面向骨干网 NIDS 的细粒度并行多模式匹配方法

邵宗有[1] 刘兴奎[2] 刘新春[3] 孙凝晖[2]
(北京科技大学信息工程学院 北京 100083)[1] (中国科学院计算技术研究所 北京 100190)[2]
(无锡城市云计算中心有限公司 无锡 214315)[3]

摘 要 随着网络带宽的快速增长,互联网正面临着日益严重的安全威胁。网络入侵检测系统(NIDS)利用模式匹配等技术对网络报文进行分析和检测,是防范网络威胁、保护网络安全的一种有效手段。但模式匹配消耗巨大的计算量,现有的技术难以满足 10Gbps 以上骨干网络 NIDS 的需求。提出了基于 Bloom filter 的细粒度并行模式匹配技术 PBPM(Parallel-Bloom-filter-based multi-Pattern Matching),PBPM 利用多个相同的 Bloom filter 分别从输入文本的不同位置处并行匹配,每个周期可完成多个字符的匹配,显著提高了匹配速率。详细讨论了在 FPGA 上的实现方式,在 Snort 2.9 规则集上的测试结果表明,PBPM 能够提供超过 20Gbps 的模式匹配需求。

关键词 多模式匹配,字符串匹配,Bloom filter,PBPM,NIDS
中图法分类号 TP393 **文献标识码** A

Fine-grained Parallel Multi-pattern Matching for Backbone Network NIDS

SHAO Zong-you[1] LIU Xing-kui[2] LIU Xin-chun[3] SUN Ning-hui[2]
(School of Information Engineering, University of Science and Technology Beijing, Beijing 100083, China)[1]
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)[2]
(Cloud Computing Center of Wuxi, Wuxi 214315, China)[3]

Abstract As the network bandwidth continuously increases, the network security has been seriously threatened by malicious behaviors and risks. Network intrusion detection system (NIDS) is one of the efficient measures to cope with intrusion threats and protect information security, which employs pattern matching techniques to analyze incoming packets and detect potential threats. However, pattern matching is such a compute-intensive task that most current techniques can't meet the demand of NIDS for backbone networks over 10Gbps speed. We proposed a novel Bloom filter based approach for pattern matching, called PBPM (Parallel-Bloom-filter-based multi-Pattern Matching). PBPM employs multiple copies of the same Bloom filter to carry out parallel matching on different positions of the input text at the same time. The fine-grained parallel approach is able to skip multiple characters per clock when implemented on FPGAs, dramatically improving pattern matching performance. Experimental results on the rule set from Snort 2.9 show that the throughput of PBPM exceeds more than 20Gbps.

Keywords Multi-pattern matching, String matching, Bloom filter, PBPM, NIDS

随着互联网的快速发展,网络上的恶意攻击行为日益增多,形形色色的病毒、木马、蠕虫层出不穷,它们对网络信息安全构成了巨大威胁。为了应对越来越严重的网络安全问题,对网络流进行实时分析和检测就显得极为必要。网络入侵检测系统(Network Intrusion Detection System, NIDS)是广泛应用的一种保护网络信息安全的手段。NIDS 通常利用模式匹配(pattern matching)技术,将报文载荷与一组模式(pattern)进行匹配,以确定报文内容是否包含可疑行为,模式可分为精确字符串和正则表达式两种。与正则表达式相比,字符串匹配简单、易于实现,因此多数规则集仍以字符串为主。本文中模式串即指字符串,两者意义相同。所有模式的集合通常称为规则集。

在深度报文检测领域,模式匹配面临着严峻的挑战。多数 NIDS 系统要求对报文的内容进行实时分析,很多研究表明,基于软件的报文检测系统仅能达到几百 Mbps 的处理速度[20],而当前骨干网带宽早已达到 10Gbps 以上,远远超过了系统的处理能力,因此必须采用专门的加速硬件才能达到对报文进行实时分析检测的目的[21]。

本文提出了基于 FPGA(Field-Programmable Gate Array)的高速多模式匹配方法,称为 PBPM (Parallel-Bloom-filter-based multi-Pattern Matching)。PBPM 利用多个并行的 Bloom filter 同时对当前输入文本的多个偏移处进行匹配,以

到稿日期:2012-10-19 返修日期:2013-01-05 本文受国家自然科学基金(61070026)资助。
邵宗有(1976—),男,博士生,主要研究方向为高性能计算,E-mail:shao@sugon.com;刘兴奎(1982—),男,博士生,主要研究方向为网络安全、可重构计算;刘新春(1968—),男,博士,副研究员,主要研究方向为可重构计算;孙凝晖(1968—),男,研究员,主要研究方向为高性能计算机体系结构、分布式操作系统。

很少的空间代价取得了每周期匹配多个字符的效果。本文第1节介绍相关工作；第2节简要介绍Bloom filter原理及其在模式匹配中的应用；第3节详细讨论PBPM的原理及其在FPGA上的实现；第4节对PBPM的性能进行了评价；最后总结全文。

