表项可存储多个转换，进而达到降低存储空间的目的。

TCAM的关键问题是容量过小，只能适用于自动机规模很小的场合。为了解决在较小的TCAM中存储尽量多的状态表项问题，Chad R. Meiners [27]提出了转换边共享和表合并技术，显著地降低了存储空间。其中，转换边共享技术将具有相同源状态和相同目的状态以及具有相同目的状态和相同输入字符的转换边合并，从而降低整个DFA状态表的空间消耗。另外，为提高匹配性能，使用了改进的K步DFA，每次消耗K个字符。在较为极端的情况下，理论分析的性能可达10G。

即使采用一些改进的技术，依然无法解决TCAM存储空间过小的问题，基于TCAM的匹配技术只能适用于小规模的模式匹配。另外TCAM的成本过高，单位存储代价高于普通的SRAM，且耗能也远远高于SRAM。

2.2.3 基于GPU的匹配技术

GPU(Graphic Processing Unit)是有巨大计算能力的多线程、高存储带宽的多核处理器。多线程流水查找GPU的纹理存储器，可以有效地隐藏时延。基于GPU实现的正则表达式匹配技术中[28-30]，多个线程并行执行，每个线程独立完成各个的任务，并将匹配结果发送到GPU全局存储器。

Vasiliadis等设计实现了基于GPU的字符串模式匹配引擎Gnort[31]，并且对其进行扩展，以支持正则表达式的匹配。每个正则表达式编译成一个DFA，匹配时每个GPU相当于一个DFA匹配引擎，各GPU流水处理待匹配的报文。当匹配命中时，GPU将相应的结果写入到全局缓冲区中。实验表明，扩展后Gnort的匹配性能可达16Gbps。

Smith等[32]设计实现了一个基于GPU的正则表达式匹配引擎，并且给出相应的优化策略。实验表明，该匹配引擎的性能比Pentium 4 CPU快6至9倍。

Naghmouchi等[33-34]对自然语言处理、搜索引擎等领域中模式匹配的应用进行研究，根据这些领域中匹配模式少、模式复杂度低的特点，对基于GPU的DFA匹配引擎深入研究，并进行性能优化，实验证明匹配性能可达到55.3Gbps。

虽然基于GPU的正则表达式匹配普遍能获得较高的性能，但是基于GPU的实现必须要有并行开发平台如CUDA等环境的支持。另外基于GPU的实现也无法解决状态爆炸的问题。

2.3 压缩存储空间

基于FSM的正则表达式匹配中，NFA的处理时间复杂度远远高于DFA，并且这种复杂度是由其理论模型决定的，在不改变系统体系结构的前提下，很难对其进行改进。而DFA的处理时间复杂度为O(1)，因此当前很多匹配技术都是围绕DFA展开的。但是在模式集比较复杂时，生成的DFA可能会发生状态爆炸，状态表空间呈指数增长，这给状态表的存储带来了很大的挑战。当前基于DFA的研究都是状态表的压缩，使压缩后的DFA满足存储需求。DFA状态表实质是一个二维矩阵，行代表状态，列代表输入字符，表中的元素代表转换边。因此DFA的压缩技术大体可分为三种途径：合并状态、合并输入字符、合并转换边。

2.3.1 合并输入字符

合并输入字符的基本思想是，将输入字符表进行集合划分，使得划分后各集合内任意两个字符是“等价”的。所谓两个字符*ci*和*cj*等价，指对DFA内任意一个状态*s*，有。陈曙晖等[35]提出合并字符进行压缩的方法，在正则表达式编译成DFA之前，对输入符号进行集合交割的预编码，以减少输入符号种类，从而达到压缩存储空间的目的。Becchi等[36]也提出了类似于子集合构造的分类方法， Kong等[37]在此基础上，进一步对其进行扩展。Ficara等[38]提出state-char的编码方法，从而对现有的编码空间进一步压缩。

2.3.2 合并单个状态的转换边

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | d |
| d2 | d2 | d2 | d3 |

 DFA的结构实质上是一个有向图，通常情况下，对于一个指定的状态，对字符表中的所有字符，它只会跳转到少量的几个状态。即对一个指定的状态，通常它下一步可迁移到的状态数目远小于|*Σ*|，其中|*Σ*|为*Σ*中字符数目。因此，每个状态的表项中都存在一定的冗余。基于这种规律，Becchi等[36]提出一种增加索引表的方法来消除转换冗余，使用一个目标状态表和一个索引表来表示原始的DFA状态转换表。其中目标状态表中每个状态的表项是变长的，用于存储该状态可以迁移到的状态ID，对于相同的目标状态只存储一次。索引表的表项大小是固定的，但是索引表中不再存储具体的目标状态ID，而是存储目标状态在目标状态表中相应表项的索引。以图2.6(a)所示的DFA为例，假设输入字符集为。对状态1而言，其原始状态转换表项如图2.6(b)所示。在Becchi的方法中，对状态1在目标状态表中只需要存储两个状态的ID，如图2.6(c)所示，在索引表中，每条迁移只需要1bit即可存储索引，如图2.6(d)所示。

(b)

|  |  |
| --- | --- |
| d2 | d3 |

|  |  |  |  |
| --- | --- | --- | --- |
| a | B | c | d |
| 0 | 0 | 0 | 1 |

(c)

1. (d)

图2.6 索引表压缩方法示例

对一个给定的状态，其迁移的目标状态最多有|*Σ*|个，因此索引表中每个索引只需要的存储空间。对于有n个状态的DFA，索引表所需的存储空间为。当原始状态转换表中存在大量冗余时，这种方法可大大压缩DFA的存储空间。

实际应用中，多数状态往往对大部分输入字符都迁移到初始状态，只对少量的字符跳转到其它状态。因此这种结合目标状态表和索引表的方式可大量压缩存储需求。

2.3.3 合并状态间的转换边

在DFA中，不同状态间也可能存在大量的冗余项，以图2.7(a)所示的自动机为例，其对应的原始状态转换表如图2.7(b)所示。由2.7(b)可以看出，DFA中各状态对相同的输入字符迁移到的状态基本都是相同的，即状态表中每一列各元素值基本相同，存在大量冗余。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 状态 | 输入字符 | | | |
|  | a | b | c | d |
| 0 | 1 | 2 | 0 | 3 |
| 1 | 1 | 2 | 0 | 3 |
| 2 | 1 | 2 | 4 | 3 |
| 3 | 1 | 2 | 0 | 3 |
| 4 | 1 | 2 | 0 | 3 |

 (a) 原始DFA (b) 原始状态转换表

图2.7 原始DFA及状态转换表

为此Kumar等[39]引入一种称为输入延迟的DFA(D2FA)方法，将DFA中一个状态的多条边用单个缺省边代替，从而降低边的存储空间。基本思想是，如果两个状态*s1*和*s2*对于部分相同的输入字符具有相同的目标状态，则可以删除*s1*的表项中所有与*s2*相同的转换表项，然后再从*s1*引一条到*s2*的转换边。在匹配过程中，若在某个状态中没有找到对应输入字符的目标状态，则沿着缺省转换边跳到缺省状态，再以缺省状态为当前状态查找对应该输入字符的目标状态。图2.7所示的DFA经输入延迟变换后，得到的D2FA及相应的状态转换表如图2.8所示，其中(a)中的虚线表示缺省转换边。在本例中，原始DFA需要存储20条转换边，而输入延迟D2FA只需要存储9条转换边。

D2FA虽然可以极大地降低了状态表的存储空间，但是缺省转换的引入增加了处理一个字符的时间复杂度，对于一个输入字符，可能要经过多次缺省转换才能到达目标状态，这大大增加了DFA处理的时间复杂度。Ficara等[40]提出另一种消除状态间相同转换表项的方法，但并不能完全消除冗余。另外，需要更新操作来更新全局状态表项，频繁的更新操作会降低实际的匹配效率。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | 输入字符 | | | | |
|  | a | b | c | d | 缺省 |
| 0 | 1 | 2 | 0 | 3 |  |
| 1 |  |  |  |  | 0 |
| 2 |  |  | 4 |  | 0 |
| 3 |  |  |  |  | 0 |
| 4 |  |  |  |  | 0 |



(a)D2FA (b) D2FA的状态表

图2.8 转换后的D2FA及对应的状态转换表

以上两种方法主要是合并状态间相同转换表项之间的冗余，Becchi等提出状态合并[36]的方法用以压缩状态间不同转换表项的冗余。状态合并的基本思想是，若两个状态的转换表项中有相同的目标状态，则不论它是否对应相同的输入字符，都将这两个状态合并起来，行成一个混合状态。合并后的混合状态将关联原始状态的两个索引表，以及合并后的状态转换表。状态合并方法主要优点在于，它只要求被合并的状态有相同的目标状态，而不要求该转换对应相同的输入字符。另外，合并进行的过程中，将不断地创造更多的合并机会。不足之处是该方法需要建立一张完全图，当DFA的状态数目为时，需要的额外空间。

2.4 新型结构自动机

2.3节的技术可以很好地对DFA的存储需求进行压缩，一些较为成熟的压缩技术的压缩率甚至可达90%以上。但这些压缩方法普遍还存在两个问题，首先，DFA状态数是呈指数级增长的，而当前的压缩技术都只能在DFA的基础上进行线性的压缩，线性的压缩技术无法从根本上解决状态空间指数增长的问题。其次，当前的压缩技术都是在生成DFA的基础上进行的。而对于一些较为复杂的规则集，在NFA编译生成DFA的过程中会发生状态爆炸，使得DFA的状态空间远远大于当前系统的存储空间，在这种情况下甚至无法生成完整的DFA，更无法进行后续的压缩工作。因此，更为先进的压缩技术需要从自动机本身的结构入手，从根本上解决DFA状态爆炸的问题。

DFA的状态爆炸主要由两种情况引起，一种是带重复次数限定符如“{m,n}”或“{n}”的规则，编译时需要考虑各种可能的排列组合情况，有时即使编译单条这种类型的规则也可能产生状态爆炸。另一种是带通配符如“.\*”的规则，在编译这种规则时将完整地复制其它规则编译成的自动机。编译单条此类型的规则不会产生状态爆炸，多条此类型的规则一起编译时会大大提高自动机的复杂度。基于此，Yu等[41]提出一种朴素的分类方法，将表达式划分成多个不相交的集合，使得每个集合内各表达式相互影响最小，然后将每个集合单独编译成一个DFA。文中根据表达式所能生的DFA的规模，将表达式分为5类，并且提出2条改写方法来降低DFA的规模。但是改写后的表达式只能适用于非重叠匹配，因此使用范围有限。

新型结构自动机中，比较典型的是Becchi提出混合自动机(HFA, hybrid-FA) [42]，它根据设定的阀值将部分NFA状态转换成DFA，在减少内存需求的同时获得近似于DFA的匹配性能。HFA的基本思想是在生成NFA后并不将其完全转换成DFA。而是在子集构造算法中根据当前NFA状态集合中各个NFA状态的类型来决定是否将其确定化，如果某个NFA状态是以上两种类型生成的，则不将这个NFA状态确定化。含有未确定化NFA状态的DFA状态称为边界状态，边界状态可以作为DFA的一部分或者NFA的一部分。生成的HFA包含一个头部的DFA，若干边界状态，每个边界状态指向若干NFA状态，图2.9是HFA的一个简单示例。

 (a) (b)

图2.9 规则“.\*a.\*bcd”和“.\*bef”生成的NFA及相应的HFA

子集构造算法在NFA状态1处不再继续生成DFA。图2.9(b)的HFA中，虚线圆表示的状态为NFA状态，虚线箭头表示NFA的状态转换，实线圆表示的状态为DFA状态，其中DFA(0,1)为边界状态。当在边界状态时，NFA状态1为NFA部分的活跃状态。此时输入字符*b*，DFA的活跃状态为DFA(0,5)，NFA活跃状态为NFA状态1和NFA状态2。实际上在表达式比较多的时候，生成的HFA通常含有多个边界状态，每个边界状态可能指向多个NFA。HFA有以下特征：1）初始状态是DFA状态；2）在访问边界状态时，相应的NFA状态才会被激活；3）没有从NFA状态到DFA状态的转换。在匹配的过程中，DFA部分始终只有一个活跃状态，而NFA活跃状态可能会增加、减少，甚至消亡。

HFA可有效地控制DFA状态数的增长，但是在匹配过程中，若边界状态经常被激活，则匹配性能会大幅下降。

2.5 本章小结

正则表达式匹配是深度报文检测的核心技术，本章对正则表达式匹配领域各相关研究方向进行归类分析。首先介绍基本理论知识，正则表达式匹配在实现上是以自动机的方式进行的，介绍了正则表达式生成自动机的过程，比较分析了不同类型自动机NFA和DFA的优缺点。NFA空间复杂度低而处理时间复杂度高，适合在FPGA上实现；DFA空间复杂度高，时间复杂度为，适用于以内存为中心的系统。然后从提升性能、压缩存储、新型结构自动机三个方向对当前的研究成果进行总结。提升性能主要依靠使用硬件如FPGA、GPU、TCAM等来加速查表匹配的过程，但硬件实现大都面临着扩展性的问题。压缩存储空间主要针对DFA，较为成熟的压缩技术可达到90%以上的压缩率，但线性的压缩技术无法从根本上解决DFA状态爆炸的问题。新型结构的自动机大都结合NFA和DFA的优点，在保证匹配效率的同时，能大大减少DFA的空间需求。

