文章编号:1671-4598(2022)09-0133-07 DOI:10.16526/j.cnki.11-4762/tp.2022.09.021 中图分类号:TP393 文献标识码:A

一种实现网络入侵检测的高效算法及其实现架构

余 伟[1],田新志[2],陈 丹[3]
(1.西安思源学院 基础部,西安 710038;
2.西安思源学院 电子信息工程学院,西安 710038;
3.西安空间无线电技术研究所,西安 710100)

摘要:为了实现网络入侵检测系统中的精确字符串匹配,文章提出了一种基于叶子一附加和二叉搜索树的字符串匹配算法及其实现架构;首先采用叶子一追加算法来对给定的模式集进行处理,以消除模式之间的重叠;然后采用二叉搜索树算法提取叶子模式及其匹配向量来构建二叉搜索树,并根据每个节点的比较结果,通过左遍历或右遍历来实现字符串的精确匹配;为了进一步提高字符串匹配算法的内存效率,提出了级联二叉搜索树;最后給出了实现精确字符串匹配的总体架构和各个功能模块的架构;实验结果表明,文章提出的设计不仅在内存效率和吞吐量方面优于目前先进的设计技术,而且具有灵活的可扩展性。

关键词:网络入侵检测系统;精确字符串匹配;叶子模式;匹配向量;二叉搜索树;流水线架构;内存效率;吞吐量

An Efficient Algorithm and Its Implementation Architecture for Network Intrusion Detection

YU Wei[1], TIAN Xinzhi[2], CHEN Dan[3]
(1. Foundation Department, Xi'an Siyuan University, Xi'an 710038, China;
2. Electronic information engineering, Xi'an Siyuan University, Xi'an 710038, China;
3. Institute of Space Radio Technology, Xi'an 710100, China)

Abstract: In order to realize the exact string matching in the network intrusion detection system, based on leaf-appended and binary search tree and its architecture, a string matching algorithm is proposed in this paper. Firstly, the leaf-appended algorithm is used to process the given pattern set to eliminate the overlap between the patterns. Then the binary search tree algorithm is used to extract the leaf patterns and their matching vectors to construct the binary search tree, By the comparison results of each node, the exact matching of strings is achieved through the left traversal or right traversal. To further improve the memory efficiency of the string matching algorithm, a cascaded binary search tree is proposed. Finally, the overall architecture and architecture of each functional module are given to realize the exact string matching. Experimental results show that the proposed design is not only superior to the current advanced design techniques in terms of the memory efficiency and throughput, but also has flexible extensible.

Keywords: network intrusion detection system; exact string matching; leaf pattern; matching vector; binary search tree; pipelined architecture; memory efficiency; throughput

0 引言

随着互联网发展成为一个巨大的全球开放网络,对互联网的攻击从未停止过,从黑客攻击、邪恶的探测攻击到网络诈骗,这些攻击甚至无需资金投入就能实现,而且可以在世界上任何地方发起;最常用的网络保护系统是防火墙和网络入侵检测系统(NIDS, network intrusion detection system)[1-2]。它们安装在网络的边界,以检查和监控流入和流出的网络流量。防火墙只执行网络的第3层或第4层过滤,基于数据包头部来处理数据包。而NIDS不仅提供第3层或第4层过滤,还提供了第7层过滤。NIDS搜索数据包头部和有效载荷以识别攻击模式(或签名)。因此,NIDS可以检测和阻止网络上传播的有害内容,识别出攻击的模式,然后采取行动终止连接或向系统管理员发出警报。

随着互联网的迅速发展和攻击数量的剧增,NIDS的设计也已成为一个巨大的挑战。现有的基于软件的解决方案无法实现内存高效利用和大的吞吐量,因此,必须采用硬件辅助。基于硬件的NIDS高速字符串匹配方案主要分为3大类:基于三态内容寻址存储器(TCAM, ternary content addressable memory)、基于动态/静态随机存取存储器(DRAM/SRAM, dynamic/static random access memory),

收稿日期:2022-03-02; **修回日期**:2022-04-11。
基金项目:陕西省教育厅计划项目(18JK104)。
作者简介:余 伟(1982-),女,陕西商洛人,硕士,讲师,主要从事计算机网络安全方向的研究。
田新志(1975-),男,湖北咸宁人,硕士,副教授,主要从事人工智能与物联网技术方向的研究。
引用格式:余 伟,田新志,陈 丹.一种实现网络入侵检测的高效算法及其实现架构[J].计算机测量与控制,2022,30(9):133-139,147.