1 相关研究

根据待查找的字符串数目不同，可将字符串匹配分为单模式串匹配和多模式串匹配。多模式字符串匹配就是在一个大的字符串 T 中搜索某些字符串 $\{P_1, P_2, \cdots, P_m\}$ 的所有出现位置。其中，T 称为文本(text)，P_i 称为模式串。由于NIDS通常包含成千上万个模式串，而多模式串匹配通过1次文本扫描就可以完成对所有模式串的匹配，其在NIDS系统中应用更为广泛，因此以下仅对多模式串匹配的相关研究作简要介绍。

当前的研究大致可分为如下3类：基于哈希(Hash)查找的、基于CAM(Content Addressable Memory)的以及基于AC(Aho-Corasick)算法的。哈希查找法是将模式串的全部或部分内容组织成哈希表，根据报文负载内容的哈希值查表确定是否命中哈希表中的规则。Liu等[1]以模式串的前3个字符构造哈希表，匹配时以报文内容的3个字节为单位进行哈希查询；Cho等[2]提出根据统计特征，以模式串的任意部分构造哈希表，来减少哈希冲突。基于哈希查找的方法面临的一个问题是模式串长短不一，因此需要构造多个哈希表或者采用子串预测等方法。

FPGA的片内逻辑完全并行工作，因此可以用FPGA的资源为每个模式串分别构造匹配电路，将报文负载同时与所有的模式串进行比较。这种蛮力比较方式在原理上与CAM是类似的，因此称为CAM法[3]。该方法有两个问题，一是比较逻辑耗费大量的片上资源；二是匹配速度受限制。当前FPGA的工作频率一般在200～300MHz，如果每周期处理1个字符，那么就只能达到1.6Gbps～2.4Gbps的速率。因此有大量的研究尝试改进该方法，目标是提高每周期处理的字符数[4]和减少资源的消耗[5,6]。CAM法的另一个问题是：规则集变化时，需要重新综合、布局布线，然后下载到FPGA芯片上，因此它不适合规则集频繁更新的情况。TCAM通过掩码来支持不同长度的模式串，能够支持规则集的快速更新，因此是一种常用的多模式串匹配[7,8]技术。但TCAM也有诸多缺点，如TCAM价格较高，功耗比普通RAM大得多。此外，TCAM的容量小，不适合规则较多的情形。

AC算法是一种经典的多模式匹配算法，该算法本质上是通过构造DFA(Deterministic Finite Automata，确定性有限自动机)，将字符的比较转化为自动机的状态转移。该算法有两个特点：一是扫描文本时不需要回溯，每次状态跳转可处理1个字符，因此匹配时间完全取决于文本长度；二是AC自动机消耗的空间随模式串的字符数增长而线性增加，因此它具有良好的可扩展性。由于AC自动机一般是基于memory的，且匹配过程简单，故在GPU以及多核平台上应用较多[9-11]，研究热点包括如何在多个计算节点间合理分配计算资源以及充分利用高速Cache以提高并行处理能力等。此外，由于AC自动机占用大量的空间，因此，很多研究试图压缩自动机的空间[12]。但又由于其天然的串行跳转特性，AC自动机的匹配性能受到较大限制。

2 基于Bloom filter的模式匹配

Bloom filter利用位组有效地表示1个集合，能够快速地判断1个元素是否属于该集合[13]。设要表示的集合有 n 个元素，记为 $S=\{x_0, x_1, x_2, \cdots, x_{n-1}\}$，Bloom filter用 m 位的位组 BV 来表示 S，位组 BV 初始化为全0。选择 k 个哈希函数(记为 $h_1, h_2, \cdots, h_k$)，每个函数产生的索引(index)值都在(0～$m-1$)之间，作为 BV 的位索引。对每个元素 $x_i \in S$，分别计算 k 个索引 $h_1(x_i), \cdots, h_k(x_i)$，并将 BV 中相应位 $BV[h_1(x_i)], \cdots, BV[h_k(x_i)]$ 置为1。已经置为1的位保持不变。m 和 k 值的选择将在第4节说明。

给定任意元素 y，可通过以下方法来判断其是否属于集合 S：计算 $h_1(y_i), \cdots, h_k(y_i)$，并检查 $BV[h_1(y_i)], \cdots, BV[h_k(y_i)]$ 是否全为1，如果是，那么 y 可能属于集合 S；如果不是，那么 y 必定不属于集合 S。注意，Bloom filter对于元素不属于集合的判断是准确的，但对于元素属于集合的判断则可能存在错误，也就是存在false positive错误。产生误判的原因在于被误判元素与属于集合的某些元素产生了相同的索引，以致该元素对应的 k 个索引处全为1。

利用Bloom filter进行模式匹配的传统方法是为每种长度的模式串分别构造1个Bloom filter[14,15,20]，各个Bloom filter独立地并行工作，互不干扰，如图1所示。由于存在误判，因此经Bloom filter判定为可能命中的数据仍然需要进一步的精确匹配。

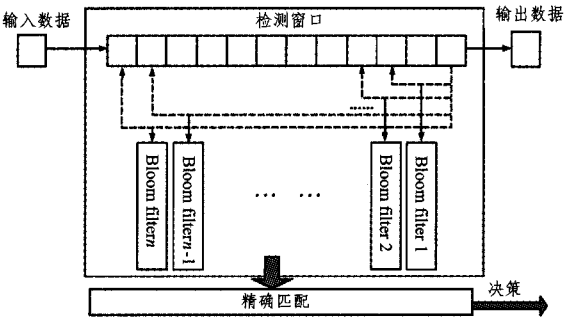

图1 基于Bloom filter的模式匹配引擎[20]