第三章 基于两级存储的匹配思想

前两章主要介绍了正则表达式匹配的基础知识，并且从当前各研究方向对匹配技术进行深入分析。本章首先结合底层硬件的访存查询操作，对正则表达式的匹配处理性能进行分析，阐明吞吐量和存储是正则表达式匹配的核心问题。进而提出基于两级存储的匹配引擎思想，结合使用小容量的高速存储器和大容量的低速存储器实现状态表的两级配置，以解决吞吐量和存储的矛盾问题。两级存储的匹配技术需要保证高级存储器中状态表项的高访问概率，通过实验统计发现，报文匹配过程中状态访问概率呈Zipf分布，这对两级存储的状态表配置非常有利。然后用随机过程中的马尔可夫链理论对报文匹配过程进行建模，使用稳态向量对状态访问概率进行预测。重点讨论了大规模状态下稳态向量的求解方法，最后对马尔可夫链模型评估，实验表明马尔可夫链的概率预测具有很高的准确性。

3.1 匹配引擎性能分析

基于DFA的正则表达式匹配面临的主要挑战是内存需求和匹配性能的矛盾，高速存储器容量不够大，无法存储整张状态表，且价格非常昂贵；而大容量的存储器匹配性能又不高，无法满足骨干网络对吞吐量的要求。下面通过一个简单的示例对自动机的匹配速度进行分析，然后对主流的存储器进行对比，最后提出我们的设计思想。

以正则表达式(ab+cd+)为例，编译后生成的自动机如图3.1所示。



图3.1 正则表达式ab+cd+对应的DFA

图中省略到状态0的转换边，假设待检测的字符串为“cabca”。初始时位于状态0，对第一个输入字符a，访问状态0的转移表项，转移到状态0；对第二个输入字符a，访问状态0的转移表项，转移到状态1；依次类推，对最后一个输入字符a，访问状态3的转移表项，转移到状态1，匹配不成功。在这次匹配过程共需5次访存操作，每处理报文的一个字节都需要访问一次存储器。下一次要访问的地址取决于上一次访问的地址和将要处理的字符。连续两次的访存操作是相关联的，因此无法使用访存的流水线技术。因此在给定报文的情况下，匹配吞吐量取决于每次访存的时间。要提高匹配性能就必须减少每次访存的时延，最好的方法是使用更加高速的存储器来存储状态表。

然而随着规则数目的增加及复杂化，将多条正则表达式编译成一个自动机的内存需求是很大的。尤其是当表达式中含有大量的通配符“.\*”及重复次数限定符“{*n*}”时，自动机的状态数可能使系统内存无法承受。尽管有很多压缩技术可对状态表进行压缩，但线性的压缩无法从根本上解决问题。虽然可以采用新型的自动机如HFA对状态数进行控制，但是状态空间仍然远超过目前高速存储器的容量。

不同类型的存储器在性能、价格、容量等方面差异很大。例如SRAM有很高的访问速度，目前已经可以达到250MHz，但是它的价格昂贵且容量不大，一般只有数M字节，只能存储几千条状态的转移表项。DRAM有高吞吐量高容量的特点，但是它有一个很长的启动时间，一次访存需要花费上百个周期才能获得结果。在正则表达式的匹配处理中，连续两次访存之间是相互关联的，下一次访存的地址取决于上一次访存的结果，因此流水线技术和DRAM的高吞吐量无法得到充分利用。FPGA内部嵌入有高吞吐量的片上存储块，并且这些存储块可通过双端口并行访问，其访存时间只有1~2ns，非常适合于连续依赖的访存。但是FPGA片上存储块的容量很小，一般都不超过10M位，只能存储数百条状态的转移表项。

虽然FPGA的片上存储块容量很小，但是它的最大频率已经超过500MHz，即使综合后频率可能只有400MHz，但仍然远高于DRAM和SRAM。以Altera Stratix II EP2S180为例，它有9M位的RAM内存块，并且每个内存块可以配置成双端口访问的模式。假设综合后的频率是400MHz，仅含一个扫描模块的匹配引擎的吞吐量可以达到400MHz\*2\*8=6.4Gbps。实际上很容易实例化多个匹配引擎来处理多条报文流，每个匹配引擎固定访问一个RAM块，并行处理可使匹配性能达到线速提升。并行化后可以达到超过20Gbps的吞吐量，非常适合于高速正则表达式匹配。

通过以上分析可知，使用单一的存储器件无法同时满足容量和性能的需求。在计算机体系结构中，Cache是用于CPU和主存之间的高速小容量存储器。它存储那些经常被访问的程序或数据的副本供CPU访问，以减少访问主存的时延。只要访存尽可能多地在Cache中进行，CPU整体访存性能就接近于Cache访存性能。高速的Cache可以满足高速的访存需求，大容量的主存可提供足够的存储空间。这与正则表达式面临的问题比较类似，借鉴于此思想，我们考虑对匹配引擎采用多级的存储架构。将经常被访问的状态表项配置到小容量的高速存储器中，将整个DFA状态表项存储在大容量的低速存储器中，通过适当的分配可以达到性能与空间的完美结合。

两级存储的体系结构，关键是要保证访存过程中高速存储器的命中率。在一般的计算机系统中，由于局部性原理，程序会趋于使用最近访问过的指令或数据。因此可以使用先进先出，最近最少使用等替换算法，不断地替换Cache块中的内容，以使Cache有很高的命中率。但是在深度报文检测中，替换一个状态的转移表项需要进行次的写存储器操作，其中为字符种类数。替换代价非常高，因此高速存储器中的状态表项必须固定配置，并且在匹配过程中不能进行替换操作。因此，对两级存储的报文匹配而言，最有利的情形是在整个文匹配过程中，状态的跳转集中在少量状态中。从概率的角度讲，即少量状态具有很大的访问概率。如果报文匹配中状态访问满足这样的特点，我们就可以把这些经常被访问的状态配置到高速存储器中。但是在深度报文检测中，报文的内容却是完全随机的，无法预知下一个要处理的字节内容，会转向哪个状态。选择那些经常被访问的状态存储到高速存储器中是提高性能的关键。

3.2 状态访问概率的统计分析

3.1节提出了两级存储结构的匹配思想，关键是要保证高速存储器中状态表项具有很高的命中率，网络报文的随机性导致我们无法对引擎的访存情况进行预测。一般情况下，报文匹配中各状态被访问的次数及访问概率都不一样，对匹配引擎而言，最理想的情况是状态的访问概率满足Zipf定律[43]。Zipf定律是哈佛大学的语言学家George于1949年发表的，它是一个统计型的分布规律，非常类似于“20/80原则”，描述少数个体占有大量资源的一种规律。例如，在英文文献中极少数单词经常被使用而绝大部分单词极少被使用，世界上极少数人占有绝大部分财富，网络中极少数网站访问频率极高。

如果报文匹配中状态访问满足Zipf定律，则可以根据高速存储器的容量大小，将部分高访问概率的状态表项配置到高速存储器中，从而使整体达到高速的匹配性能。基于这个假设，我们通过实测的报文，对流行的L7-filter规则集进行测试。L7-filter是Linux Netfilter的一个扩展模块，主要功能是基于数据流的应用层内容过滤，它采用正则表达式来描述协议数据特征。L7-filter共有114条协议规则，并且很多规则含有较多的”.\*”及“{n}”字段，如一条简单的bgp协议规则”\xff{16}..?\x01[\x03\x04]”，这些字段的出现很容易导致DFA状态爆炸，甚至无法生成DFA。为保证实验可行性（即能生成完整DFA），并没有编译规则集中所有规则，而是随机选取部分规则进行编译。

测试网络报文选取DARPA 1999[44]的测评数据包，DARPA测评包是MIT林肯实验室在1999年设计的测评数据包，涵盖五大类典型攻击方式，是业界及研究领域广泛认可的测评数据集。DARPA1999测评包共五周，每周有5个外网数据包和5个内网数据包。在本系统中选择使用外网数据包，外网数据包平均大小300M左右。

首先随机选取30条规则，编译得到含71226个状态的DFA。采用第一周周一的外网数据包进行测试（共316243KB，1362869个报文），并统计各状态的访问概率。统计发现，只有46个状态的访问概率在万分之一以上，有98%的状态在这次报文匹配中从未被访问。为表示方便，我们只选取了访问概率最高的100个状态绘图，并对状态的访问概率按大小排序，如图3.2所示。由于不同状态访问概率差异过大，对图3.2中纵坐标的访问概率取自然对数。



图3.2 部分状态访问概率分布

由图3.2可看出，DFA状态表项的访问极其集中，极少数状态有很大的访问概率，绝大部分状态访问概率几乎为0。统计发现，仅有1181个状态在这次报文匹配中被访问到。由实验数据进一步可以得到状态访问概率函数，如图3.3所示，仍然取访问概率最大的100个状态。



图3.3 部分状态访问概率函数

由图3.3可以看出，前10个状态的访问概率之和已占到95%，是典型的Zipf分布。并且随着状态数的增加，访问概率增加的幅度逐渐变缓。

图3.2和图3.3只能说明，在一定规模的自动机下，对固定的报文段进行匹配，状态的访问概率呈现明显的Zipf特征。还需要进一步探究在不同规模的规则集及匹配不同报文段时状态访问概率的分布，即需要进一步判断状态访问分布是否独立于规则集和报文。

首先研究不同规模自动机下，状态访问概率分布情况。为此分别随机取10、20、30条规则进行编译，得到状态机的规模如表3.1所示。

表3.1不同数量规则集的自动机规模

|  |  |  |
| --- | --- | --- |
| 规则数量 | NFA | DFA |
| 10 | 249 | 362 |
| 20 | 405 | 932 |
| 30 | 942 | 71226 |

分别用以上DFA自动机对DARPA99第一周周一外网报文进行匹配，统计状态访问概率并进一步得到状态访问概率函数如图3.4所示。



图3.4 状态访问概率函数与自动机规模的关系

由图3.4可以看出，在不同规模的自动机下，状态访问概率函数的走势是一样的。极少数状态有很高的访问概率，访问概率最高的前10个状态的概率之和达到95%-99%。状态访问服从Zipf分布，并且与自动机的规模无关。

为验证状态访问概率分布与具体匹配的报文无关，对同一个规则集取不同的报文进行匹配，统计访问概率分布。规则集选取上文用到的30条规则，报文选择第一周所有的外网报文，共5个数据包。统计得到的状态访问概率分布函数如图3.5所示。



图3.5 状态访问概率函数与报文关系

由图3.5可看出，在匹配不同报文时，状态访问概率分布都呈现Zipf分布特性，说明状态访问概率的分布与具体匹配的报文无关。图3.4和图3.5可以说明，状态的访问概率分布呈Zipf特性，且与自动机规模无关，与具体匹配的报文无关。



图3.6 匹配不同报文时状态访问概率差异

另外还需要进一步验证状态访问概率的稳定性，即对一个给定的自动机，在不同的报文匹配中，高访问概率的状态表项都是一样的；或者说同一个高访问概率的状态表项，在不同匹配中访问概率差别不大。若满足这样的分布，则说明对给定自动机，状态访问概率呈Zipf分布，且高访问概率状态表项是固定的，即可以将这些表项固定配置到高速存储器中。为此，用30条规则集对DARPA99第二周的五个外网数据包进行匹配，并按状态标识统计访问概率情况。为表示方便，只选取30个高访问概率的状态，统计在不同匹配报文下这些状态的访问概率变化情况，如图3.6所示。为方便绘图，对高概率状态的标识号重新分配，即横坐标所表示的0-29。

由图3.6可看出，高访问概率的状态，在匹配不同报文时的访问概率差异很小。说明对于给定自动机，高访问概率的状态表项是固定的。这对状态表项的配置非常有利，我们只要能够想办法确定那些访问概率较高的状态，就可以利用两级存储的匹配技术获得很高的性能。

3.3 理论性能分析

上一节的统计分析表明，基于DFA的报文匹配中，状态访问服从Zipf分布，且独立于规则集和报文。本节我们结合具体存储器件的参数，从理论上简单分析基于两级存储的匹配技术可以达到的性能。

在我们的设计中，选择FPGA片上存储器作为一级存储器，因为片上存储器速度很快，可以达到400MHz。另外主流的FPGA都有若干个可以同时被访问的存储块。如果将相同的内容存储在不同的存储块中，访存操作可以在不同的存储块中并发进行，吞吐量可以提高数倍。SRAM访存性能很高，并且SRAM控制电路简单，通过多个SRAM集中控制足以容纳整个状态表，因此可选择SRAM作为二级存储器。

基于两级存储的匹配引擎结构如图3.7所示，其中虚线框内的部分代表FPGA部分，RAM为FPGA片上存储块，SRAM为二级存储器，如果需要可以在此基础上增加DRAM为三级存储器。RAM中状态表项是固定配置的，因此RAM不需要和SRAM连接。多块RAM的目的是并发访问，提高吞吐量；多块SRAM目的是解决状态表存储空间问题。匹配过程中，如果要查找DFA状态，则优先访问RAM，RAM未命中时再访问SRAM。



图3.7两级存储的架构图

实验中各级存储器件的访存时间按主流存储器的频率计算，如表3.2所示。其中FPGA型号Altera Stratix II EP2S180，频率为500MHz，综合后频率按400MHz计算，访存周期为1/2（双端口）。SRAM型号K7N641845M，频率250MHz，访存时间为3个周期。