以及基于 SRAM 逻辑的方案。尽管基于 TCAM 的引擎可以在一个时钟周期内检索结果，但它们功耗太高，并且其吞吐量受到 TCAM 相对较低的速度限制；基于 SRAM 和 SRAM 逻辑的解决方案需要多个周期来执行一次搜索。因此，通常采用流水线技术来提高吞吐量。同时，基于 SRAM 的方法受到内存限制，导致低效的内存利用，从而限制了所支持的字典大小。此外，由于 I/O 管脚数量的制约，在这些架构中很难采用外部 SRAM。由于这两个限制，即使最先进的基于 SRAM 的解决方案都不能很好地扩展以支持更大的字典。这种可扩展性一直是硬件中实现 NIDS 的主要问题。

一般来说，基于硬件的字符串匹配架构可以分为 3 类：TCAM、流水线非确定有限自动机（NFA，non-deterministic finite automaton）和流水线确定有限自动机（DFA，deterministic finite automaton）。

在文献［3］提出的基于 TCAM 的架构中，利用未使用的 TCAM 记录来提高搜索性能，选择性地生成 TCAM 表项来合并重叠的匹配条件，从而大大减少访问 TCAM 表项的数量。但整个字典都存储在 TCAM 中，它对任何输入都有确定的查找时间。这种架构的优点是内存效率高（接近 1 字节/字符）和更新机制简单，但实现的吞吐量低；在文献［4－6］提出的流水线 NFA 字符串匹配架构中，把所有给定的字符串模式组合起来构建一个 NFA，把 NFA 实现为一个流水线网络，并行地匹配多个字符串模式。这些设计尽管能够实现 8～10 Gbps 的吞吐量，但主要缺点是缺乏灵活性和所支持的字典相对较小；文献［7］提出的字段合并（FM，field merge）NFA 架构，将每个输入字符划分为 k 个位字段，并构造一个树结构的 NFA。然后，把字典“垂直地”划分为正交的位字段，每个位字段用于构造一个树形结构的 NFA，称为部分状态机（PSM，partial state machine）。此外，在 PSM 的构造过程中，维护一个逐级辅助表（ATB，auxiliary table），以将每个完全匹配状态唯一地映射到同一级别上的一组部分匹配状态。这种设计的优势在于，通过采用相对较小的 ATB，可以简化流水线遍历和 PSM 的输出合并。字段合并 NFA 完全基于内存，通过更新内存内容可以很容易地执行动态字典更新，但其内存效率依赖于字典，不同字典之间的差异很大；文献［8－9］提出的位分割（BS，bit split）DFA 架构基于[10]的算法。它将 N 个模式的字典树转换为有 O（N）个状态的 DFA，对于每个字符需要 O（1）计算，无论字典大小。然而，状态转换表中的有效（非零）状态通常非常稀少，这就导致了大量的内存浪费，使得内存带宽和延迟成为这种架构的瓶颈；为了提高内存效率，文献［11］提出了位分割算法，该算法将一个完整的 DFA 分割成几个分裂状态机。分裂状态机中的每个状态都维护一个部分匹配向量，以将状态映射到一组可能的匹配。尽管这种设计提高了内存效率，但对于具有数千个状态的大型字典来说，从 DFA 构建分裂状态机的成本非常高。尽管可以优化位分割方法的性能，但这样的优化需要为每个组规划资源，并使动态字典更新更加困难；可变步幅（VS，variable-stride）多模式匹配架构[12]也是一种基于 DFA 的方法，其主要思想是可以一步扫描可变数量的字符。尽管这是一个 ASIC 实现，但它实现了较高的内存效率，并且可以移植到 FPGA 平台。但这种设计的吞吐量较低，因为它高度依赖于字典。

上述研究的主要问题是低内存效率、低吞吐量和相对较小的支持字典。此外，这些设计或多或少依赖于字母表的大小，随着字母表大小的增加，内存效率会降低；另一方面，基于 TCAM 的方案虽然具有较高的内存效率和可扩展性，但吞吐量低和功耗高。随着模式数据库的增长，这些问题会变得更加严重。

为此，本文提出了一种预处理算法和一种可扩展、高吞吐量、内存高效的大规模字符串匹配架构。由于采用了线性流水线架构，所以具有灵活的扩展性，还能保证固定的延迟。

1 背景知识以及问题定义和记号

1.1 字符串模式匹配

字符串模式匹配（也称为精确字符串匹配或字符串匹配）是 NIDS 最重要的功能之一，它提供了内容搜索功能。字符串匹配算法是将给定字典（或数据库）中的所有字符串模式与流经设备的流量进行比较。

1.2 问题定义

字符串模式匹配问题可以表述为：给定一个长度为 K 的输入字符串、一个由全部 ASCII 字符构成的字母表 Σ（$|\Sigma|=256$）和一个由 N 个字符串模式 $\{S_0, S_1, \cdots, S_{N-1}\}$（其字符属于字母表 Σ）构成的字典，找到并返回给定输入字符串中每个模式的所有出现（如果存在的话）。

1.3 记号

图 1 所示为表 1 示例字典对应的前缀树[13-14]表示。给定一个字典和相应的前缀树表示，定义以下术语和记号：

1）字符串、模式或签名：标识任何恶意内容的独特特征；

2）*（星号）：包含一个空的任何字符串；

3）$\{n\}$：重复前一项 n 次，其中 n 是一个正整数；

4）$|S|$：字符串 S 的长度（以位表示），例如当每个 ASCII 字符占用 8 位时 $|abcd|=32$；

5）S_1 是 S_2 的前缀当且仅当 $S_2=S_1$（，如 $S_1=ab$，$S_2=abcd$，S_1 说成是 S_2 的前缀孩子，S_2 是 S_1 的前缀父亲。为简单起见，分别用孩子和父来表示前缀孩子和前缀父亲。在 $S_1=S_2$ 时，S_1 是 S_2 的前缀，反之亦然；