该方法存在的问题主要有3个。1)目前FPGA的工作频率一般在200～300MHz，该方法每个周期只能处理1个字符，因此只能达到2Gbps左右的处理速度，难以满足高速NIDS系统的要求。2)不同规则集模式串的长度分布往往差别很大，为了保证较低的误判率，每个Bloom filter的位组大小需要根据其包含的模式串数目进行调整，因此，当规则集变化时，该方法可能需要重新配置FPGA。3)每个Bloom filter均需要几个哈希函数，数目众多的哈希函数消耗大量的FPGA逻辑资源。

提高匹配速度的一个方法是采用 q 个完全相同的匹配引擎，每个匹配引擎分别处理不同的数据报文[15]，但该报文级的并行方法消耗 q 倍的逻辑资源，而且由于报文长度不同，导致数据报文可能被乱序提交。本文提出的PBPM是一种细粒度的并行匹配方法，克服了以上所有缺点，并适用于10Gbps以上的高速骨干网NIDS系统。

3 细粒度并行字符串匹配方法

模式串的长度分布往往是没有规律的，如Snort 2.8[16]规

则集中最长的串为 122 字节，而 Snort 2.9 规则集的最长串达 347 字节。分析 Snort2.9 规则集发现，随着长度增加，模式串的数目呈减少趋势，但不同长度的模式串个数相差很大：长度为 4 字节的串达 723 个，长度为 47 字节的串则只有 3 个；从 52～347 字节的模式串一共只有 179 个，大多数超过 64 字节的长度没有或者只有 1 个模式串。因此，为每种长度的串构造 Bloom filter 的方法缺少灵活性。

为了将所有模式串用 1 个 Bloom filter 来表示，从每个模式串中提取出相同长度的子串作为该模式串的标识，记为 Marker。例如，假设有如下只有 3 个模式串的规则集{virus，attacking，attacker}，设 Marker 长度为 4，那么该规则集每个串的 Marker 可以分别为“viru”，“atta”和“ttac”。注意，1 个模式串的 Marker 可以是其任意子串，且不一定从起始位置开始，但不能与其他模式串的 Marker 相同。在上例中，“attacker”的 Marker 还可以是“tack”，“acke”等，但不能是“atta”，因为后者已被其余模式串占用。Marker 的提取方法将在 3.3 节介绍，在此先假定规则集中任意的模式串都有唯一确定的 Marker。

显然，一段文本能够匹配模式串的必要条件是该文本中存在能够匹配某个模式串的 Marker 的子串（下文称为文本片段），如果一段文本的所有文本片段都不能匹配任意 Marker，那么该文本必然不能匹配任意模式串。因此，可以以所有模式串的 Marker 为元素来构造 1 个 Bloom filter，当出现可能匹配的文本片段时，再通过后续的精确串匹配模块作进一步的判定。

由于待匹配文本的任意位置处都有可能匹配，因此每个位置处的文本片段都必须访问 Bloom filter，故每个周期只能移动 1 个字符。为了提高匹配速度，提出的 PBPM 采用了多个相同的 Bloom filter 分别对多个连续位置处的文本片段进行并行查找。PBPM 分为两个步骤，分别是并行匹配和精确匹配，总体结构如图 2 所示。

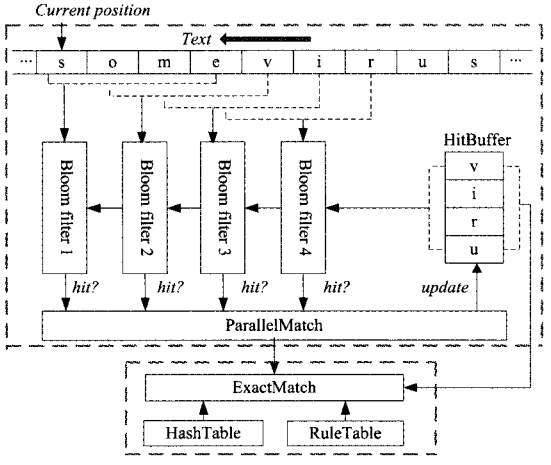

图 2　PBPM 结构框架

3.1　并行匹配引擎

观察到在实际的匹配过程中，只有少数位置处的文本片段才会真正匹配，大多数的文本片段都会被 Bloom filter 排除。利用这一特点可以构造 q 个相同的 Bloom filter，每个 Bloom filter 分别处理当前不同偏移位置处的文本片段，如图 2 上方虚线框所示。图 2 中共有 4 个 Bloom filter，故 $q=4$，即并行度为 4。并行度为 q 的并行匹配引擎在本文中也称为 q-路（q-way）匹配引擎。注意，每个 Bloom filter 均包含了所有模式串的 Marker，但它们所用的哈希函数均不相同，因此，被某个 Bloom filter 判定为命中的文本片段，可能被其他 Bloom filter 排除。利用这一特性，可以大大减少 Bloom filter 误判的概率。