表3.2存储器件性能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 存储器 | 频率 | 访存周期 | 访存时间(ns) | 吞吐量(Gbps) |
| FPGA | 400MHz | 1/2 | 1.25 | 6.4 |
| SRAM | 250MHz | 3 | 12 | 0.67 |

假设FPGA片内RAM块的命中率按90%计算，则系统整体匹配性能为(6.4\*0.9+0.67\*0.1)\*9=52Gbps，即使一级存储器的命中率只有80%，总体的匹配性能也能达到46Gbps，可以满足许多场合的应用。

3.4 小结

本章结合底层硬件的访存查询操作，对基于DFA的匹配引擎进行分析。指出单一的存储器件无法同时满足较大规模DFA的容量需求和吞吐量需求，进而提出基于两级存储的匹配引擎。通过一系列实验对匹配过程中状态访问的分布进行分析，实验发现状态访问呈Zipf分布，并且与自动机和报文无关。在此实验基础上，结合具体存储器件的参数，对两级存储的匹配引擎进行定量的理论分析。在高级存储器命中率为90%时，可获得50Gbps以上的吞吐量。

第四章 报文匹配的马氏模型

第三章提出了基于两级存储的匹配思想，并指出在高级存储器命中率较高时，可以获得很高的吞吐量。关键是如何用理论的方法来确定那些访问概率较高的状态，因为我们不可能每次都通过一系列的实验来进行统计分析。本章通过随机过程中的马尔可夫链理论对报文匹配中的状态迁移过程进行建模，使用稳态向量作为对状态访问概率的预测。并且重点探讨了稳态向量的求解过程，最后通过将理论状态访问概率与实际匹配中的概率比较，说明马尔可夫链的预测具有很高的准确性，可以为两级存储的匹配思想提供理论支撑。

4.1 马尔可夫链模型

4.1.1 基本概念

马尔可夫过程是一类特殊的随机过程，最初由[俄罗斯](http://zh.wikipedia.org/wiki/%E4%BF%84%E7%BD%97%E6%96%AF)[数学家](http://zh.wikipedia.org/wiki/%E6%95%B0%E5%AD%A6%E5%AE%B6)[安德烈·马尔可夫](http://zh.wikipedia.org/wiki/%E9%A9%AC%E5%B0%94%E5%8F%AF%E5%A4%AB)所研究。在该过程中，随机变量在时刻*n*的值只依赖于它上一时刻*n-1*的值，其中随机变量的值即系统所处的状态。如果马尔可夫过程的状态空间是离散的，这类马尔可夫过程称作马尔可夫链[45]。在马尔可夫链中，状态以ID的形式标记为0,1,2……这里我们主要研究离散时间的马尔可夫链。

在离散时间马尔可夫链中，随机变量*S(n)*代表系统在时刻*n*所处的状态。*S(n)*只与*S(n-1)*有关，而与*S(0)*,*S(1)*……*S(n-2)*都无关，即将来的状态只与当前的状态有关，而与过去的状态（即当前以前的历史状态）无关。这是马尔可夫链最本质的特征，这种无关性可以用条件概率分布描述如式(4-1)所示。

 (4-1)

在基于DFA的报文检测中，假设报文内容是随机的，那么报文匹配过程中的状态迁移就是一个随机过程。而下一步要转移的状态只与当前状态和输入字符有关，与之前任意时刻所处的状态都无关，具有典型的马尔可夫性。因此我们可以用马尔可夫链的理论来研究状态转移规律。

我们将时刻*k*时，状态*i*下一步转移到状态*j*的概率记为，即。如果这个转移概率与时刻*k*无关，即任意时刻状态*i*下一步转移到状态*j*的概率都相等，记为，这样的马尔可夫链叫齐次马尔可夫链，齐次马尔可夫链的状态转移特征如式(4-2)所示，其中*k*为任意正整数。在报文匹配中，由于自动机是固定的，任意时刻状态间转移概率不变，因此基于DFA的报文匹配具有齐次马尔可夫链的性质。

 (4-2)

4.1.2 转移转移概率矩阵

对于齐次马尔可夫链，我们可以用一步转移概率矩阵***P***来描述其一步转移概率特征，如式(4-3)所示。其中*m*代表系统中状态总数，矩阵中第*i*行第*j*列的元素代表状态*i*一步转移到状态*j*的概率，齐次马尔可夫链的状态转移概率矩阵与时刻无关。

 (4-3)

显然***P***的每个元素均为非负，并且每行之和均为1。对于给定的DFA，通常可以表示成一个五元组的形式(,*Σ*,δ,,*F*)，其中表示DFA的状态集，*Σ*为输入字符表（包含256个字符），δ为转移函数，为初始状态，*F*为接受状态集。转移概率矩阵***P***是一个的方阵，***P***中任意一个元素代表状态*i*随机转移到状态*j*的概率。

在DFA中，假设匹配引擎当前所处的状态为，如何求解它一步转移到状态的概率？如果不存在字符*a*满足，那么显然从转移到的概率为0。如果存在字符*a*满足，由于DFA中转移函数函数值唯一，并且报文内容是随机的，即字符表中每个字符出现的概率是相等的，那么从转移到的概率与字符表中满足的字符*a*的个数成正比。例如，当前状态接收一个输入字符只能转移到状态或，满足的字符集为*B*，满足的字符集为C，则从转移到和的概率分别为和，转移到其它状态的概率均为0。

进而可以通过状态表计算一步转移概率矩阵P，其中第i行第j列的元素：。由于假定报文内容是随机的，P 的每个元素的值介于0和1之间，并且每行元素之和为1。以规则“.\*ab+ac”为例，对应的DFA如图4.1所示。



图4.1 规则“ab+ac”的DFA

通过上述方法可求得转移概率矩阵如式(4-4)所示。

 (4-4)

从式(4-4)中可以得出每个状态转移到其它状态的概率，例如状态0接收字符*a*转移到状态1，接收其它255个字符转移到自身，所以，。

4.1.3 状态分布向量

为第3介绍稳态分布做准备，本节先引入状态分布向量的概念。我们将系统在时刻*k*时处于状态*j*的概率记为，即，由式4-2可得到式(4-5)。其中，系统中状态总数为*m*，*i*和*j*都是[1,*m*]之间的正整数。进一步，系统在时刻*k*时，处于各状态的概率组成的向量如式(4-6)所示，称为分布向量，它描述了系统在各状态的分布概率。

 (4-5)

 (4-6)

由于是系统在时刻*k*时分布在各状态的概率，显然可得式(4-7)，其中*k*为任意自然数。

 (4-7)

由式(4-3)、(4-5)、(4-6)进一步可得式(4-8)。

 (4-8)

由式(4-8)可得：





(4-9)

以图4.1所示的DFA为例，初始向量分布，经过三步转移之后的向量分布为。在给定初始向量分布和转移概率矩阵的情况下，可以通过式(4-9)计算任意时刻的向量分布。其中表示*k*步转移概率矩阵，中第*i*行第*j*列的元素表示状态*i*正好经过*k*步转移到达状态*j*的概率。要求，关键是要计算。但是通常情况下求的计算量很大，一般不采用直接计算的方式。在的规模较大时，一般通过扩展、特征值表示法、Jordan标准形、Z变换等方法[46]来求解。

4.1.4 周期性

周期性是马尔可夫链的重要性质，定义周期性的状态如下：

**定义1**：如果有正整数*d*，*d* >1，只有当*n*=*d*，2*d*，3*d*，…时，，或者说当*n*不能被*d*整除时，，则称状态*i*是具有周期性的状态。其中，代表状态*i*经过*n*步转移到达状态*j*的概率。注意状态*i*和状态*j*也可能不是指具体某一个状态，而是分别代表一类状态中的某一个，下面举例说明。

设有四个状态（0，1，2，3）的马尔可夫链，它的一步转移概率矩阵如式(4-10)所示，易知其状态转换图如图4.2所示。

 (4-10)



图4.2 式4-10的状态转换图

图中的数据表示状态转移概率，四个状态可以分为{0,1}和{2,3}两个子类，显然从0或1出发，经过任意2*k*次转换，都会回到0或1的某一个状态。该过程有确定性的周期转移：{0,1}→{2,3}→{0,1}→{2,3}…，它的周期为2。在正则表达式的DFA转换图中，基本上不会出现周期性的转移，因此在后面的讨论中我们不再考虑周期性转移的问题。

4.2 可约马尔可夫链

4.1节介绍了马尔可夫链的基本概念，本节讨论马尔可夫链另一重要特征可约性，可约性是后面求解大规模马尔可夫链稳态分布的关键问题。

可约马尔可夫链描述了这样一类系统，一旦系统访问了某些特殊的状态，就不能再访问其它状态。下面以两个示例来区分可约性和不可约性，以图4.3(a)为例，从任意一个状态出发，我们都可以经过一步或有限若干步到达任意状态，这样的马尔可夫链称为不可约马尔可夫链。以图4.3(b)为例，从某些状态出发，经过任意步转换可能都无法到达其它状态，例如从状态0出发，只能到达状态0，这样的马尔可夫链称为可约马尔可夫链。

  (a) (b)

图4.3 可约马尔可夫链和不可约马尔可夫链

传统上可约性可如下定义：一个马尔可不夫链不可约当且仅当存在一个正整数，使得所有的元素都非0。根据马尔可夫链的定义，从某些状态出发，无法转移到其他一些状态。据此可将状态空间划分为两个集合，闭态集和过渡状态集，划分之后状态转换关系如图4.4(a)所示。当系统处于中时，它即可以转移到中，也可以转移到中，但系统一旦处于中后，它就无法再转移到中。闭集合中任意两个状态是相互可达的，通常情况下，一个可约马尔可夫链可能含有多个闭态集，图4.4(b)为含有两个闭态集的状态空间，从中的状态出发可以转移到或或中的状态，但从出发不能转移到外任意一个状态，从出发也不能转移到之外任意一个状态。

(a) (b)

图4.4 可约马尔可夫链状态空间划分

以图4.5(a)所示的DFA为例，其状态转移概率矩阵如图4.5(b)。图中转移边上的数字代表转移概率，从2状态出发可以到达0、1、3三个状态，但是从0、1、3三个状态出发都不能到达2状态，所以0、1两个状态和2状态是不相通的。这个马尔可夫链有两个闭集{0,1}和{3}。



  
(a) (b)

图4.5 含两个闭态集的马尔可夫链

对于大规模的马尔可夫链，通过人工的方法完成状态分类是不可行的。Xie在1998年提出的有限状态马尔可夫链状态空间划分算法[47]，可高效地完成状态集的空间划分。为了进一步计算的方便，通常会根据状态空间的划分对转移概率矩阵的形式进行调整，调整后的形式如图4.6(a)所示。其中为子随机方阵，所谓子随机方阵即每行元素之和小于或等于1（但至少有一行元素之和小于1）的非负方阵，这是因为代表的是过渡状态集内部的转移关系，过渡状态除了会转移到自身外，还会转移到闭集合的状态，因此每行元素之和小于等于1。其中每一个为行随机方阵，行随机方阵指每行元素之和为1的非负方阵。因为每一个代表一个闭态集内部的转移关系，而闭态集中没有转移到闭态集之外的转移边，实际上每一个闭态集都可视为一个不可约的马尔可夫链。每一个为非负矩阵，代表过渡状态集到闭集的转换关系。具体的转换方法为，依次扫描各集合中的状态，并进行行列转换，对状态数为的马尔可夫链，需要做次行列转换。按此方法对图4.5(b)中的转移概率矩阵转换后得到的矩阵如图4.6(b)所示。

1. (b)

图4.6 调整后的转移概率矩阵

4.3 稳态分布

4.1节和4.2节分别介绍了马尔可夫链的基本概念和马尔可夫链的可约性，本节主要研究马尔可夫链的极限特征，即当时，状态分布向量的取值，即稳态分布。在计算机与通信理论中，稳态分布可以用来计算一些性能参数，如吞吐量、时延、丢包率等。事实上，在转移次数较大时，系统就会进入稳态。因此在报文匹配的应用中，可以用稳态分布作为对状态访问概率的预测。

4.3.1 稳态分布的基本概念

对于齐次非周期性马尔可夫链，当时分布向量稳定到一个定值[48]，并且满足式(4-11)。即在稳态条件下，分布向量不再随状态转移而变化。显然，是的特征值为1时的特征向量。当满足式(4-11)时，称马尔可夫链到达稳态，称为稳态分布或稳态向量。

 (4-11)

式(4-11)表明，若是当前分布向量的值，则经过一步状态转移后，分布向量的值仍为。稳态分布并不是指在稳态时以概率1处于某个固定的状态，而是指各状态分布概率不再随状态转移过程而变化。它包含两层意思，一是分布向量达到稳态，二是*n*步转移概率矩阵达到稳态。例如一个含有5个状态的马尔可夫链，其稳态向量，它代表当该马尔可夫链趋于稳定时，系统处于状态1的概率是20%，位于状态2的概率是10%……我们可以认为在稳态时，该系统最可能处于状态3。

稳态分布只与转移概率矩阵相关，而与初始分布向量无关，因此可以判定当达到稳态时应该满足一定的结构特征。事实上，对不可约马尔可夫链而言，在达到稳态时满足以下定理及推论。对于可约马尔可夫链的稳态问题放在4.3.3节讨论。