6）S_1 和 S_2 是不同的当且仅当 S_1 不是 S_2 的前缀且 S_2 不是 S_1 的前缀，否则，认为它们是重叠的。所有模式对都是不相交的一组模式称为不相交模式集；

7）$S_1<S_2$ 当且仅当 S_1 中的第一个非匹配字符（从左边开始）的 ASCII 值比 S_2 中的对应字符小，例如 $S_1=abcd$，$S_2=abef$（$S_1<S_2$，因为 $c<e$；

8）字符串 S 的二进制表示记为 B_S，将 S 的所有字符转

换成其相应的 ASCII 值即可得到；

9）字符串 S 的范围表示用 L 位（$L>|S|$）表示为 $R_S=[R_{S_L},R_{S_H}]$，其中 $R_{S_L}=B_S0\{n\}$ 和 $R_{S_H}=B_S1\{n\}$，满足 $n=L-|S|$。在 $L=|S|$ 时，$R_S=[B_S,B_S]$；

10）任何节点（从前缀树的根到该节点的路径表示字典中的模式）都称为模式节点。如果一个模式位于一个叶子节点上，则它就称为叶子模式节点，否则，就称为非叶子模式节点。

表 1　事例模式数据库(字典)(最大模式长度为 8)

节点	模式	节点	模式	节点	模式	节点	模式
P_1	*an*	P_5	*anchor*	P_9	*between*	P_{13}	*cat*
P_2	*and*	P_6	*bee*	P_{10}	*bet*	P_{14}	*hi*
P_3	*andy*	P_7	*be*	P_{11}	*beat*	P_{15}	*hint*
P_4	*ant*	P_8	*been*	P_{12}	*car*	P_{16}	*hire*

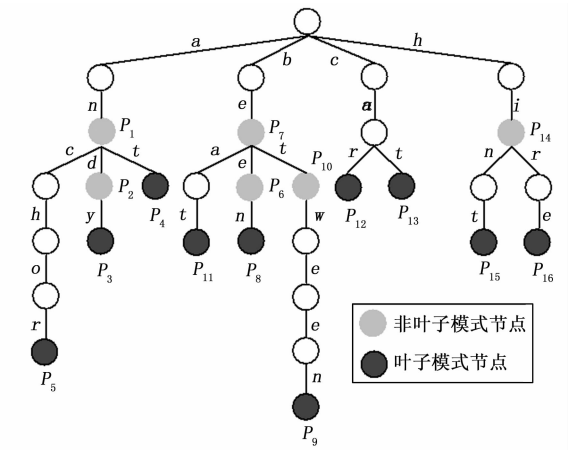

图 1　表 1 示例字典的前缀树

2　字符串匹配算法

2.1　前缀属性

字符串匹配可以构建为前缀匹配问题。考虑两个字符串 S_A 和 S_B，在 L 位数空间（$\Omega=[0\{L\},1\{L\}]$ 中，$L\geq\max(|S_A|,|S_B|)$，令 $n_A=L-|S_A|$，$n_B=L-|S_B|$ 和 $n_{AB}=abs(|S_B|-|S_A|)=abs(n_B-n_A)$。下面給出用作为本文研究基础的属性。

属性 1：给定两个不同的字符串 S_A 和 S_B，如果 $|S_A|=|S_B|$，则 S_A 和 S_B 不重叠。

属性 2：给定两个不同的字符串 S_A 和 S_B，如果它们重叠，则其中一个必须是另一个的前缀。

属性 3：给定两个不同的字符串 S_A 和 S_B，如果 S_A 是 S_B 的前缀，则 $S_A<S_B$。

2.2　叶子－追加算法

对于字符串匹配引擎来说，树搜索算法是一个很好的选择，因为查找延迟不依赖于模式的长度，而是取决于字典中模式的数量。然而，为了采用树搜索算法，需要对给定的模式集进行处理，以消除模式之间的重叠。

对此，本文提出一种叶子－追加算法来分离模式。算法以前缀树作为输入，输出一组不相交的模式。所有孩子（或非叶子）模式都与它们的父亲（或叶子）模式合并。令 L 为模式的最大长度，每个父模式都有一个长度为 L 位的匹配向量（MV，matching vector）附属于它。匹配向量是一个二进制字符串，指示父模式中包含多少个子－父模式以及它们是什么。位置 i 的值为 1 意味着有一个长度为 i 字节的子－父模式，从父模式的开头开始。例如，如果父模式是 *andy*，且它的匹配向量是 0111，则包含 3 个子模式：*an*、*and* 和 *andy*，分别对应于位置 2、3 和 4 上的 1。一个模式可以是多个父模式的子模式（前缀）。图 2 所示为两个父模式 *andy* 和 *between* 的示例合并。前缀树是从一组给定模式构建的。树的每个节点包括：1）一个叶子位；2）一个模式位；3）一个模式；4）一个匹配向量。叶子位和模式位分别确定节点是叶子节点还是模式节点。叶子－追加算法按顺序遍历前缀树，并将非叶子模式推到它们的叶子模式，图 3 所示为图 1 的前缀树在执行叶子－追加算法后的结果。一旦前缀树被附加叶子后，就收集叶子及其相关的匹配向量，以形成一个不相交的模式集。表 2 所示为表 1 示例字典在叶子－追加算法后的字典。