并行匹配过程分为两个阶段，在图 2 所示的 ParallelMatch 功能模块的控制下工作。在第一个阶段，ParallelMatch 模块收集 q 个 Bloom filter 的命中信息，并根据命中结果确定文本移动的距离以及是否需要进一步的精确匹配。如果 q 个 Bloom filter 均显示不命中，则当前位置处的 q 个字符可直接跳过，继续从第（$q+1$）个位置开始匹配。否则，所有显示为可能命中的 Bloom filter 对应的文本片段都有可能命中某个模式串的 Marker，在这种情况下，进入第二个阶段。

第二阶段负责将第一阶段中被 Bloom filter 预测为命中的那些文本片段同时广播给 q 个 Bloom filter，只要有任意 1 个 Bloom filter 排除该文本片段，该文本片段就不可能匹配任意 Marker，即可跳过。否则，所有 q 个 Bloom filter 均不能排除，于是启动精确匹配过程（图 2 下方虚线框，见 3.2 节）。如果还有未被排除的文本片段，则以相同的过程重复第二阶段的处理。图 2 中 HitBuffer 用于暂存单个 Bloom filter 查询结果为可能命中的文本片段，通过该缓冲区，文本片段被广播到所有的 Bloom filter。

以图 2 中文本“somevirus”为例，当前 4 个 Bloom filter 的文本片段分别是“some”，“omev”，“mevi”和“evir”，假设都没有发生误判，那么所有 Bloom filter 查询均不命中，因此直接跳过当前的 4 个字符。在下一周期，从第 5 个字符继续匹配，此时，第 1 个 Bloom filter 的文本片段为“viru”，查询结果必然为命中，因为“viru”是模式串“virus”的 Marker。所以，启动第二阶段，将“viru”广播给所有的 Bloom filter，由于“viru”已经被加入到所有 Bloom filter，故没有 Bloom filter 能够排除“viru”，于是该文本片段的并行匹配过程结束，继续进行精确匹配（见 3.2 节）。

第二个阶段的目的是减少 Bloom filter 的误判率，从而尽量避免更加费时的精确匹配过程。假设每个 Bloom filter 的 false positive 概率均相同，设为 p，由于所有 Bloom filter 的哈希函数均不相同，因此当 1 个 Bloom filter 在第一阶段出现误判后，第二阶段广播给其他（$q-1$）个 Bloom filter 仍出现误判的概率为 p^{q-1}。也就是说，经过并行匹配引擎两个阶段的处理后，1 个不匹配任意 Marker 的文本片段未被排除的概率为 p^q。如果单个 Bloom filter 的误判率 $p=0.1$，那么并行度为 4 时，总的误判率 $p^q=0.1^4$，即万分之一。

注意，PBPM 的 q 个并行 Bloom filter 与图 1 中的多个 Bloom filter 是完全不同的。图 1 中的每个 Bloom filter 只对应 1 种长度的模式串，所有 Bloom filter 的输入文本均从当前位置开始，因此，输入本文每次只能移动 1 个字符。而 PBPM 的每个 Bloom filter 均保存了所有的模式串，每个 Bloom filter 的文本片段取自输入文本的不同偏移位置，因此，输入文本每次可以移动 q 个字符。

3.2　精确匹配引擎

不能被并行匹配引擎排除的文本片段，需要由精确匹配引擎 ExactMatch（见图 2 下方）作最终判断。精确匹配过程也可分为两个阶段：分别访问哈希表 HashTable 和规则表

RuleTable。两个表的结构如图 3 所示。

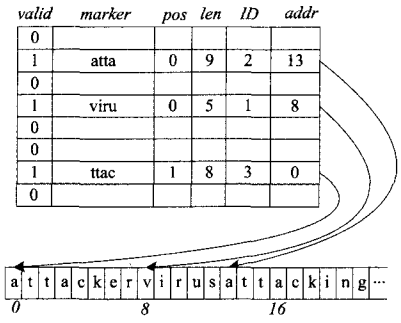

图 3 哈希表和规则表

哈希表存储所有规则的 Marker，即构造哈希表时以 marker 为关键字(key)计算索引，图 3 上方的表格是含有 3 个元素的规则表{virus，attacking，attacker}建立的哈希表，哈希表的容量为 8。解决哈希冲突的方法可以选择随机探查(random probing)、线性探查(linear probing)或冲突链(direct chaining)等[17]，略。每个哈希表项包含 6 个域：valid 表示相应项是否被占用；marker 为关键字；pos 表示该项的 marker 在模式串中的偏移位置；len 表示所在模式串的长度；ID 为模式串的唯一编号；addr 表示 marker 所在模式串在规则表中的起始地址。

规则表比较简单，只需按照任意顺序将所有模式串依次存储即可，上述规则集对应的 RuleTable 示于图 3 的下方。当然，出于性能方面的考虑，可以将每个模式串均保存在按一定整数字节对齐的边界上。