**定理1**：对不可约马尔可夫链，存在一个正整数，使得对状态空间中任何状态*i*，*j*均有，令，是一行随机方阵，且它同列的每个元素均相等，一般将称作稳态矩阵。

**推论1:** 的行矢量满足且，且该矩阵是唯一能满足上述关系的矩阵。

**推论2：**系统稳定后的分布向量与初始向量分布无关，即，其中*i*为任意状态。

例如，一个不可约马尔可夫链，其转移概率矩阵如图4.7(a)所示，其稳态矩阵如图4.7(b)所示，显然其稳态向量为。

1. (b)

图4.7 不可约马尔可夫链的转移概率矩阵及稳态矩阵

通过以上分析可知，求解稳态向量的等价于求解稳态矩阵，下面介绍一些常用的求稳态矩阵的方法。

4.3.2 常用的求稳态矩阵方法

本节介绍一些常用的求解稳态矩阵的方法，并分析各种方法的优缺点。常用求解稳态矩阵的方法有：转移矩阵重复相乘；求的特征值的特征向量；差分方程；Z变换；直接数值计算法解线性方程组；迭代法解线性方程组[46]。

矩阵重复相乘法容易产生舍入误差，影响结果的正确性。另外，这种方法比较机械，需要一直重复相乘，直到矩阵的数值不再变化，因此它的效率也会非常低。特征向量法就是求解的特征值对应的特征向量，它对马尔可夫链的规模要求比较苛刻，只有在马尔可夫链规模特别小的时候，数学工具包才能较容易地计算出特征向量。而在报文匹配中，一个小规模的自动机也有成千上万个状态，因此特征向量法不适合用在我们的模型中。差分方程法只有在矩阵是带状矩阵时比较有效，所谓带状矩阵是指在矩阵中，所有的非零元素都集中在以主对角线为中心的带状区域中，其中最常见的是三对角带状矩阵。在报文匹配的应用中，转移概率矩阵没有明显的结构特征，即使经过行列变换，依然无法呈带状矩阵的形式，因此差分方程法也不适用于我们的模型。Z变换方法主要使用在矩阵是下三角矩阵或下Hessenberg矩阵[49]的场合，下三角矩阵是指[矩阵](http://science.scileaf.com/wiki/440)的对角线以上的元素都为0的矩阵，下Hessenberg矩阵指主对角线上面某条次对角线之上所有元素为0的矩阵，实质上是一种扩展的下三角矩阵。报文匹配中的转移概率矩阵也不具备这样的特征。

最后两种方法是通过求解线性方程组来计算稳态矩阵，即需要求解如式(4-12)所示的方程组，其中为稳态向量。

 (4-12)

数值计算解线性方程组是一种通用的方法，这种方法对的结构没有要求，但是要求的规模不能过大。因为的规模较大时，舍入和截断误差带来的影响非常显著，另外结果的准确性还依赖于计算机的精确度及计算规模。显然，解线性方程组的方法也不适用。迭代法求解线性方程组也是一种通用的方法，常用的迭代法有雅可比迭代、高斯-赛德尔迭代、超松弛迭代[50]等。这类方法对的结构没有要求，另外舍入和截断误差对结果影响很小，且这种方法的结果准确性只依赖于计算机的精确度，而与计算规模无关。从这些优点看来，迭代法求解线性方程组是较为理想的计算方法。但用迭代法求解方程组的时间复杂度为，其中为状态数。在规则集较为复杂的时候，P的阶数可能达到数百万，求解时间非常长。另外即使能够求出一些稳态向量的解，还需要额外的计算来确定哪个解是状态0所在行对应的稳态向量。

4.3.3 可约马氏链的稳态向量求解

通过4.3.2的分析可以知道，虽然稳态矩阵或稳态向量有多种求解方法，但这些方法或者对计算比较敏感、或者是对矩阵结构有特殊要求，都不适用于报文匹配中稳态矩阵的求解。因此，需要探索新的计算方法来计算稳态矩阵。

在深度报文检测系统中，规则集都比较复杂，一般得到的自动机都是可约的，即从某些状态出发无法到达所有的状态。尤其是当规则中带有首锚时，例如，规则“^a.\*d”和规则“.\*b?c”编译得到的DFA如图4.8所示。很明显，这是一个可约马尔可夫链，从状态0之外任何一个状态都不能到达状态0。其中，{0}是过渡状态集，{1,3}和{2,4,5}是闭集。从匹配的角度来看，不带首锚的规则通常在闭集中匹配成功，带首锚的规则可能在过渡状态集或闭集中匹配成功。



图4.8 规则“^a.\*d”和“b?c”的DFA

由于报文匹配的应用中自动机都是可约的，下面主要讨论可约马尔可夫链的稳态矩阵及稳态向量的问题。在可约马尔可夫链中同样存在稳态矩阵，但是它的稳态矩阵不满足同列元素都相等的特征，下面我们从可约马氏链的转移概率矩阵的结构进行说明。在可约马氏链中，可以按状态之间的可达性将状态空间分为过渡状态集和闭集，闭集通常可能有多个。再经过行列变换之后，可以得到如图4.6(a)所示结构的矩阵。例如在图4.8中的DFA，经过行列变换后得到的转移概率矩阵如式(4-13)所示。

 (4-13)

由图4.6(a)的矩阵出发，可进一步得如式(4-14)所示。

 (4-14)

其中

 (4-15)

当时，由于是子随机方阵，有[46]，且

 (4-16)

于是，

 (4-17)

进而，

 (4-18)

我们将这个稳态矩阵记作，显然该矩阵也满足方程，但是和可约马尔可夫链的稳态矩阵有很大区别，它没有同列各元素值相等的特征。但是对而言，因为每个相当于一个不可约马尔可夫链，所以同列各元素值都相等。虽然每行并不相等，不能用一个稳态向量来对其描述。但是在报文匹配的应用中，自动机的状态总是从状态0开始的，因此我们只需要得到中对应状态0的那一行向量。

从式(4-18)可知，求稳态矩阵需要分别求解和，又由式(4-17)可知，的求解依赖于和。对于大规模稀疏矩阵的求逆，现在已经有较为成熟的方法，一般可通过LU预处理+GMERS、迭代法等求解[51]。所以关键是对的计算，如果每个闭集的规模都比较小，可以很快地通过矩阵的幂运算得到。再利用式(4-17)计算，进而得到稳态矩阵，第一行元素即为从状态0出发的稳态向量。

实践证明，通常得到的闭集规模都比较小，只有数百阶，因此适合用第二类方法求解稳态向量。另外大规模矩阵相乘可能会带来精度的影响，但这并不影响最终的结果，因为我们需要的是稳态时每个状态概率的相对大小而不是绝对大小。

以式(4-13)矩阵为例，通过以上方法可求得稳态矩阵如式(4-19)：

 (4-19)

我们取所在行对应的向量作为稳态向量，可以得到理论上各状态访问概率，闭集合中状态访问概率排序为。

4.4 模型评估

4.3节对稳态分布的相关问题进行了详尽的阐述，并且给出了在报文匹配的应用中，求解不可约马尔可夫链的稳态分布的合理方法。本节通过实际报文匹配进行仿真实验，对稳态向量预测的准确性进行评估。

规则集选取L7-filter protocols，为保证生成完整的DFA，随机选取40条规则，编译生成的NFA状态数为1142，子集合算法生成的DFA状态数为210962。测试数据集有两组，第一组是1999年DARPA的第一周周一的外网数据包（323M，共1362869个报文），由于DAPRA数据量小，报文较短，且缺少很多新的报文协议，可能对性能评价有一定的影响。第二组数据是我们于2011年11月15日21:00，在华南某OC192链路上被动获取的报文CAPTURE（2.3G，共5293074个报文）。

首先讨论理论上状态访问概率问题，根据编译得到的DFA，可按照4.3节的方法，计算转移概率矩阵，再对该矩阵划分状态空间，共得到109个闭态集，每个闭态集平均状态数71，过渡状态集共有状态数203223。再求解稳态矩阵，最后将状态0对应行的向量作为稳态向量，稳态向量中每个元素的值作为对相应状态的访问概率的预测。通过马尔可夫链计算的各状态理论访问概率如图4.9所示，闭集合中DFA状态总数为7739，为了方便表示，图中只取了访问概率最高的600个状态。横坐标表示状态ID，因为高访问概率的状态在状态空间中可能比较分散，为了表示方便，对横坐标按状态访问概率从大到小进行排序。纵坐标表示状态访问概率，考虑到不同状态间概率差异太大，为表示方便，对纵坐标取概率的常用对数。



图4.9 部分DFA状态理论访问概率

由图4.9可以看出，理论上DFA的状态访问概率分布极其集中，极个别状态访问概率很大，而绝大多数状态访问概率很小。如横坐标0对应的状态理论访问概率为97%，而横坐标600对应的状态理论访问概率只有6E-15。说明理论上预测到的状态访问分布也服从Zipf分布，另外不难发现，图4.9的曲线与第三章实测的状态访问概率曲线走势基本一致。虽然它们对应不同的自动机，但也能说明理论预测分布模型和实际模型比较稳和。

下面对理论预测的访问概率分布和实际匹配中状态访问概率分布进行对比。对DARPA报文和CAPTURE报文分别进行仿真匹配实验，统计得到DFA状态访问概率函数，如图4.10所示。其中粗线表示理论计算的状态访问概率分布，细线表示匹配CAPTURE报文时统计的状态访问概率分布，虚线表示匹配DARPA报文时状态访问概率分布。由图4.10可以看出，实际DFA状态访问概率函数与理论访问概率函数曲线走势相同，并且CAPTURE报文的概率函数曲线更接近于理论值。这主要是因为DARPA报文数据量小，报文较短，稳态特性体现的没有匹配CAPTURE报文时明显。虽然实际值与理论值之间存在一定的偏差，但偏差比较小，说明通过马尔可夫链理论计算出的访问概率准确性很高。



图4.10 匹配DARPA时DFA状态实际访问概率与理论访问概率差值

通过图4.9和4.10可以看出，理论上和实际上对DFA状态的访问都很集中，极少数状态有很高的访问概率。由此统计出实验中DFA命中率随一级存储器空间大小的变化，如图4.11所示。



图4.11 DFA命中率随一级存储器空间大小变化

由图4.11可以看出，在一级存储器容量比较小时，命中率随容量的增加而急剧提高；当容量达到15KB时，随着容量的增加，命中率几乎保持不变。说明一级存储器只需要很小的容量就能使DFA的访问达到很高的命中率。一般FPGA片上存储器单块RAM容量可达数百KB，事实上只需要分配数十KB存储这些访问概率较高的状态表项即可，剩余的空间可以用于报文FIFO、流管理等。

4.5 小结

本章是理论建模部分，通过马尔可夫链对报文匹配中状态迁移过程进行建模。详细介绍了马尔可夫链的基本概念，并讨论了周期性、可约性等特征。在报文匹配的应用中，常见的模型为非周期性不可约马尔可夫链。稳态分布是非周期性马尔可夫链在转移次数趋于无穷大时，系统在状态空间中各状态的分布概率，可以作为对状态访问概率的预测。对于大规模马尔链的稳态分布求解，常用的方法或者对转移概率矩阵结构有特殊要求，或者效率太低，无法适应在报文匹配中的应用。比较实际的方法是划分状态空间，通过求解各闭集合的稳态分布来计算整个马氏链的稳态分布。最后，通过仿真实验对马尔可夫链模型进行评估，实验证明在报文匹配的应用中，马尔可夫链模型的预测准确性很高。

第五章 基于NetMagic的系统实现

第四章用马尔可夫链模型对报文匹配过程建模，并且说明了稳态向量对状态访问概率预测的准确性。因此，在实际应用中，我们可以用该模型计算出来的稳态向量来指导状态表项在两级存储器中的配置。

为了压缩研发时间和成本，匹配引擎的实现基于国防科技大学网络与信息安全研究所研制的开放式网络交换平台NetMagic。采用该平台的另一个重要的原因是基于存储器件的选择，NetMagic上FPGA的片内存储资源可以满足我们对性能的需求，另外，NetMagic上配置的512Mb的DDR2 SDRAM则可以满足大容量的状态表存储需求。本章首先介绍NetMagic开发平台，然后分别从硬件平台和软件平台说明匹配引擎的实现过程，最后是实验环境的搭建及实验结果分析。

5.1 NetMagic开发平台

本节从NetMagic体系结构及软硬件处理流程、UM总体结构、报文处理流程、报文格式及地址空间分配四个方面详细介绍开发平台。

5.1.1 NetMagic体系结构

NetMagic的设计主要是为研究人员和学生在网络创新实验中快速构建高性能网络系统，并对建模和算法等进行验证。NetMagic采用新型部分可编程交换机体系结构技术设计实现，将通用的高密度FPGA与商业的以太网交换芯片结合使用，可提供高端口密度下报文线速交换转发能力，有效支持用户自定义报文处理逻辑[52]。另外它是一个与控制设备低耦合的独立网络设备，不需要嵌入到特殊的主机设备中，因此简化了网络实验系统的构造[53]。此外，通用的管理控制接口则使NetMagic避免平台相关性可能导致的兼容性和可移植性问题。NetMagic平台的基本组成如图5.1所示，由NetMagic基本硬件平台和可运行在远程主机上的管理平台组成。