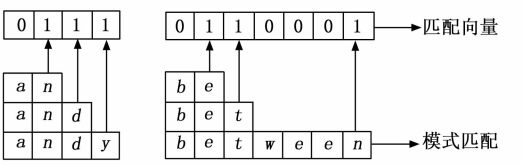

图 2　示例合并

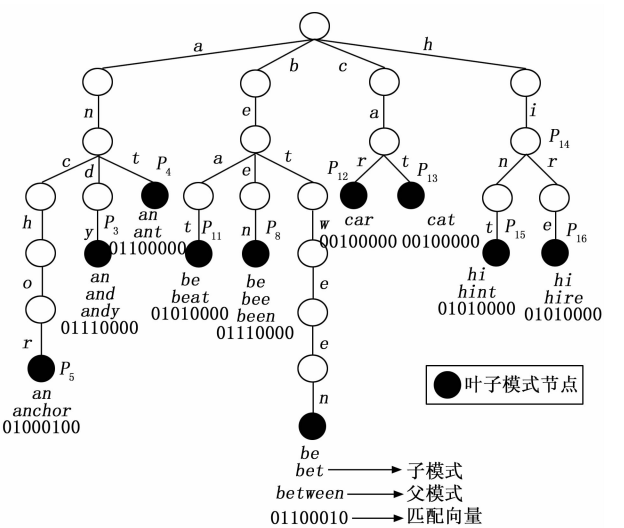

图 3　叶子－追加树

表 2　表 1 示例字典在叶子－追加算法后的模式

节点	模式	匹配向量	节点	模式	匹配向量
P_3	*andy*	01110000	P_{11}	*beat*	01010000
P_4	*ant*	01100000	P_{12}	*car*	00100000
P_5	*anchor*	01000100	P_{13}	*cat*	00100000
P_8	*been*	01110000	P_{15}	*hint*	01010000
P_9	*between*	01100010	P_{16}	*hire*	01010000

2.3 二叉搜索树字符串匹配算法

本节提出一种基于完全二叉搜索树（BST，binary search tree）[15]的高效内存数据结构。二叉搜索树算法是一种在排序列表中查找特定元素的技术。

给定字典经过叶子－追加，并提取叶子模式及其匹配向量，叶子模式用于构建 BST，BST 中的每个节点包含一个模式和一个匹配向量。构建了相应的 BST 之后，根据每个节点的比较结果，通过左遍历或右遍历来执行字符串匹配。如果输入字符串小于或等于该节点的值，则将其转发到该节点的左子树，否则转发到右子树。

为表 2 中的不相交模式集构建的完整 BST 示例如图 4 所示，示例中考虑的是最大长度为 8 个字符的模式。完整的 BST 映射如下：每一层的节点都存储在一个连续的内存块中。令 x 表示节点 A 的索引值。A 有 2 个孩子节点，索引值为 $2x$（左孩子）和 $2x+1$（右孩子）。在以 2 为基的数字系统中，$2x=\{x,0\}$ 和 $2x+1=\{x,1\}$，其中 $\{\}$ 表示串接。采用这个内存映射，不需要将孩子指针存储在每个节点上，这些指针是通过简单地将当前节点的地址与比较结果位串接起来显式地实时计算的。

输入流用 8 个字符的窗口进行处理，每次前进 1 个字符。例如有一个 8 个字符的输入字符串 anchorer 到达，匹配状态向量（MSV，matching status vector）被重置（MSV=00000000）。在树的根（P_{12}），将输入与节点的模式 car 进行比较，得到不匹配和“less than”结果。输入字符串向左遍历。在节点 P_{11}，它与模式 beat 进行比较，再次得到相同的结果。然后输入字符串被转发到左边。在节点 P_3，输入字符串匹配节点的模式 andy，得到一个匹配和一个“less than”结果。当输入字符串与孩子模式 an 匹配时，MSV 的第二位被设置为 MSV=01000000。输入字符串然后遍历到左边。在节点 P_5，输入字符串匹配节点的模式 anchor 且 MSV=01000010，这就是最后结果。

注意，一个输入字符串的所有匹配模式必须是最长匹配模式的前缀。因此，叶子－追加步骤确保这些模式包含在最长模式的同一节点中。如果输入字符串 S_I 到达该节点，则应该找到所有匹配模式，即输入字符串 S_I 到达了包含最长匹配模式的节点。

2.4 BST 字符串匹配算法的内存效率

本节分析采用提出的叶子－追加算法生成的不相交模式集（S_{DP}，set of disjoint patterns），分析中用到以下记号：

1）L：不相交模式的最大长度（以字节为单位）；

2）$\overline{L}$：不相交模式的平均长度，$1\leqslant\overline{L}\leqslant L$；

3）N_{DP}：不相交模式的数目；

4）M_{DP}：不相交模式集的大小（以字节为单位）；

5）M_{BST}：采用 BST 数据结构存储 S_{DP}所需内存的大小（以字节为单位）；

每个不相交模式有一个长度为 L 位（或 $L/8$ 字节）的匹配向量。M_{DP}和 M_{BST}分别根据式（1）和（2）计算，式（3）为 BST 字符串匹配算法的内存效率：