精确匹配引擎 ExactMatch 的工作过程如下。首先以并行匹配查找失败的文本片段为关键字计算哈希值，查找哈希表。如果在哈希表中未找到 marker 域与该文本片段相同的项，那么表明并行查找出现了误判，即该片段实际上不能命中任意模式串，对该文本片段的处理结束。否则，该文本片段命中了某模式串的 marker，需要从规则表中读取模式串，并与文本片段前后的文本进行逐字节的比对。模式串的长度和在规则表中的地址均可从哈希表项中获得，因此模式串可直接读取规则表获得。需要注意的是，待比对文本的位置要从当前文本片段所在的位置向前回溯一定的字节，数量由哈希表项中的 pos 域决定。如果该域为 0，那么不需回溯，而直接从文本片段所在位置开始。比如，假设当前文本片段为“ttac”，查找哈希表后得到 pos 为 1，len 为 8，则需要从该片段所在位置的上一个字符开始与从规则表中取得的“attacker”进行比对，共比对 8 字节。

由于每个模式串都有唯一的 Marker，当出现可能命中的文本片段时，只需要将文本与唯一的 1 个模式串进行比对即可，因此 1 次访存可以实现 8、16 或更多字节的匹配。有些研究采用自动机[20]等方法进行精确匹配，这些方法每次访存通常只能匹配 1 个字节，与之相比，本文方法大大提高了精确匹配部分的效率。

3.3 marker 提取

以上详细介绍了 PBPM 的原理以及实现，接下来简要介绍 marker 的长度确定及提取方法。

由 PBPM 的特点可知，只有不小于 marker 长度的模式串才能够加入到 Bloom filter 中，小于 marker 长度的串必须进行特殊处理。如果 marker 的长度太小，如只有 1 个或 2 个字节，那么 Bloom filter 出现误判的概率将很高，严重影响了匹配性能；反之，如果 marker 太长，则需特殊处理的短模式串会很多。为了提高灵活性，我们设置 1 个配置寄存器来指定 marker 的长度，该寄存器在 1～8 之间取值。所有小于 marker长度的短模式串可以利用在 FPGA 内部构造基于存储的 CAM 等方法来实现。在 Snort 规则集上的实验表明，当 marker 的长度为 4 时，即能够达到比较理想的并行处理速度。

1 个 marker 应代表唯一确定的 1 个模式串，因为如果两个模式串使用相同的 marker，那么在精确匹配过程中，1 个文本片段需要同不止 1 个模式串匹配，不仅使得处理更复杂，而且会降低匹配速度。但在有些情况下，不可能为每个模式串均找到唯一的 marker，比如若规则集中包含两条全为 0 的串，但分别为 5 字节和 6 字节，那么这两个模式串的 marker 必定相等。在这种情况下，可以只保存最短的那个模式串，当匹配该模式串时，由软件作进一步处理。

一种提取 marker 的简单算法如下。首先统计所有能够作为 marker 的子串能够代表的模式串，然后按照模式串数目递增的顺序选择子串以及相应的模式串。并以规则表{virus，attacking，attacker}为例，设 marker 长度为 4，则所有的可作为 marker 的子串包括{viru，irus，atta，ttac，tack，…，cker}，共 10 个。其中{atta，ttac，tack}可代表的模式串各有两个，其余 7 个子串只能代表唯一的模式串。确定 marker 时，先从代表最少模式串的子串开始，逐一判断该子串能代表的每个模式串是否已经确定 marker，如果尚未确定，则以该子串作为该模式串的 marker。在实例中，先判断只能代表唯一模式串的 7 个子串，该 7 个子串判断完之后，即可确定所有 3 个模式串的 marker，一种可能的结果是{viru，acki，acke}。在判断另外 3 个子串时，由于每个模式串已经确定 marker，因此结果不再变化。

该算法的复杂度为 $O(ln)$，其中 n 为模式串的数目，l 为模式串的平均长度。一般地，$l<<n$ 成立，因此 marker 提取的复杂度为 $O(n)$。在 Snort 2.8 和 Snort 2.9 规则集上的测试结果表明，除了全 0 和全 1 模式串之外，所有其余模式串均能找到唯一的 marker。

3.4 FPGA 实现

为了验证 PBPM 的可行性，在 Xilinx Virtex5 XC5VLX 110T FPGA 上实现了包含两个 8-路并行匹配引擎的 PBPM 系统。Bloom filter 的位组存储在片内 BlockRAM 中，由于 Virtex 5 FPGA 的 BlockRAM 是双端口访问的，两个并行匹配引擎可共用相同的 Bloom filter 位组。每个 Bloom filter 包含 256k 比特位，占用 8 个 BlockRAM 块。精确匹配部分的哈希表和规则表保存在片外的 8MB SRAM 中。用于 PBPM 部分的寄存器、LUT 以及 BlockRAM 分别占总容量的 5.3%、12%、47%。工作频率为 200MHz，理论上最大速率可达25.6 Gbps。