本系统采用NetMagic08型号，其硬件结构如图5.2所示。NetMagic08以一片中等规模的FPGA为核心，外带4个RJ45电口和4个SERDES光口，平台对外提供一个百兆以太网控制端口。NetMagic的硬件平台提供高速报文交换能力，平台内部集成的FPGA可有效提供可重构能力。平台内部预先集成管理模块，支持远程控制端基于NMAC管理访问协议，通过以太网端口，以流表管理等方式实现对硬件平台的管理。为了对报文进行重组，NetMagic08还设计了一片容量为512Mb的DDR2 SDRAM进行报文缓冲。报文从某个端口进入，根据转发规则决定从某个端口输出或送至DDR2缓存。



图5.1 NetMagic平台基本组成



图5.2 NetMagic08硬件结构图

NetMagic软件系统包括Libpcap/Libnet开发包、NMAC协议及用户控制程序。NMAC协议是控制端与NetMagic硬件平台的通信协议，提供双方通信协议并定义相应规范，Libpcap/Libnet为NMAC的运行提供底层接口函数支持，控制程序由用户根据不同的需求进行编写。NetMagic的软件系统基于远程主机上的基本网络协议栈，将基于NMAC协议的管理报文封闭在IP报文中，通过互联网络发送到硬件平台，实现远程访问、管理及控制。FPGA内部的管理模块负责基于NMAC协议与外部控制端建立连接，并对控制端的命令进行解析，转换成总线命令，完成对NetMagic内部功能模块的控制及存储器的管理工作。

5.1.2 UM总体结构

用户模块(User Module, UM)是用户可根据需求自定义开发的模块，其它模块的功能及代码已固定，一般不需修改。图5.3是FPGA内部逻辑结构图。UM在内部模块中处于核心地位，通过相关接口分别与管理模块、通用数据路径(Common Data Path, CDP)及DDR2控制器相连。UM接收CDP输入模块送入的报文，同时可以根据需要决定报文是否暂存到外部存储器中，以便根据需要对报文进行修改等操作，最后根据一定的转发规则把报文输出给CDP输出模块，实现报文的转发输出。为实现对UM中的报文及状态的实时管理控制，UM也可以根据管理模块发送的管理命令，对UM中的寄存器或RAM表进行读写。

图5.3 FPGA内部逻辑结构图

输入控制模块从网口接收以太网格式的报文数据，进而将报文转换成139位的数据格式，写入到输入控制模块内部的二级FIFO。此时如果UM向输入控制模块发送数据请求命令，则输入控制模块从其二级FIFO中读取数据，同时上传给UM及输入控制模块和输出控制模块之间的FIFO。

UM获取报文后，可对报文进行修改操作，然后生成处理规则发送给输出控制模块。输出控制模块根据UM发送的处理规则，对接收到的报文进行截断、复制、丢弃等操作，然后再将处理后的139位的报文格式转换成以太网报文，从指定的端口转发出去。

当用户需要通过UM对UM内部寄存器（FPGA的RAM块）或外部存储单元进行读写操作时，则可以通过百兆管理接口发送基于NMAC协议的以太网管理报文。管理模块将收到的NMAC协议报文解析后，转换成命令及数据发送给UM。UM根据读写命令及相应数据进行读写操作。

UM与DDR2的控制器之间的接口完成对DDR2 SDRAM中数据的配置及访问。另外在CDP输入输出控制模块之间还有一个缓冲FIFO，用于将报文直接旁路到输出模块，而不需要通过UM的转发。

UM与各接口之间的信号及时序关系比较复杂，具体可以参见工作手册[52]，这里不再详细介绍。

5.1.3 UM报文处理流程

UM报文处理流程实际上就是UM与CDP之间的交互过程，UM直接与CDP的输入控制模块及输出控制模块相连。UM与CDP之间详细的接口线路图如图5.4所示。CDP的输入控制模块将接收到的报文同时往两个方向复制：UM和CDP输入输出FIFO。UM一边接收报文，一边提取报文的关键字，生成规则信息。规则信息在报文尾到达前生成完毕，然后等待输出控制模块的许可，即CDP的规则FIFO有空闲空间时（um2cdp\_rule\_usedw<5’d30），再将规则信息送至CDP输出控制模块。



图5.4 UM与CDP之间接口线路图

CDP的输出控制模块解析UM发送的规则信息（um2cdp\_rule，共30位），根据规则信息的最高位决定从哪条路径读取报文，若最高位是0，表示从UM读取报文；若最高位是1，表示从CDP输入输出缓冲FIFO读取报文。由于规则信息是在报文尾到达UM时才送至CDP输出模块的，因此在CDP输出控制模块解析完规则信息准备转发报文时，必有一个完整的报文在UM或CDP输入输出FIFO中。

UM共有三种处理模式：旁路处理、穿透处理、混杂处理。旁路处理中UM不修改报文内容，输出模块从FIFO中读取报文；穿透处理中UM可修改报文，输出模块从UM读取报文；混杂模式中，输出模块可根据规则信息选择从FIFO或UM中读取报文。不论采用哪种处理模式，输出控制模块都需要UM发送的规则信息来决定报文的转发操作。

5.1.4 报文格式及地址空间分配

CDP及UM获得的报文宽度均为139位，其中高11位为控制信息，低128位为报文数据，UM及CDP根据控制信息来确定报文的处理方式。端口收到的报文都是以太网报文，以太网报文最小长度64个字节，即每个报文至少有四拍数据。具体报文格式如图5.5所示。



图5.5 内部报文格式

其中[138:136]位表示头尾标识，101代表报文头部，100代表报文中间，110代表报文尾部。[135:132]代表该拍报文中有效字节数，0000表示最高一个字节有效，0001表示最高两个字节有效，依次类推，1111表示16个字节都有效。[131:128]表示该报文的输入端口号，只在每个报文的第一片报文中标识。

NetMagic内部地址总线均为32位，系统地址空间主要由UM、CDP、DDR2几部分组成，地址空间分配表如表5.1所示。

其中，CDP内部的寄存器用于统计各端口的报文接收、转发、丢弃及出错情况。UM地址空间可进一步分为UM内部寄存器和UM内嵌RAM表，如表5.2所示。

表5.1 NetMagic地址空间分配

|  |  |
| --- | --- |
| 系统地址空间  31 28 | 再分配地址空间  27 0 |
| 0000：CDP | CDP内部寄存器地址 |
| 0001：UM | UM地址空间 |
| 0100 | DDR2地址空间 |
| 其余 | 保留 |

表5.2 UM内部地址空间分配

|  |  |
| --- | --- |
| 27 26 | 25 0 |
| 00：内部寄存器 | 内部寄存器地址 |
| 01：UM内嵌RAM转发表 | UM内部RAM表 |
| 10-11 | 保留 |

UM内部寄存器均为32位宽，具体定义参见开发手册。另外，用户可根据实际需要来确定RAM地址需要多少位。

5.2 系统设计实现

系统实现部分包括编译器、控制端程序、NetMagic硬件平台程序。下面分别介绍各部分的功能及实现。

5.2.1 编译器

编译器功能主要包含四步：将正则表达式集合编译成NFA，NFA转换成DFA，根据DFA求解稳态向量，状态表重命名。

在编译器的实现中，正则表达式构造NFA采用Thompson构造算法，NFA生成DFA采用子集合构造算法，生成DFA后还需要最小化的化简操作，稳态向量的计算方法参照4.3.3节。状态表重命名是按稳态向量中各状态理论访问概率大小，将状态标识号重新分配的过程。重命名后状态理论访问概率随标识号增加而减小，即标识号越小，理论访问概率越大。实际上只需要调整闭态集合中状态的标识号，因为过渡状态集中的状态理论访问概率都为0。这样做的目的是为了提高访存的效率，因为高访问概率的状态标识号通常都不连续，如果直接将这些状态表项存储在高级存储器中，匹配过程中要逐一比对这些表项，增加了寻址复杂度，严重降低了寻址效率。但是如果给这些高概率的状态重命名后，只需要根据标识号就可以方便地计算出状态表项的地址。重命名算法很简单，其伪代码如图5.6所示。

其中，NextBiggest是根据稳态向量，找出当前集合C中访问概率最大的状态ID。ExchangeID函数将DFA状态表项中两个状态的ID相互交互，交互方法很简单，每交换一对状态的标识，只需要完整扫描一次状态表，将状态表中这两个状态的标识互换。在重命名后得到的DFA中，状态访问概率随标识号增加而减小。如此就可以根据高级存储器容量大小，将从0开始的标识号连续的部分高访问概率的状态表项存储在一级存储器中。

|  |
| --- |
| 输入：稳态向量；原始DFA；所有闭合状态集合C  输出：重命名后的DFA |
| **Procedure Rename**   1. for(i=0; i<|C|; i++) 2. begin 3. j = NextBiggest(S,C); 4. Delete(C,j); 5. ExchangeID(DFA,i,j); 6. end |

图5.6 DFA重命名算法

5.2.2 控制端程序

控制端程序主要提供基于NMAC协议对NetMagic内部寄存器、RAM表及DDR2 SDRAM的配置、查询及修改等操作功能。在报文检测系统中其主要的功能是将状态表项配置在FPGA内嵌RAM块及DDR2 SDRAM中，并提供匹配结果查询功能，控制端程序界面如图5.7所示。

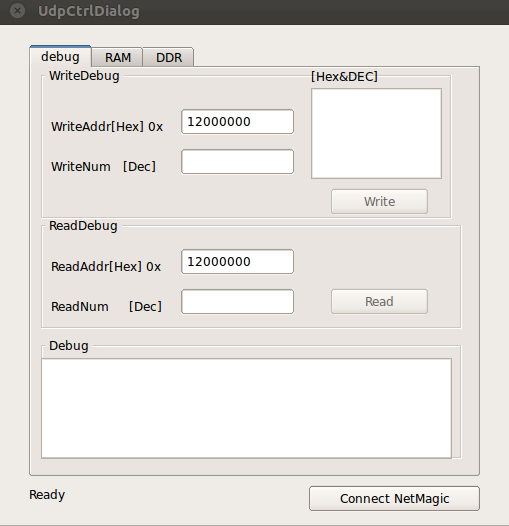


图5.7 控制端程序界面

在FPGA内嵌RAM块的配置中，根据第四章的分析，我们选择在每个RAM块中配置20个状态表项，每个状态的转移表项需要256\*2B=512B的空间，故每个RAM块需要分配10KB的空间，位宽为18。NetMagic只有四个电口和四个光口，只能做四输入四输出的转发操作，因此选择实例化四个匹配引擎，需要四个RAM块，RAM块资源消耗40KB。RAM块起始地址为32’h14000000，每个RAM块占用10240个地址，第一个RAM块地址为32’h14000000-32’h140027ff，依次类推。

控制端程序从状态0的第一个转移表项开始，依次读取DFA状态转移表前20个状态的各转移表项，并写入各RAM块的地址空间，各RAM块存储的内容都是前20个状态的转移表项。

DDR2 SDRAM中配置完整的DFA状态转移表，由于DDR2的容量相对较大，对RAM块中状态表项的重复存储带来的开销可以忽略，这可以进一步降低寻址的复杂度。DDR2中状态表的配置需要通过与DDR2 SDRAM控制器交互来实现，配置过程较为复杂。DDR2 SDRAM的地址空间为32’h40000000-32’h43ffffff。 另外，DDR2控制器数据通路宽度为128位，为了提高配置效率，每次将一个状态的四个转移表项合并成128位写入到DDR2 SDRAM中。

另外，还需要将匹配规则与终止状态的对应关系配置到存储器中。为了提高处理速度，可利用FPGA内部的剩余空间来配置这些表项。控制端程序还提供匹配结果查询功能，FPGA片内实例化一个256\*18bit的RAM块RESULT\_RAM，用于存储各条规则匹配命中的次数，其地址空间为32’h14003000-32’h140001ff。

5.2.3 NetMagic硬件程序

NetMagic硬件程序主要指UM模块的代码，UM主要功能是报文匹配处理并生成转发规则。UM内部逻辑结构图如图5.8所示。图中主要包括表项配置模块、报文采集分发模块、匹配处理模块、规则采集处理模块，虚线框内四个模块代表四个匹配引擎。由于UM并未对报文内容修改，所以将输出控制模块报文来源都设置为输入控制模块，图中省略了UM到输出控制模块的数据通路。

表项配置模块根据管理模块的命令，将DFA状态表项按地址配置到DDR2 SDRAM中。报文采集分发模块读取输入控制模块的报文流，存放至其内部的报文缓冲FIFO\_PACK。然后按轮转的方法，依次将每个完整的报文分发到各匹配引擎的报文缓冲区，一个完整报文的各节拍报文必须按序送到同一个匹配引擎。

匹配引擎i内部包含一个报文缓冲区FIFO\_Pi和一个规则缓冲区FIFO\_Ri，匹配引擎i从报文缓冲区FIFO\_Pi读取报文，若命中某条规则，则将Result\_RAM中该规则的命中次数加1。匹配引擎i每处理完一个报文，将该报文的处理规则写入内部规则缓冲FIFO\_Ri。各匹配引擎在匹配处理过程中分别访问各自的一级存储器RAM表，但是共用二级存储器DDR2 SDRAM。