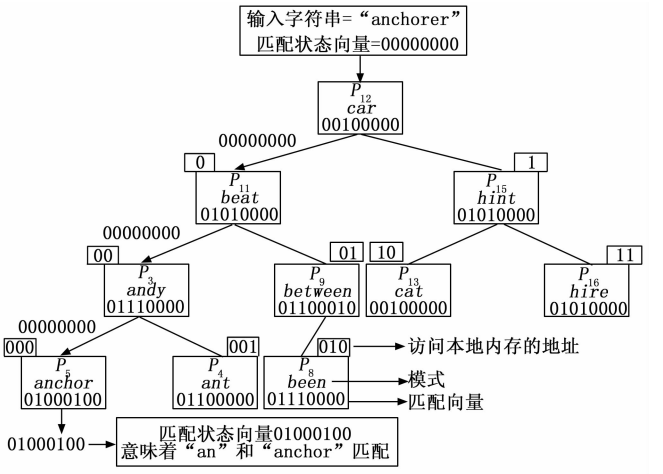

图 4 表 2 中不相交模式集的示例完整 BST

$$M_{DP}=N_D\times\overline{L} \tag{1}$$

$$M_{BST}=N_{DP}\times(L+L/8)=9N_{DP}/8\times L \tag{2}$$

$$M_{BST}/M_{DP}=9/8\times L/\overline{L} \tag{3}$$

从式（3）可见，BST 字符串匹配算法的内存效率依赖于比值 $L/\overline{L}$。当$\overline{L}=L$ 时，内存效率达到最佳情形，而当$\overline{L}=1$ 时，内存效率最差。

2.5 BST 级联

如前所述，可以通过降低比值 $L/\overline{L}$ 来提高 BST 字符串匹配算法的存储效率，具体可通过将模式拆分成长度为 L_S 的段来实现。把不相交模式集（S_{DP}）划分为分段集的集合。分段集 i 由集合 S_{DP}中所有不相交的模式的 i 个段构成。对于每一个分段集 i，构建一个前缀树 i，然后追加叶子。将叶子－追加树 i 的叶子模式收集到集合 S^i_{DP}中，$0\leqslant i<k$，$k=\lceil L/L_S\rceil$。

每个不相交的集合 S^i_{DP}由一个 BST 模块（或段 BST）匹配，这些 BST 级联如图 5 所示。BST 中的每个节点包括：（1）一个搜索键，（2）一个匹配向量和（3）一个标签。一个具有非零值的标签（或非零标签）表明存在后续段。这时，标签用于在下一个 BST 中验证匹配；BST_0 用模式的第一个段作为搜索键，而 BST_i（i（0）用前一个段的标签与模式的段 i 的串接作为搜索键。每个 BST 输出 2 个数据字段：（1）匹配状态向量和（2）匹配标签（ML，match label）。ML 是匹配整个输入字符串的节点的标签。只有当 BST_{i-1}返回一个非零标签时，才需要在 BST_i 中执行搜索。

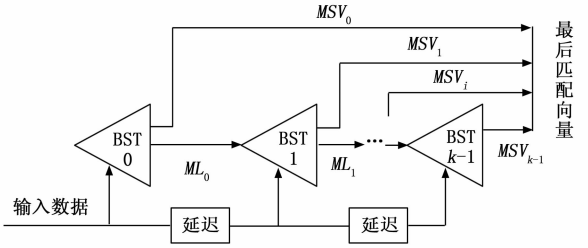

图 5 BST 级联原理的框图

3　实现架构

3.1　总体架构

本文提出一种基于流水线的二叉搜索树来实现可扩展、高吞吐量、内存高效的字符串匹配架构，实现原理框图如图 6 所示。如上所述，匹配步骤包括（1）模式匹配和（2）标签匹配，分别由模式匹配模块（PMM，pattern matching module）和标签匹配模块（LMM，label matching module）完成。输入数据流一次输入到 PMM L 个字节，输入窗口每个时钟周期前进 1 个字节。然后，PMM 根据模式数据库匹配输入字符串，同时 LMM 匹配〈前缀，后缀，匹配向量〉组合，以验证长模式并输出匹配结果。临界点是输入窗口的大小 L 与 LMM 中的记录数之间的关系。窗口大小 L 应大于或等于 LMM 的匹配延迟。因此，L 应根据字典的大小来选取。

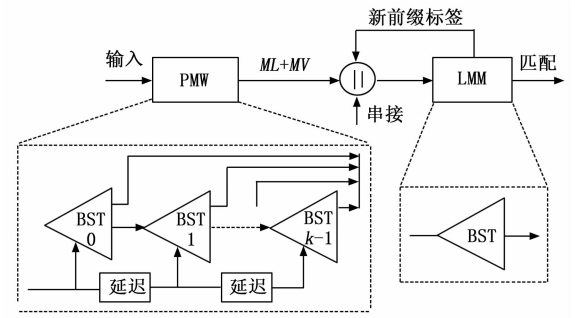

图 6　总体架构原理框图

3.2　PMM 架构

流水线用于每个时钟周期生成一个查找操作，以提高吞吐量。流水线级的数量由搜索树的高度决定。树的每一层映射到一个流水线级，流水线级有自己的内存（或表）。级联 BST 用于提高内存效率。因此，PMM 架构中使用了多个 BST。