4 性能评价

本节首先给出 Bloom filter 位组大小与误判率的关系，然后从存储等资源需求和匹配速度两个方面对 PBPM 的性能进行分析，最后与现有的匹配方案作对比。

Bloom filter 的主要缺点是存在 false positive 误判，即 1

个不属于集合的元素以一定概率被判定为属于该集合，设该概率为 p。文献[18]详细推导了该概率的公式以及取得最小值的条件，本文仅引用其结论。设集合元素数目为 n，Bloom filter 位组包含 m 个位，哈希函数个数为 k，则在哈希函数均匀分布的情况下，任意 1 个不属于集合的元素被误判为属于集合的概率为：

$$p=(1-e^{-\frac{kn}{m}})^k \tag{1}$$

k 在满足下式的情况下，p 具有最小值：

$$k=\frac{m}{n}\cdot \ln 2 \tag{2}$$

将式(2)代入式(1)，得 p 的最小值 $p_{min}=0.5^k$，此时，位组 BV 中 0 和 1 的个数大致相等。

在 m 和 n 确定的情况下，式(2)给出了 k 的最优取值。但在实际应用中，元素的个数 n 是根据规则集变化的，在位组大小 m 固定的情况下，n 越小，则 k 越大，但过多的哈希函数影响计算效率，因此往往选用固定数目的函数。当元素个数 n 与 m 相比足够小时，增加 k 虽然可以进一步降低误判率，但由于误判率已经足够小，因此没有必要增加 k；而当 n 与 m 比值较大，即 Bloom filter 中的元素数目较多时，减小 k 能够降低误判率。可见，设定 k 为相对较小的固定值，即能够满足一般的需要。图 4 给出了在 n 与 m 的比值变化的情况下，不同哈希函数取不同值的误判率。可以看到，当 n/m 较小时，不管 k 如何取值，误判率均较低；当 n/m 较大时，则宜选用较小的 k 值。此外，在硬件实现中，哈希函数消耗逻辑资源，因此，我们在实现中将函数个数固定为 3 个，后面的分析也假定 $k=3$。

由式(1)可求解出，当给定误判率 p 时，位组数目 m 与元素数 n 的关系：

$$m=-\frac{k}{\ln(1-p^{\frac{1}{k}})}\cdot n \tag{3}$$

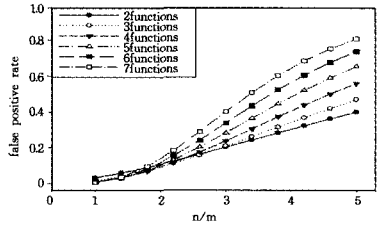

图 4　误判率 p 与函数个数 k 的关系

当 $k=3$ 时，由式(3)求得 m/n 随 p 的增加而变化的曲线，如图 5 所示。式(3)可作为确定 Bloom filter 位组大小的依据。比如，当要求 p 不大于 0.1 时，m/n 必须不小于 4.8，即位组大小至少是元素数目的 4.8 倍。同样地，当 $m/n=8$ 时，有 $p\approx0.03$，也就是说，只要为每个元素平均分配 1 个字节的空间，就可获得不大于 0.03 的误判率。

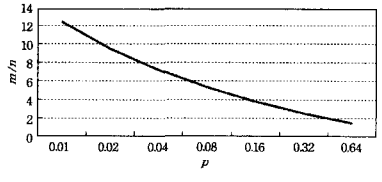

图 5　位组大小与误判率 p 的关系

4.1　存储需求

PBPM 消耗的空间主要包括并行查找的多个 Bloom filter 以及图 3 所示的哈希表和规则表。在以下的分析及测试中，均使用 Snort 2.9 规则集作为测试规则集。该规则集共有有效字符串 4345 个，总的字符数目为 75467，平均长度为 16.1字节。小于 4 字节的模式串可用 CAM 等方法实现，因此只将 4 字节以及更长的串加入 Bloom filter 中。Snort 2.9 中长度超过 4 字节的模式串共有 3922 个，去掉完全相同的模式串后，剩余 2788 个，平均长度为 21.7 字节。

如果以模式串的前缀作为 Bloom filter 的元素，那么只有 2006 个不同的前缀子串，故多条规则有相同的前缀。其中出现最多的前缀是“User”，在 69 条规则中出现。出现较多的前缀还有“docu”、“Cont”和“.php”等，分别出现了 30，20，19 次。而若以 3.3 节算法选取 marker，则除了全 0 和全 1 的模式串之外，每个规则都能找到唯一的 marker，最终有 2783 个 marker 被加入到 Bloom filter 中。

Bloom filter 位组的大小只与元素数目有关，与单个元素的长短无关。根据本节开头对误判率的分析，可为每个元素分配 1 字节空间，因此 1 个 Bloom filter 的位组占用 2783 字节。Xilinx Virtex5 系列 FPGA 的每个 BlockRAM 块大小为 4kB，因此，1 个 BlockRAM 即可实现 1-路 Bloom filter。若并行度为 8，那么 8 个并行的 Bloom filter 只需要 8 个 BlockRAM 块，共 32kB。Bloom filter 的位组占用的空间随元素的数目增长而线性增加，现代 FPGA 的片内存储已经达到几十兆比特，因此，PBSM 可支持包含多达几十万个模式串的规则集。

哈希表 HashTable 中每个表项为 16 字节，考虑到存在冲突，需要留有一定的余量，假设哈希表中最多只允许三分之一的表项有效，那么相当于每个模式串占用 48 字节。规则表 RuleTable 占用的空间与规则集总的字符数目相等。因此，两张表占用空间可估算为 $(48+L)n$，其中 n 为规则数目，L 为规则的平均长度。对 Snort 2.9 规则集来说，$L=21.7$，$n=2783$，代入 $(48+L)n$，得规则表和哈希表只要不少于 194kB 的空间即可，如以 8M 字节 SRAM 计算，则足以保存 40 个 Snort 2.9 规则集。