图5.8 UM内部逻辑结构图

规则采集处理模块采用与报文采集分发模块同样的轮转方式，依次从各匹配引擎的规则缓冲中读取规则，并写入到UM与输出控制模块之间的规则缓冲FIFO\_RULE。

下面分别介绍各模块的详细实现。

5.2.4 表项配置模块

表项配置模块根据管理模块的命令对DDR2 SDRAM进行写入或读取，其模块结构如图5.9所示。



图5.9 表项配置模块结构图

表项配置模块的信号列表如表5.3所示。

表5.3 表项配置模块信号列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名称 | 方向 | 位宽 | 描述 | 备注 |
| ddr2um\_ready | input | 1 | 控制器ready信号，高有效 |  |
| ddr2um\_rdata | input | 128 | 读ddr返回的数据 |  |
| ddr2um\_valid\_rdata | input | 7 | 读ddr返回的数据信息 |  |
| ddr2um\_valid\_empty | input | 1 | 标识信息FIFO是否为空 |  |
| um2ddr\_wrclk | output | 1 | 写时钟 |  |
| um2ddr\_wrreq | output | 1 | 写出口数据FIFO的写信号 |  |
| um2ddr\_wdata | output | 128 | 写出口数据FIFO的数据 |  |
| um2ddr\_command\_wrreq | output | 1 | 命令FIFO的写信号 |  |
| um2ddr\_command | output | 34 | 命令 |  |
| um2ddr\_rdclk | output | 1 | 读时钟 |  |
| um2ddr\_rdreq | output | 1 | 读入口数据FIFO的读信号 |  |
| um2ddr\_valid\_rdreq | output | 1 | 读操作时信息FIFO的读信号 |  |

UM与DDR2控制器通过四个FIFO相连：出口数据FIFO（写）、入口数据FIFO（读）、命令FIFO和信息FIFO，主要包括读写两种操作，读操作和写操作都需要发送具体的操作命令来完成。

写操作时，需要先将待写入数据写至出口数据FIFO之后，再将首地址、偏移量、写命令等写入命令FIFO。在状态表的配置中，由于状态表在控制端主机，而控制端只能通过管理模块连接UM，因此需要借助UM完成状态表配置过程。每写一次数据需要通过管理模块将四个转移表项写入到四个指定的寄存器，再通过管理模块将一条写命令的信息写入寄存器UM2DDR\_COMMAND0和UM2DDR\_COMMAND1中。表项配置模块检测到UM2DDR\_COMMAND1[2]=1，并且此时ddr2um\_ready为1时，就可以执行写操作。即首先将四个寄存器的数据合成一个128位的数据写入数据FIFO，再将两个命令寄存器的命令合成后写入命令寄存器。由于通过FPGA写入DDR2的速度远远高于控制端写入NetMagic寄存器的速度，因此控制端程序可以不断地循环写入表项和命令。

读操作时，需要先将读操作的命令、地址信息写入命令FIFO。DDR2控制器将读出的数据写入到入口数据FIFO，并将返回的信息写入到信息FIFO。UM检测到信息FIFO非空时，说明有数据返回，然后才能从入口数据FIFO读数据。状态表的配置过程不涉及读操作，DDR2的读操作主要用在报文匹配中。

5.2.5 报文采集分发模块

报文采集分发模块不断地从输入控制模块读取报文，存放至其内部的报文缓冲FIFO\_PACK。并且按轮转的方法，依次将每个完整的报文分发到各匹配引擎的报文缓冲区。报文采集分发模块的结构如图5.10所示。



图5.10 报文采集分发模块结构图

报文采集分发模块的信号列表如表5.4所示，表中省略部分信号。

表5.4 报文采集分发模块信号列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名称 | 方向 | 位宽 | 描述 | 备注 |
| um2cdp\_tx\_enable | output | 1 | 标识CDP输入模块可以向UM传报文 |  |
| cdp2um\_data\_valid | input | 1 | CDP向UM传数据的有效信号 |  |
| cdp2um\_data | input | 139 | CDP向UM传送的报文数据 |  |
| trans\_rdreq | 内部信号 | 1 | 经过仲裁后产生的读报文请求 | 寄存器 |
| trans\_data\_q | output | 139 | 模块分发给匹配引擎的报文数据 |  |
| packet\_rdreq0 | input | 1 | 匹配引擎0产生的读报文请求 |  |
| packet\_wrreq0 | output | 1 | 对匹配引擎0中报文FIFO的写请求 |  |
| packet\_rdreq1 | input | 1 | 匹配引擎1产生的读报文请求 |  |
| packet\_wrreq1 | output | 1 | 对匹配引擎1中报文FIFO的写请求 |  |
| packet\_rdreq2 | input | 1 | 匹配引擎2产生的读报文请求 |  |
| packet\_wrreq2 | output | 1 | 对匹配引擎2中报文FIFO的写请求 |  |
| packet\_rdreq3 | input | 1 | 匹配引擎3产生的读报文请求 |  |
| packet\_wrreq3 | output | 1 | 对匹配引擎3中报文FIFO的写请求 |  |

报文采集分发模块采用轮转的方式将报文分发到各匹配引擎的报文缓冲区，因此需要设置一个内部寄存器FIFO\_ID，用来记录当前需要将报文转发到哪个匹配引擎的FIFO。FIFO\_ID初始值为0，表示当前报文需要转发到FIFO\_P0中，每分发一个完整的报文，需要将FIFO\_ID值加1并对4取模。

报文采集分发模块只要检测到FIFO\_ID对应匹配引擎的报文缓冲有足够空间，并且规则缓冲有空闲空间时，就将um2cdp\_tx\_enable置1，向输入控制模块请求读入报文。trans\_rdreq实质上是一个寄存器，表示FIFO\_ID对应匹配引擎的读报文请求信号，默认为0。例如，当前FIFO\_ID值为3，并且匹配引擎3请求读报文，则trans\_rdreq的值就为1，而其它匹配引擎的读请求对trans\_rdreq的取值无影响。同理，各匹配引擎报文FIFO的写信号packet\_wrreq也由FIFO\_ID决定，只有FIFO\_ID对应匹配引擎的报文FIFO才会被写入报文数据。例如当前FIFO\_ID值为2，只有packet\_wrreq2的值为1，packet\_wrreq0、packet\_wrreq1、packet\_wrreq3的值均为0。输出信号trans\_data\_q为所有匹配引擎报文FIFO提供数据，如此即可实现报文的轮转分发。

以太网最大报文长度1500字节，加上18字节的链路层地址信息共1518字节，按NetMagic内部报文格式可分为95个报文片，约1.6K。根据资源使用情况，报文分发模块的报文缓冲FIFO\_PACK设置为1000\*139bit，宽度为139，可存储十个完整的报文。每个匹配引擎的内部报文缓冲设为300\*139bit，宽度为139，可存储三个完整的报文。

5.2.6 匹配处理模块

报文匹配处理模块是整个报文检测系统的核心，它直接与报文缓冲、状态表及规则缓冲相连，实现报文匹配的功能。对于单个匹配处理模块，其模块结构如图5.11所示。



图5.11单个匹配处理模块的结构图

单个匹配处理模块的信号列表如表5.5所示。

表5.5 单个匹配处理模块的信号列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名称 | 方向 | 位宽 | 描述 | 备注 |
| data\_rdreq | output | 1 | 读报文请求 |  |
| data\_q | input | 139 | 从报文缓冲读出的报文数据 |  |
| tableram\_rden | output | 1 | 匹配时查RAM表读信号 |  |
| tableram\_address | output | 14 | 匹配时查RAM表的查表地址 |  |
| tableram\_data | input | 16 | 查表获得的转移状态ID |  |
| ddr2um\_ready | input | 1 | 控制器ready信号，高有效 |  |
| um2ddr\_command\_wrreq | output | 1 | 命令FIFO的写信号 |  |
| um2ddr\_command | output | 34 | 命令 |  |
| ddr2um\_valid\_empty | input | 1 | 标识信息FIFO是否为空 |  |
| um2ddr\_valid\_rdreq | output | 1 | 读信息FIFO的读信号 |  |
| ddr2um\_valid\_rdata | input | 7 | 信息FIFO返回的数据信息 |  |
| um2ddr\_rdreq | output | 1 | 读入口数据FIFO的读信号 |  |
| ddr2um\_rdata | input | 128 | 读ddr返回的数据 | [1] |
| rule\_usedw | input | 3 | 对应规则FIFO的空间使用情况 |  |
| rule\_wrreq | output | 1 | 写规则FIFO信号 |  |
| rule\_q | output | 30 | 规则信息 |  |

注[1]: ddr2的数据通路是128位，一次可以读出8个转移表项，即8个转移状态，具体转移到哪个状态由当前的输入字符决定。

匹配处理模块的处理逻辑较为简单，但是涉及到读报文、写规则、写结果、查表等操作，因此实现过程比较复杂。匹配引擎在报文缓冲非空时就立即读报文缓冲，开始一个完整报文的匹配处理过程。由于报文检测是基于报文内容的，所以需要去掉报文头部的信息。对以太网报文而言，链路层信息加上IP首部共34个字节，在NetMagic内部报文格式中每拍报文有128bit的数据信息，因此需要从第三拍报文的第三个字节开始匹配处理。

匹配处理过程就是一个连续查表过程，若查表中得到的某个状态命中某条正则表达式规则，说明该报文满足一定的协议特征，对该报文后面的尚未匹配的报文内容不再进行匹配，而是连续读取剩余节拍的数据，直至下一个报文首报。若查表中得到的状态一直未命中某条正则表达式规则，就接着读该报文的下一拍数据，直至到达该报文的尾部。匹配处理中若命中某条规则，则根据该规则的ID号，将Result\_RAM中该规则的命中次数加1。Result\_RAM中的结果信息可以通过控制端程序读出。一个完整报文的匹配处理逻辑如图5.12所示。



图5.12 报文匹配处理逻辑

另外，匹配处理中不论是否命中规则，在处理完每个完整报文的最后一拍数据时，都需要生成一条规则信息，写入到规则FIFO。每个匹配引擎的规则FIFO定义为5\*30bit，可以存储五条规则，每条规则信息为30位，规则信息的定义如表5.6所示。

表5.6 转发规则定义

|  |  |
| --- | --- |
| 字段 | 描述 |
| [29] | 报文来源，1表示报文来自输入控制模块，0表示报文来自UM |
| [28:25] | 报文操作模块，0000为根据端口输出，0001表示丢弃，0010表示截断，其它保留 |
| [24:17] | 报文截断控制 |
| [16:0] | 输出端口号，[7:0]对应8个物理端口，其它保留 |

在报文匹配中，未对报文进行修改，因此报文来源一律设为输入控制模块。[28:25]设为0000，根据输出端口输出。[7:0]根据需要进行设置，系统实现中分四个输入端口和四个输出端口，并且每个输入端口分别对应一个不同的输出端口。

在具体查表过程中，优先查找RAM中的状态表项，在RAM中未命中时再查找DDR2中状态表项。实际上5.2.1的重命名方法给报文匹配的寻址过程带来极大便利，对于第个匹配引擎，假设当前状态的ID值为，当前匹配的报文字符的ASCII码对应的数值为。若的ID小于20，直接访问RAM块中状态表，地址值为；否则，直接访问DDR2中状态表，地址值为。

5.2.7 规则采集处理模块

规则采集处理模块读取各匹配引擎规则FIFO中的规则，并将这些规则发送到输出控制模块的规则缓冲FIFO\_RULE。规则采集处理模块的结构如图5.13所示。



图5.13 规则采集处理模块结构图

规则采集处理模块的信号列表如表5.7所示。

同报文采集分发模块的工作原理类似，规则采集模块也采用轮转的方式依次从各匹配引擎的规则缓冲中提取规则，模块内部设置一个寄存器RULE\_ID，用来记录当前需要从哪个规则缓冲提取规则。同时，为保证报文与相应规则严格的对应关系，RULE\_ID的初始值也必须设置为0。每采集一次规则，需要将RULE\_ID值加1并对4取模，表示取下一条规则的缓冲ID。

规则采集处理模块根据CDP输出控制模块规则缓冲FIFO\_RULE的使用情况来采集规则，只要FIFO\_RULE有剩余空间，规则采集模块就根据RULE\_ID的值读取对应匹配模块的规则缓冲中的规则，并将该规则写入到FIFO\_RULE。输出控制模块则持续读取FIFO\_RULE中的规则，并根据规则转发输入输出模块之间FIFO中的报文至相应物理端口。

表5.7 规则采集处理模块信号列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名称 | 方向 | 位宽 | 描述 | 备注 |
| rule\_rdreq0 | output | 1 | 读匹配模块0的规则缓冲FIFO\_R0的读信号 |  |
| rule\_q0 | input | 30 | 读模块0规则缓冲返回的规则信息 |  |
| rule\_rdreq1 | output | 1 | 读匹配模块1的规则缓冲FIFO\_R1的读信号 |  |
| rule\_q1 | input | 30 | 读模块1规则缓冲返回的规则信息 |  |
| rule\_rdreq2 | output | 1 | 读匹配模块2的规则缓冲FIFO\_R2的读信号 |  |
| rule\_q2 | input | 30 | 读模块2规则缓冲返回的规则信息 |  |
| rule\_rdreq3 | output | 1 | 读匹配模块3的规则缓冲FIFO\_R3的读信号 |  |
| rule\_q3 | input | 30 | 读模块3规则缓冲返回的规则信息 |  |
| cdp2um\_rule\_usedw | input | 5 | 返回输出控制模块的规则FIFO空间使用情况 |  |
| cdp2um\_rule\_wrreq | output | 1 | 写输出控制模块的规则缓冲的写信号 |  |
| cdp2um\_rule | output | 30 | 写输出控制模块的规则缓冲的规则信息 |  |