基本流水线和 BST 的单个级的框图如图 7 所示。为了利用架构提供的双端口特性，把该架构配置为双线性流水线。这种配置使得匹配速率加倍。在每一级，内存有 2 组读/写端口，以便每个时钟周期可以输入 2 个字符串。内存中每个记录的内容包括：（1）一个模式 P，（2）一个匹配向量 ***MV*** 和（3）一个模式标签 PL。在每个流水线级，有 4 个数据字段转发自前一级：（1）输入字符串 S_I，（2）匹配状态向量 ***MSV***，（3）内存访问地址 $Addr$ 和（4）匹配标签 ML。转发的内存地址用于检索模式及其存储在本地内存中的相关数据。将此信息与输入字符串进行比较，以确定匹配状态。在匹配的情况下，匹配标签 ML 和匹配状态向量 ***MSV*** 被更新。比较结果（如果输入字符串大于节点的模式，则为 1，否则为 0）添加到当前内存地址并转发到下一级。

在流水线的每一级有一个比较器。它将输入字符串与节点的模式进行比较，并用节点的匹配向量来生成匹配状态向量。输入包括：（1）一个输入字符串 S_I，（2）一个模式 P，（3）一个匹配向量 ***MV***，（4）一个模式标签 PL 和（5）一个匹配标签。输入字符串 S_I 和模式 P 进入到字节比较器，比较器执行两个输入的字节比较。结果（M_7-M_0）输入到匹配向量解码器。解码器的输出与节点的匹配向量进行与（AND），然后结果与字符串比较器的输出进行与（AND），比较器将模式标签和匹配标签进行比较，以生成一个 8 位匹配向量（***MSV***）。

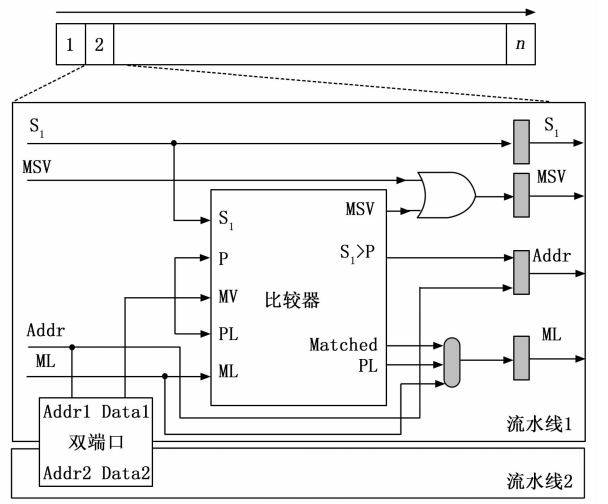

图 7　模式匹配模块及其基本流水线级的架构

3.3　LMM 架构

LMM 架构几乎与 PMM 架构相同。不同之处在于（1）只有一个 BST，（2）每个记录没有匹配向量和（3）不再需要匹配标签字段。匹配操作实际上更简单，因为它不需要处理重叠问题。此外，每个节点中搜索键的大小实质上更短，使得比较更快。在设计中，用一个完整的 BST 来搜索标签匹配表。LMM 架构及其基本的流水线级如图 8 所示。

令 N_l 表示 LMM 表中的记录数目。LMM 的延迟时间为 $\lceil \log_2(N_l+1) \rceil$。如前所述，输入窗口大小 L 必须大于或等于 LMM 的延迟，即 $L \geqslant \lceil \log_2(N_l+1) \rceil$。

3.4　字典更新

字典更新包括 2 个操作：1）模式删除和 2）新模式插入。第一种更新需要删除现有的模式。每个模式可以包括（a）多个模式和（b）仅一个模式。在情形（a），删除可以通过重置已处理模式的匹配向量的位 i 来完成，其中 i 是待删除模式的长度。例如，考虑有匹配向量 0111 的模式 andy。要删除模式 and，第 3 位设置为 0，匹配向量变为 0101。在情形（b），有 2 种可能方法：惰性删除和完全删除。惰性方法可以采用在情形（a）中描述的相同更新机制来执行。完全删除时，如果树的结构发生变化，则必须重新构建 BST，并且必须重新加载每个流水线级的整个内存内容。