4.2　匹配速度

为了测试 PBPM 在真实环境下的性能，采用以下 4 组数据作为测试用例。第 1 组测试数据来自 MIT 林肯实验室的 DARPA 入侵检测实验数据集[19]；第 2 组来自某 10G 骨干网，共包含 21、527、565 个报文；第 3 组从本地局域网抓取，共 4679905 个报文；第 4 组为随机产生的 100MB 数据；4 组数据分别记为 MIT99、Backbone11、Local12 和 Random12。Bloom filter 位组的大小设定为 4K 字节，即 1 个 BlockRAM 块的大小。

首先统计了 4 组测试用例在 Snort 2.9 规则集上平均每个字符需要进行精确匹配(即 Bloom filter 排除失败)的次数，结果见表 1。

表 1　每个字符需要进行精确匹配的平均次数

Usecases	MIT99	Backbone11	Local12	Random12
# of exact matches	0.0135	0.0130	0.0170	0.0001

可以看到，Bloom filter 的排除功能非常明显，需要精确匹配的平均次数均在 0.02 以下，也就是说，文本中 98%以上的位置已被 Bloom filter 排除。从该表还可以得出，PBPM 在

各测试用例上能够达到的理论最快匹配速度：假设有足够多的并行匹配引擎，使得精确匹配模块始终处于忙碌状态，那么理论上，表 1 中精确匹配次数的倒数即为能够达到的最大加速比。以最差的 0.0170 次访问（Local12）为例，加速比为 1/0.0170≈58.8 倍，假设工作频率为 200MHz，则 PBPM 的最大处理速率超过 94Gbps。

以上给出了在不考虑资源使用的情况下，PBPM 能够获得的最大匹配速度，下面分析 1 个并行匹配引擎能够达到的加速比。定义加速比为 PBPM 每个周期能够处理的平均字符数目，也就是每周期平均跳过的字符数目。图 6 显示了并行度分别为 4、8 和 16 时的加速比。可以看到，PBPM 在所有测试用例上均取得了较明显的加速效果，且并行度越大，加速比越高。但随着并行度的增大，不同规则集上加速效果的差别也越来越大。仔细观察还可以发现，表 1 中精确匹配次数较大的用例，其加速比相对较小，这与上面的分析是吻合的。

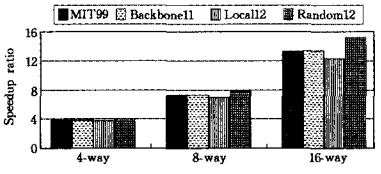

图 6　各测试用例的加速比

4.3　与其他方法的对比

为了与已有的匹配方案进行对比，统计了多种典型的模式匹配方法，并将它们的原理、实现平台以及吞吐率列于表 2。从该表中可以看到，目前多数基于专用硬件实现的 NIDS 系统的处理性能多在 1～5Gbps 之间，只有少数技术的匹配速率超过 5Gbps，离骨干网络 10Gbps 的需求差距较大。与上述方案相比，本文提出的 PBPM 方法提供了 20Gbps 以上的处理性能。

PBPM 具有良好的可扩展性和灵活性。增加并行度或者并行匹配引擎个数均能够线性提高匹配速率。通过调整 Bloom filter 位组的大小便可在匹配速率和规则集大小之间取得平衡：当规则集较小时，可适当减小位组，增加并行度，从而提高匹配速率；反之，当速度要求不是太高时，则可以减少并行度，以支持大的规则集。此外，由于 Bloom filter、哈希表以及规则表均可实时修改，因此 PBPM 支持规则集的在线更新优于 CAM 法等方案。

表 2　各算法的吞吐率

Approaches	Techniques	Platforms	Throughput/Gbps
FNP[1]	Hash	Vitesse IQ2000 NP	0.2～0.4
Cho[2]	Hash	Xilinx Virtex-4 FPGA	2.0
Dharmapurikar[14]	Bloom filter	Xilinx XCV2000E FPGA	2.5～3.2
刘[20]	Bloom filter	Xilinx Virtex4 FPGA	2.7
Sourdis[3]	CAM	Xilinx Spartan3 FPGA	2.0～6.0
Sourdis[4]	CAM	Xilinx Virtex2 FPGA	6.1～11.0
Clark[6]	CAM	Xilinx Virtex 1000 FPGA	0.8
Yu[7]	TCAM	—	2.0
Gnort[9]	AC DFA	NVIDIA GeForce 8 GPU	2.3
Scarpazza[10]	AC DFA	IBM Cell/B.E.	2.5～4.4
CardGuard[11]	AC DFA	Intel IXP1200 NP	0.5
PBPM	Bloom filter	Xilinx Virtex5 FPGA	>20

结束语　采用专用加速硬件提高处理能力是当前高性能 NIDS 系统的发展趋势，本文提出了一种基于 FPGA 的多模式匹配方法 PBPM。与现有的基于 Bloom filter 的模式匹配方法类似，PBPM 首先利用 Bloom filter 排除文本中大多数不可能命中的位置，对不能排除的，由精确匹配引擎作进一步判断。与现有方法相比，PBPM 的多个 Bloom filter 分别从文本的不同位置处匹配，因此，每个周期可同时完成多个字符的比对，极大地提高了匹配速率。此外，通过为每一个模式串选择唯一确定的 marker 作为该模式串的代表子串，大大简化了精确匹配部分的比对过程。