5.3 实验与分析

5.3.1 实验环境

实验环境整体部署如图5.14所示。



图5.14 实验环境整体部署

实验主要过程是控制端主机将状态表配置到NetMagic平台，将报文配置到测试仪1中，测试仪1向四个光口发送报文，经NetMagic匹配处理后转发至四个电口，测试仪2从四个电口接收报文数据，控制端从测试仪2读取相关信息。

图中控制端主机功能包括编译规则集生成NFA，子集合算法生成DFA，DFA最小化，根据DFA求解转移概率矩阵，根据转移概率矩阵求解稳态向量，使用稳态向量的结果对DFA重命名，得到重命名后的DFA。在完成上述步骤之后，还需要通过控制端的主机，将硬件代码编译综合，生成SOF文件，并将SOF文件通过JTAG的方式烧入到NetMagic的FPGA芯片中。然后通过控制端程序，基于NMAC协议将DFA状态转移表配置到NetMagic中FPGA的RAM块和DDR2 SDRAM中。

控制端主机配置如下：CPU Intel Core i5-3330，主频3.00GHz，内存大小8G，主机操作系统MS Windows8。其中自动机编译器程序的开发环境为VMware上虚拟机系统Ubuntu 12.10，gcc4.6，稳态向量的计算中借助数学工具Matlab。硬件代码的编辑、编译、综合、调试、布线及烧入采用QuartusⅡ11.0，烧入方式为JTAG，硬件代码仿真采用Modelsim软件。控制端程序开发环境为VMware上虚拟机系统Ubuntu 12.10，gcc4.6，libnet-1.1.4，libpcap-1.1.1，开发平台为Qt。

NetMagic主要功能是报文的匹配转发，从四个光口接收数据，经过匹配处理后，转发至四个电口。其中来自光口S1的报文转发至电口R1，来自光口S2的报文转发至电口R2，依次类推。NetMagic型号采用NetMagic08。

网络测试仪1的主要功能是根据配置的网络报文及工作模式，按一定的速率将报文分别发送至NetMagic的四个光口。网络测试仪2的主要功能是从NetMagic的四个电口读取报文数据，并对读取的报文进行统计。

网络测试仪1和网络测试仪2均采用IXIA公司的Optixia XM2 10G测试仪。该测试仪支持四个百兆口、四个千兆口及两个万兆口，并且都支持收发操作。由于NetMagic的端口速度不超过1Gbps，因此实验中两个测试仪均使用千兆口。

实验中正则表达式规则集采用Linux L7-filter中的协议规则集。测试网络报文有两组，第一组是1999年DARPA的第一周周一的外网数据包（323M，共1362869个报文）。第二组数据是我们于2011年11月15日21:00，在华南某OC192链路上被动获取的报文CAPTURE（2.3G，共5293074个报文）。

5.3.2 实验过程

首先是编译规则集生成自动机的过程。L7 filter protocols中共有114条规则，单独编译时均能生成完整的自动机，但是将所有规则集一起编译将会产生状态爆炸。实验中用服务器将前57条规则集一起编译时，由于存储空间限制，产生了一个包含五千多万个状态的不完整DFA。

而在本系统中，硬件平台NetMagic的存储资源非常有限，二级存储器DDR2 SDRAM只有512Mb，每个状态需要存储空间512B，因此最多可存储128K个状态表项。因此自动机的规模不能超过128K，即131072。我们随机选取部分规则集进行编译，对生成自动机的规模进行统计，结果如表5.8所示。

表5.8 部分规则集及自动机规模

|  |  |  |
| --- | --- | --- |
| 规则数量 | NFA | DFA |
| 10 | 171 | 482 |
| 20 | 538 | 1905 |
| 30 | 1083 | 110226 |
| 40 | 1369 | 1293970 |

从表5.8可以看出自动机规模增长得很快，并且与规则数目不是简单的线性关系。在规则数较少时，DFA状态数与NFA状态数是简单的线性关系，当规则数量增多时，NFA规模依然呈线性增长，而DFA规模则急剧膨胀。这主要取决于规则的复杂度及规则之间的交互关系，有时甚至数条较为复杂的规则集合就能使DFA状态规模达到数万。从表5.8的数据可以看出，前三组规则的自动机规模适合于本系统的硬件平台。

在生成DFA后，可以根据DFA状态转移表求转移概率矩阵，进而对DFA的状态空间进行划分。DFA状态空间划分情况如表5.9所示。

表5.9 DFA状态空间划分

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 规则数量 | DFA | 过渡集合状态数 | 闭集合个数 | 闭集合平均状态数 |
| 10 | 482 | 427 | 3 | 18 |
| 20 | 1905 | 1570 | 12 | 28 |
| 30 | 110226 | 105359 | 76 | 64 |
| 40 | 1293970 | 1242261 | 661 | 78 |

从表5.9可以看出，过渡状态在DFA中所占的比例非常大，基本上超过90%。另外随着DFA规模的急剧增长，闭集合的数目也急剧增长，但是闭集合中状态数目增长幅度较慢。并且在百万级规模的DFA中，闭集合中平均状态数不超过100，这种小规模的闭集合对稳态向量的计算非常有利。

状态空间划分后，可以根据状态空间划分情况对转移概率矩阵进行行列变换，得到如图4.6的矩阵形式，进而利用4.3.3节的计算方法求解各个状态机的稳态向量。计算中关键是对大规模矩阵的求逆，这个过程可借助Matlab等数学工具实现。最后，根据稳态向量对DFA进行重命名操作，得到重命名后的DFA。进而利用控制端程序将DFA状态转移表项配置到NetMagic的FPGA片内RAM和片外DDR2 SDRAM中。至此，完成状态表配置工作。

在进行报文检测之前，首先需要了解NetMagic的端口转发速率，产品说明书上NetMagic的电口和光口速率均为1Gbps。我们可以通过报文转发实验对端口速率进行测定，测试时NetMagic只做转发，不做匹配。具体方法是测试仪1以不同的速率向NetMagic的四个光口发送一定数量报文数据，测试仪2统计从四个物理端口读取的报文数，进而计算端口速率。将报文长度固定设为500字节，发送报文数目设置为1000000。由于NetMagic是以轮转的方式转发各端口的报文，因此各输出端口的速率基本相同，这里只统计单个端口的丢包情况，统计结果如表5.10所示，其中接收速率是通过发送速率和丢包率计算出的。

表5.10 报文转发时NetMagic端口丢包统计

|  |  |  |  |
| --- | --- | --- | --- |
| 发送速率  (Gbps) | 接收报文数 | 丢包率 | 接收速率  (Gbps) |
| 1.0 | 725481 | 28% | 0.725 |
| 0.9 | 920971 | 8% | 0.828 |
| 0.88 | 961863 | 4% | 0.845 |
| 0.86 | 999274 | 0.1% | 0.859 |
| 0.84 | 1000000 | 0 | 0.84 |

从表5.10可以看出，NetMagic端口转发速率并没有达到1Gbps，测试仪1以1.0Gbps的速率发报文时，测试仪2只能以0.725Gbps的速率接收报文。随着发送端发送速率的下降，丢包率也在急剧下降。当发送速率为0.86Gbps时，丢包率只有0.1%，当发送速率降为0.84时，测试仪2收到所有报文。因此我们可以将0.86Gbps大致作为NetMagic一个端口的实际报文转发速度，在报文匹配处理中也是采用类似的方法来确定实际吞吐量。因此，在报文匹配处理中，测试仪1发送报文的速率不能超过0.86Gbps。

5.3.3 性能分析

在正则表达式匹配的系统中，一般有两个性能指标，吞吐量和单位存储代价。单位存储代价可通过理论分析得出，匹配引擎的存储代价主要是对自动机状态表项的存储消耗。包括两部分：一级存储器RAM资源消耗和二级存储器DDR2的消耗。在实现中使用4个RAM块，每个RAM块10KB，共计40KB，RAM块每兆字节约80元人民币。DDR2的存储资源为512Mb，以金士顿DDR2 800为例，每兆字节存储代价约为0.1元人民币。所以整个匹配引擎的存储代价为每兆字节0.15元人民币，每位存储代价接近于DDR2，而远低于RAM，更远低于TCAM这类昂贵的存储器件。

3.3节已经分析过，在资源充裕的条件下，即采用更高级的FPGA作为一级存储器，SRAM或DDR3作为二级存储器，在高级存储器命中率为90%时可达到52Gbps的匹配性能，当前研究的性能如表5.11所示。

表5.11 与当前研究成果的性能对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实现方法 | FPGA[23] | GPU[34] | TCAM[27] | 两级存储技术 |
| 性能（Gbps） | 10 | 55 | 10 | 52 |

由表5.11可以看出，除了GPU加速技术外，基于两级存储的匹配技术具有很大的性能优势。而GPU加速技术只能解决简单的正则表达式匹配问题，另外它需要有开发平台的支持。

而在我们的实现中，匹配性能严重受限于硬件平台的器件速度。在NetMagic平台中，每个端口最大传输速度为1Gbps，因此实验中从四个端口注入数据，最大传输速度不会超过4Gbps。而实际测试中，每个端口仅能达到0.86Gbps，因此最大传输速度不会超过3.44Gbps。

实验分别以表5.8中前三组规则集作为模式集输入，以DARPA99的外网数据包及实际捕获的报文CAPTURE作为报文输入，统计得到整个系统的匹配性能如表5.12所示。

表5.12 匹配性能统计（单位：Gbps）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 规则数目 | Mon | Tue | Wed | Thu | Fir | 平均 | CAPTURE |
| 10 | 2.36 | 2.19 | 2.44 | 2.07 | 2.41 | 2.29 | 2.47 |
| 20 | 2.38 | 2.05 | 2.29 | 2.32 | 2.11 | 2.23 | 2.64 |
| 30 | 2.43 | 2.27 | 2.19 | 2.38 | 2.30 | 2.31 | 2.52 |

从表5.12可以看出，基于DARPA数据集的匹配，基本上可达到2.3Gbps的吞吐量。并且吞吐量不随自动机规模的增加而变化，这是由其理论模型决定的，在第三章中已经说明，状态访问概率分布不随自动机规模的增加而变化。因此在自动机规模增加时，对高级存储器的命中率没有影响。

另外，对DARPA报文的匹配性能明显低于对CAPTURE报文的匹配性能2.5Gbps，这主要是因为DARPA报文数据量小，报文较短，稳态特性体现的没有匹配CAPTURE报文时明显。

5.4 小结

本章基于NetMagic开发平台，以FPGA片内RAM作为两级存储器，片外DDR2 SDRAM作为二级存储器，实现了基于两级存储的匹配技术。另外，利用FPGA内嵌多块可并行访问RAM的特性，实例化多个匹配引擎，使匹配速度进一步线速提升。但是受平台器件性能的限制，实际的匹配性能只能达到2.5Gbps，与主流的匹配引擎还有很大差距，主要原因有以下几方面：（1）NetMagic中可供使用的存储资源过少，目前只实例化4个匹配引擎，每个引擎用一块10KB的RAM存储20个状态表项。若选用存储资源更加丰富的FPGA，则匹配速度会随匹配引擎数目的增加而线性提升。（2）编译后NetMagic实际工作频率只有125MHz，有待通过器件升级达到更高的工作频率。（3）DDR2访存速度与RAM差距过大，即使少量的访问DDR2也会使整体性能下降很多，可采用更加高速的外部存储器件如DDR3、SRAM等。

结 束 语

正则表达式的匹配是深度报文检测、协议识别等系统的核心技术。目前的研究主要集中于两方面：1）性能的提升，利用高速存储或加速器件如FPGA、TCAM、GPU等强大的并行能力对性能进行提升；2）内存需求的压缩，基于状态表的冗余或改进自动机的结构来压缩状态表空间，以牺牲性能为代价压缩存储需求。随着系统中规则数目的增加及网络带宽的提升，内存和吞吐量需求同时提高，存储和性能的矛盾将进一步加剧。

为解决存储和性能的矛盾，本文从匹配引擎整体结构入手，提出一个新的方向：基于多级存储的匹配引擎。研究发现，在进行大量报文匹配时，自动机中状态访问概率分布呈现Zipf特性，即只有极少数状态是经常被访问的，绝大部分状态访问概率极小。进而将高访问概率的状态表项配置在小容量的高速存储器中，低访问概率的状态表项配置在大容量的低速存储器中。高速存储器可解决性能问题，低速存储器可解决存储需求。我们采用马尔可夫链理论对报文匹配过程中的状态迁移进行建模，以稳态向量作为对状态访问概率的预测。实验证明，马尔可夫链的预测准确性很高，可以作为报文匹配的理论模型。