在第二种更新中，把新模式插入到现有的字典中。如果新模式有父模式，则只需要更新父匹配向量来包含新模式。如果新模式不是任何现有模式的孩子，则添加一个新节点。

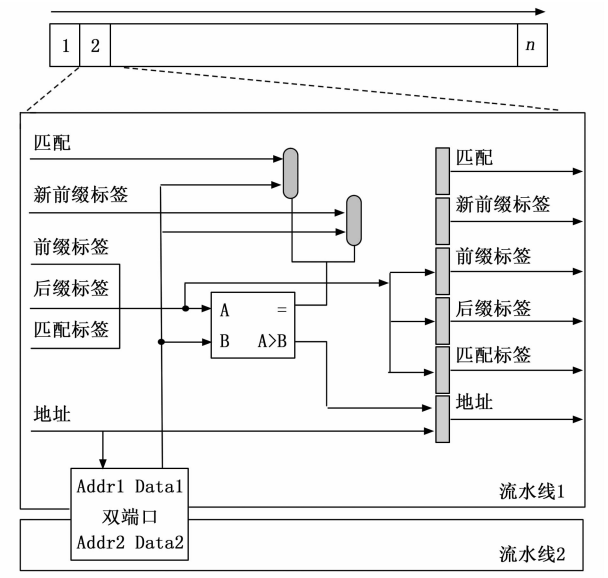

图 8 标签匹配模块及其基本流水线级的架构

3.5 模块可扩展性

本文设计的架构可以横向扩展，也可以纵向扩展，或二者同时扩展。在横向方向上，可以添加更多的流水线级来支持更大的字典。如果片上存储器的数量不够，则可以将流水线级扩展到外部 SRAM；在纵向扩展中，可以多次复制架构。这时，全部流水线有相同的内存内容，并且可以同时匹配更多的数据流。由于每个流水线中的匹配是独立执行的，所以输入流可以在每个流水线中（流内）交叉地匹配不同字符；每个流水线还可以匹配独立的输入流（流间），以增加总的吞吐量；除了这两种情形外，还可以在两个方向上扩展架构，以支持更大的字典，同时获得更高的吞吐量。

4 性能评价

4.1 实验设置

采用 Snort[16] 和 Rogets[17] 用于实验的安全字典。Snort 是恶意互联网活动的签名列表，是一种普遍的开源和跨平台 NIDS，它采用签名和包头来检测恶意 Internet 活动。作为一个开源系统，Snort 规则由网络安全社区提出，以制定广泛接受和有效的规则集；Rogets 是系统黑客通用密码破解者单词列表，该字典通常为系统黑客提供可行的密码和词表，相反，在网络安全社区中用于防止服务器/系统密码被黑。两个实验字典的统计数据如表 3 所示。

表 3 Rogets 和 Snort 字典的统计数据

字典	模式数	最大长度/byte	大小/byte
Rogets	21 667	22	160 652
Snort	8 368	323	217 680

4.2 内存效率

本节对所提出的字符串匹配算法的内存效率进行评价。两个实验字典的不同段长度值的标签匹配表中的记录数如表 4 所示。

表 4 不同段长度值的标签匹配表中的记录数

长度	4	8	12	16	20	24
Rogets	30 724	12 634	1 001	38	6	0
Snort	48 954	26 537	17 246	13 446	9 813	7 705

如前所述，段长度 L 需要大于或等于 LMM 的匹配延迟，所以在设计中用一个完整的 BST 来实现 LMM。因此，标签匹配表中的记录数不应超过 2^L-1，故分析中采用 3 个 L 值（16，20，24）。对于每个 L 值，采用不同 $L_S \in \{4, 8, 12, 16\}$ 值的级联 BST 方法实现 PMM。

表 5 所示为实验得到的结果，其中 PMM 的内存需求和 LMM 的内存需求分别表示为 M_{PMM} 和 M_{LMM}（以比特为单位），对于 L 和 L_S 的每个组合的总体设计的内存效率表示为 M_E；可见，当 L=24 和 L_S=4 时，Rogets 和 Snort 的内存效率都达到了最佳，分别为 0.56 字节/字符和 1.32 字节/字符。最大的 Rogets 和 Snort 字典分别只需要 100 kB 和 300 kB 的片上内存，而先进的 FPGA 设备需要超过 4 MB 的片上内存。因此，所提出的架构足以满足 NIDS 的实际实现。还可看到，Rogets 字典有更好的内存效率，因为 Rogets 的字母表较小为 26，而 Snort 为 256。

4.3 吞吐量

采用带 Virtex-5 FX200T 的 Xilinx ISE 12.4 为 FPGA，在 Verilog 中实现所提出的模式匹配核。表 6 所示为对于不同模式段长度（以字节为单位）的实验结果。可见，在所有情形下，逻辑资源的数量都很低（不到总资源的 15%）。对于 24 字节的段长度，实现了 197 MHz 的频率，实现了每个流水线 1.6 Gbps 的吞吐量，在双流水线架构下，吞吐量达到了 3.2 Gbps。

4.4 与目前先进设计的性能比较

在内存效率和吞吐量方面进行比较。尽管内存效率影响最大支持字典的大小，但吞吐量决定整个设计的处理速度。对于内存效率而言，较小的值表示更好的设计，而对于吞吐量，较大的值表示更好的设计。表 7 所示为本文设计和目前先进设计的比较结果。这些先进设计方法包括字段合并（FM，field merge）[7]、位分割（BS，bit split）[8] 和可变步幅（VS，variable stride）[12]。为了公平比较，对段大小为 8 字节的整个 Rogets 和 Snort 字典计算内存效率，并采用相同的 FPGA 设备实现。