理论分析说明，在资源充裕的条件下，两级存储的匹配技术可以达到52Gbps。我们基于开放式网络开发平台NetMagic实现了基于两级存储的匹配引擎，但受限于该平台的工作频率、端口数目、存储器件性能等条件，该平台上的匹配引擎仅能达到2.5Gbps的吞吐量。实际上，该平台在不做匹配，只进行报文转发时最大吞吐量也仅为3.44Gbps。另外，每位存储价格接近于低速存储器DDR2，远远低于其它高速存储器件的单位存储代价。本文提出的基于两级存储的匹配方法适用于所有需要状态转换，而状态转换是性能瓶颈的应用，如基于特征码的匹配、基因比对等。

虽然目前系统能以较低的存储代价获得一定的吞吐量，但较目前主流的研究性能还是偏低。这主要受NetMagic平台的工作频率和硬件资源的限制，未来工作中将选取更加高速的FPGA器件，并搭载DDR3或SRAM等高速的外部存储器件。

致 谢

衷心地感谢我的导师陈曙晖老师。陈老师有着深厚的学术修养和丰富的科研经历，在两年半的硕士阶段给予我莫大的帮助。在课业学习方面，悉心地为我进行选课方面指导，并督促课业上的进步。在实践方面，研一上课阶段就开始布置一些编程任务，对网络、Linux系统、数据结构等方面的内容查遗补缺。在科研方面耗费大量精力对我进行指导点拨，鼓励我拓展视野、多做一些大胆的尝试，多表达自己的想法。陈老师踏实肯干、追求卓越的作风，也为我树立了最好的学术榜样。最难能可贵的是陈老师在繁重的课题压力下还能为我创造宽松的学习环境，使我能潜心进行课题研究，顺利完成硕士阶段的课题任务。再次表示感激之情。

感谢毛席龙老师、毛健彪师兄、胡勇庭同学和张彥龙同学在NetMagic平台上对我的诸多指导和帮助。

感谢陈洪义同学在学习、生活中的方方面面对我的支持与帮助，感谢一起走过的青春。

感谢系所领导两年来的培养和关怀，感谢615全体教师的栽培与指导。

感谢张硕师兄、谢裴岱师兄、程冕师兄在学习和生活中对我的指导和关怀。

其次，感谢学院领导和学员队队干部对我们的支持和帮助，感谢与五队同学们在一起的奋进的岁月。

最后，感谢我的双亲和李敏，你们永远是我的最爱。你们无时无刻不在的关心、鼓励和宽容是我最大的动力，谢谢！

参考文献

1. 张树壮，罗浩，方滨兴. 面向网络安全的正则表达式匹配技术[J].软件学报，2011, 22(8): 1838-1854.

ZHANG S Z, LUO H, FANG B X, Regular Expressions Matching for Network Security[J], Journal of Software, 2011,22(8):1838-1854.

1. Application Layer Packet Classifier for Linux [EB/OL]. 2009. http://l7-filter.sourceforge.net.
2. Snort user manual, the snort project [EB/OL]. http://www.snort.org/

Assets/82/snort\_manual.pdf.

1. Introduction to bro [EB/OL] http://www.bro-ids.org/wiki/index.php/Bro.
2. TippingPoint X505. 2008. http://www.tippingpoint.com/products\_ips.html.
3. Cisco IOS IPS deployment guide. http://www.cisco.com.
4. Suhyung Jo and Dowon Hong, The Study of Text Extraction for Forensic Data. Journal of Information Assurance and Security，2010(5)：384-391.
5. Yinghua Guo，Jill Slay，Jason Beckett. Validation and verification of computer forensic software tools—Searching Function. Digital investigation 2009(6):S12–S22.
6. Jooyoung Lee，Sungkyung Un，Dowon Hong. Improving Performance in Digital Forensics: A Case Using Pattern Matching Board. International Conference on Availability Reliability and Security (2009):1001-1005.
7. McCulloch W, Pitts W. A Logical Calculus of Ideas Immanent in Nervous Activity[J]. Bull. Math. BioPhys, 1943, 5: 115-133.
8. Thompson K. Regular expression search algorithm [J]. Communications of the ACM, 1968, 11 (6): 419-422.
9. 陈明奇, 吴秋新等. 入侵检测[M]. 第3版. 北京: 人民邮电出版社, 2006. 108-110
10. PCRE[EB/OL]. http://www.pcre.org. 2010.
11. POSIX regex[EB/OL]. http://www.opengroup.org/onlinepubs/000095399/basedefs/xbd\_chap09.html. 2010
12. J.E.Hopcroft，J.D.Ullman．自动机理论、语言和计算导论[M]．刘田，姜晖，王捍贫．第二版．北京：机械工业出版社，2004，1-366.
13. Thompson K. Regular expression search algorithm[J]. Communications of the ACM, 1968, 11 (6): 419-422.
14. Hopcroft J.E, Motwani R, Ullman J.D. Introduction to Automata Theory, Languages, and Computation[M]. Addison-Wesley, 2001.
15. M.Becchi and P.Crowley. Efficient Regular Expression Evaluation: Theory to Practice. In Proceedings of ACM/IEEE ANCS, November 2008.
16. Sidhu R., Prasanna V.K. Fast Regular Expression Matching using FPGAs[C]. in 9th Annual IEEE Symposium on FCCM, Washington, DC, 2001: 227-238.
17. CHO Y H, NAVAB S, MANGIONE-SMITH W H. Specialized hardware for deep network packet filtering[C]. Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications. London: Springer-Verlag, 2002
18. CLARK C R, SCHIMMEL D E. Scalable pattern matching for high speed networks[C]. Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.Washington, DC: IEEE, 2004:249
19. BADRAN T F,AHMAD H H,ABDELGAWAD M A. reconfigurable multi-byte regular-expression matching architecture for signature-bascd intrusion detection[C]. // ICTTA 2008: Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications. Washington, DC: IEEE Computer Society, 2008:1-4
20. KORENEK J, KOSAR V. Efficient mapping of nondeterministic automata to FPGA for fast regular expression matching[C]. DDECS: Proceedings of IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Washington, DC:IEEE, 2010:54
21. Kunyang Peng, Siyuan Tang, Min Chen, Qunfeng Dong: Chain-Based DFA Deflation for Fast and Scalable Regular Expression Matching Using TCAM. [ANCS 2011](http://www.informatik.uni-trier.de/%7Eley/db/conf/ancs/ancs2011.html#PengTCD11): 24-35.
22. Kunyang Peng, Qunfeng Dong, Min Chen: TCAM-based DFA Deflation: A Novel Approach to Fast and Scalable Regular Expression Matching. IWQoS 2011: 1-3
23. I. Sourdis and D. Pnevmatikatos. Pre-decoded cams for efficient and high-speed nids pattern matching.In Proc. FCCM, 2004.
24. Chad R. Meiners, Jignesh Patel, Eric Norige, Eric Torng, Alex X. Liu, Fast Regular Expression Matching using Small TCAMs for Network Intrusion Detection and Prevention Systems. In USENIX Security 2010.
25. Lei Wang, Shu Hui Chen, Yong Tang, Jinshu Su, Gregex: GPU Based High Speed Regular Expression Matching Engine. In [Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)](http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5975322), 2011.
26. Geza Szabo, Istvan Godor, Andras Veres, Szabolcs Malomsoky, Sandor Molnar , Traffic Classification over Gbit Speed with Commodity Hardware. IEEE J. Communications Software and Systems, vol. 5, 2010.
27. [Sudheer Ponnemkunnath](http://www.springerlink.com/content/?Author=Sudheer+Ponnemkunnath) and [R. C. Joshi](http://www.springerlink.com/content/?Author=R.+C.+Joshi), Efficient Regular Expression Pattern Matching on Graphics Processing Units. [Communications in Computer and Information Science](http://www.springerlink.com/content/1865-0929/), 2011, Volume 168, Part 1, 92-101.
28. Vasiliadis G., Antonatos S., Polychronakis M., et al. Gnort: High Performance

Network Intrusion Detection Using Graphics Processors[C]. in Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, Cambridge, MA, USA, 2008: 116-134.

1. Smith R., Goyal N., Ormont J., et al. Evaluating GPUs for Network Packet Signature Matching[C]. in Proceedings of the International Symposium on Performance Analysis of Systems and Software, 2009.
2. Naghmouchi J., Scarpazza D.P., Berekovic M. Small-ruleset regular expression matching on GPGPUs: quantitative performance analysis and optimization[C]. in Proceedings of the 24th ACM International Conference on Supercomputing, Tsukuba, Ibaraki, Japan, 2010: 337-348.
3. flex[EB/OL]. http://flex.sourceforge.net/. 2010.
4. Chen S H, Su J S, Fan H P, et al. An FSM State Table Compressing Method Based on Deep Packet Inspection [J]. Journal of Computer Research and Development,2008, 45 (8) : 1299-1306

(陈曙晖，苏金树，范慧萍等. 一种基于深度报文检测的FSM状态表压缩技

术[J].计算机研究与发展，2008,45(8): 1299-1306)

1. Becchi M, Cadambi S. Memory-Efficient regular expression search using state merging. In: Proc. of the IEEE Infocom. Anchorage, 2007. 1064-1072.
2. Kong SJ, Smith R, Estan C. Efficient signature matching with multiple alphabet compression tables. In: Proc. of the 4th Int’l Conf. on Security and Privacy in Communication Networks. Istanbul, 2008. 1-10.
3. Ficara D, Giordano S, Procissi G, Vitucci F, Antichi G, Pietro AD. An improved DFA for fast regular expression matching. ACM SIGCOMM Computer Communication Review, 2008, 38(5):29-40.
4. Sailesh Kumar，Sarang Dharmapurikar，Fang Yu，Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In: SIGCOMM’06. Pisa，Italy: September 2006，11-15. http://www.arl.wustl.edu/~sarang/fp289-kumar.pdf.
5. Ficara D, Giordano S, Procissi G, et al. An improved DFA for fast regular expression matching. ACM SIGCOMM Computer Communication Review, 2008, 38(5):29-40.
6. YU F, CHEN Z, DIAO Y, et al. Fast and memory-efficient regular expression matching for deep packet inspection [J], Architecture for Networking and Communications Systems, 2006. ANCS2006. ACM/IEEE Symposium on. IEEE, 2006, 93–102.
7. BECCHI M, CRWOLEY P. A Hybrid Finite Automaton for Practical Deep Packet Inspection[C]. ACM Computer Communication Networks, CoNEXT 2007, December 10-13, 2007, New York, NY, U.S.A.
8. ZIPF G K. Human Behavior and the PrincipIe of Least Effort[M]. Cambridge，MA：Addison-wesIey，1949.
9. [DARPA Intrusion Detection Data Sets - MIT Lincoln Laboratory](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CCgQFjAA&url=%68%74%74%70%3a%2f%2f%77%77%77%2e%6c%6c%2e%6d%69%74%2e%65%64%75%2f%6d%69%73%73%69%6f%6e%2f%63%6f%6d%6d%75%6e%69%63%61%74%69%6f%6e%73%2f%63%79%62%65%72%2f%43%53%54%63%6f%72%70%6f%72%61%2f%69%64%65%76%61%6c%2f%64%61%74%61%2f&ei=w7xnUu6bOsWFiQe-sYCABg&usg=AFQjCNFwPbEwf2Naz_PFBHyhyRx6DHj55A&sig2=rFNs9SgAuygxAPNnkuQGmQ&bvm=bv.55123115,d.aGc)[EB/OL]. http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/
10. 王梓坤. 随机过程论[M].北京：科学出版社，1965.

WANG Z K, The theory of stochastic process[M]. Beijing: Science Press,1965

1. GEBALI F. Analysis of Computer and Communication Networks[M]. Victoria: Springer Science, 2008 p.123
2. XIE A G, BEEREL P A. Efficient State Classification of Finite State Markov Chains[A][C]. DAC98,San Francisco, 1998 ACM 0-89791-964-5/98/06. p.605-610.
3. 侯振挺，郭青峰. 齐次可列马尔可夫过程[M].北京：科学出版社，1978.

HOU Z T, GUO Q F, Countable and Homogeneous Markov chain[M].Beijing:

Science Press,1978.

1. 陈志杰等. 高等代数与解析几何[M]. 第2版. 北京: 高等教育出版社, 2008. 217-258.

CHEN Z J et al. Higher Algebra and Analytic Geometry[M]. The second edition, Beijing: Higher Education Press, 2008.217-258.

1. 关治，陈景良. 数值计算方法[M]. 北京：清华大学出版社，2002.

GUAN Z, CHEN J L. Numerical Computing Analysis [M]. Beijing: Tsinghua Press, 2002

1. DAVID K, WARD C. Numerical Analysis[M]. Beijing: China Machine Press, 2005.
2. Mao Xi-Long, Li Tao, Sun Zhi-Gang, Handbook of NetMagic innovation experiment platform : Innovation platform for Next Generation Internet Architecture, Changsha, Press of NUDT, 2012.07)

（毛席龙，李韬，孙志刚， NetMagic 创新实验平台设计指南：下一代互联网体系结构创新平台，湖南长沙，国防科学技术大学出版社，2012年7月）

作者在学期间取得的学术成果

1. 陈曙晖，徐成成.《基于两级存储的正则表达式匹配技术》.2013通信学报,已录用.
2. 徐成成，陈曙晖. 国家发明专利《基于两级存储的正则表达式匹配技术》.已接受.
3. 徐成成，陈曙晖. 《基于NetMagic的正则表达式匹配技术》. 2013 第二届中国互联网学术年会，已发表.
4. 徐成成，陈曙晖.《NetMagic上的正则表达式匹配技术》. 小型微型计算机系统，已录用.