可以看到，对于同一字典，本文设计在内存效率和每流吞吐量方面都优于目前的先进设计方法。与最先进的设计 FM[7] 相比，本文设计实现了超过 4 倍的内存效率提高，只需要 FM 的 1/4（对于 Rogets 和 Snort 来说）内存，同时获得 1.4 倍的每流吞吐量。与 BS 设计相比，本文设计只需要 BS 的 1/51（对于 Rogets）和 1/26（对于 Snort）的内存，同时实现 1.8 倍的每流吞吐量。此外，本文设计的内存效率和吞吐量不依赖于符号集（或字母表）的大小，而对于其他设计，当符号集增大时，它们的性能会显著下降。

表 5　不同段长度值的 Rogets 和 Snort 字典的内存效率

Sort 字典

$L=16$				$L=20$				$L=24$			
L_s	M_{PMM}	M_{LMM}	M_{Eff}	L_s	M_{PMM}	M_{LMM}	M_{Eff}	L_s	M_{PMM}	M_{LMM}	M_{Eff}
4	1 681 664	8 373 990	1.47	4	1 692 000	677 097	1.36	4	1 738 988	562 465	1.32
8	2 101 728	8 373 990	1.71	8	2 165 246	677 097	1.63	8	2 179 246	562 465	1.57
12	2 463 490	8 373 990	1.92	12	2 436 170	677 097	1.79	12	2 190 958	562 465	1.58
16	2 354 990	8 373 990	1.85	16	2 575 922	677 097	1.87	16	2 698 280	562 465	1.87
				20	2 462 830	677 097	1.80	20	2 831 408	562 465	1.95
								24	2 626 600	562 465	1.83

Rogets 字典

$L=16$				$L=20$				$L=24$			
L_s	M_{PMM}	M_{LMM}	M_{Eff}	L_s	M_{PMM}	M_{LMM}	M_{Eff}	L_s	M_{PMM}	M_{LMM}	M_{Eff}
4	716 803	2 470	0.56	4	717 671	414	0.56	4	717 795	0	0.56
8	1 549 896	2 470	1.21	8	1 551 237	414	1.21	8	1 551 237	0	1.21
12	2 186 955	2 470	1.70	12	2 187 246	414	1.70	12	2 187 246	0	1.70
16	2 816 526	2 470	2.19	16	2 818 803	414	2.19	16	2 818 803	0	2.19
				20	3 454 620	414	2.69	20	3 455 040	0	2.69

表 6　不同段长的实现结果

段长度/byte	16	20	24
频率/MHz	217	210	197
吞吐量/Gbps	3.5	3.4	3.2

表 7　本文设计、VS、BS 和 FM 的性能比较

架构		内存效率/(byte/char)	吞吐量/(Gbps/stream)
本文架构	Snort	1.32	3.2
	Rogets	0.56	3.2
FM 架构	Snort	6.33	2.28
	Rogets	2.16	2.28
BS 架构	Snort	34.1	1.76
	Rogets	28.5	1.76
VS 架构	Snort	2.4	NA

5　结束语

本文针对大规模字符串的模式匹配提出了一种叶子—追加算法和二叉搜索树的字符串匹配算法，它可以在不增加模式数量的情况下拆分给定的模式集，还提出了实现该算法的总体架构；实验结果表明，与目前先进的设计相比，本文设计获得了更好的内存效率和吞吐量；在未来的研究中，我们打算改进本文的算法，以在每个时钟周期匹配多个字符，也进一步研究标签匹配模块的替代架构。

参考文献：

[1] 李　文. 基于特征选择的网络入侵检测模型研究 [J]. 计算机测量与控制，2017，25 (8)：214－217.

[2] 陈　广，韩卫占，张文志. 基于深度学习的加密流量分类与入侵检测 [J]. 计算机测量与控制，2020，28 (1)：54－60.

[3] LIN H T，WANG P C. Fast TCAM-Based Multi-Match Packet Classification Using Discriminators [J]. IEEETransactions on Multi-Scale Computing Systems，2018，4 (4)：686－697.

[4] 杨强，林亚坤，刘永强，等. 用于大流量网络入侵检测系统的模式匹配算法和系统 [P]. CN106487803A：2017－03－08.

[5] KIM H J，CHOI K I. A Pipelined Non-Deterministic Finite Automaton-Based String Matching Scheme Using Merged State Transitions in an FPGA [J]. Plos One，2016，11 (10)：1－24.

[6] ERDEM O. Tree-based string pattern matching on FPGAs [J]. Computers & Electrical Engineering，2016，49：117－133.

[7] AVALLE M，RISSO F，SISTO R. Scalable Algorithms for NFA Multi-Striding and NFA-Based Deep Packet Inspection on GPUs [J]. IEEE/ACM Transactions on Networking，2016，24 (3)：1704－1717.

[8] 唐亚哲，李　勋. 一种基于模式分类的异构位分割状态机多模匹配方法 [P]. CN110874426A：2020-03-10.

[9] KIM H J. Memory-efficient parallel string matching scheme using distributed pattern grouping without matching vectors [J]. Electronics Letters，2016，52 (13)：1124－1126.

[10] HUNG C L，LIN C Y，WU P C. An Efficient GPU-Based Multiple Pattern Matching Algorithm for Packet Filtering [J]. Journal of Signal Processing Systems，2016，86 (2－3)：347－358.

(下转第 147 页)