论文最后对 PBPM 的性能做了评价，并与已有方法进行对比。从测试结果来看，PBPM 超过了绝大多数已有的技术，可用于 10Gbps 以上的高速骨干网络的 NIDS 系统。在未来的工作中，我们将探索该方法在其他领域的应用，比如同样需要多模式匹配的生物信息学等领域。

参 考 文 献

[1] Liu Rong-tai, Huang Nen-fu, Chen Chih-hao, et al. A fast string-matching algorithm for network processor-based intrusion detection system[J]. ACM Trans on Embedded Computing Systems, 2004, 3(3): 614-633

[2] Cho Y, Mangione-Smith W. Fast reconfiguring deep packet filter for 1 + gigabit network[C]// Proc. of IEEE FCCM'05. Washington, DC: IEEE Computer Society, 2005: 215-224

[3] Sourdis I, Pnevmatikatos D. Pre-decoded CAMs for efficient and high-speed NIDS pattern matching[C]// Proc. of IEEE FCCM'04. Washington, DC: IEEE Computer Society, 2004: 258-267

[4] Sourdis I, Pnevmatikatos D. Fast, large-scale string match for a 10Gbps FPGA-based network intrusion detection system[C]// Proc. of FPL'03. LNCS 2778. Berlin: Springer, 2003: 880-889

[5] Cho Y, Mangione-Smith W. Deep packet filter with dedicated logic and read only memories[C]// Proc. of IEEE FCCM'04. Washington. DC: IEEE Computer Society, 2004: 125-134

[6] Clark C, Schimmel D. *Efficient reconfigurable logic circuits for* matching complex network intrusion detection patterns[C]// Proc. of FPL'03. LNCS 2778, Berlin: Springer, 2003: 956-959

[7] Yu F, Katz R, Lakshman T. Gigabit rate packet pattern-matching using TCAM[C]// Proc. of the IEEE ICNP'04. Washington, DC: IEEE Computer Society, 2004: 174-183

[8] Zheng Kai, Zhang Xin, Cai Zhi-ping, et al. Scalable NIDS via negative pattern matching and exclusive pattern matching[C]// Proc. of IEEE INFOCOM'10. Piscataway, NJ: IEEE, 2010: 1-9

[9] Vasiliadis G, Antonatos S, Polychronakis M, et al. Gnort: High performance network intrusion detection using graphics processors[C]// Proc. of the 11th Int. Symp. on Recent Advances in Intrusion Detection. LNCS 5230, Berlin: Springer, 2008: 116-134

[10] Scarpazza D, Villa O, Petrini F. High-speed string searching against large dictionaries on the Cell/BE processor[C]// Proc. of IEEE IPDPS'08. Piscataway, NJ: IEEE, 2008: 1-12

[11] Bos H, Huang K. Towards software-based signature detection for intrusion prevention on the network card[C]// Proc. of the 8th Int. Symp. on Recent Advances in Intrusion Detection. LNCS 3858, Berlin: Springer, 2005: 102-123

[12] Shenoy G, Tubella J, Gonzalez A. A performance and area efficient architecture for intrusion detection systems[C]// Proc. of IEEE IPDPS'11. Washington, DC: IEEE Computer Society, 2011: 301-310

（下转第 103 页）

Hadoop 计算矩阵乘法和快速傅里叶变换的时间对比。当采用小数据集时，采用 Hadoop 实现的矩阵乘法的运行时间为 40s，采用本文提出的集成了 GPU 的 Hadoop 计算时间为 23s，加速效果并不是特别明显，这主要由于 GPU 启动比较耗时造成的。当采用大数据集时，GPU 加速的效果变得很明显。如当采用中数据集时，采用 Hadoop 计算时间为 680s，采用本系统的计算时间为 120s。快速傅里叶变换的运行时间和加速效果与矩阵乘法类似。可见对于那些数据密集型同时是计算密集型的应用，集成了 GPU 的 Hadoop 平台，其加速效果好。

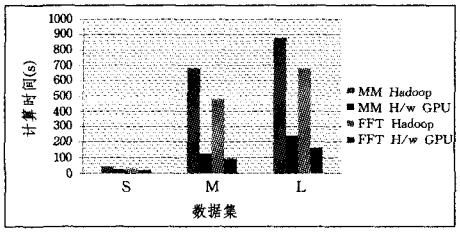

图 2　不同数据集时两种算法的计算时间

图 3 为采用集成了 GPU 的 Hadoop 系统计算矩阵乘法时进行系统优化的效果图。当不采用系统优化时，CUDA 代码的自动生成和编译占用时间约为整个计算过程的 10%（采用 S 数据集时为 20%）。当采用了系统优化后，CUDA 代码的生成和编译时间占整个计算过程的百分比几乎可以忽略。可见，系统通过采用专用线程进行代码转换和编译连接工作可以有效提高系统的运行效率，以降低非计算任务占用的系统时间。

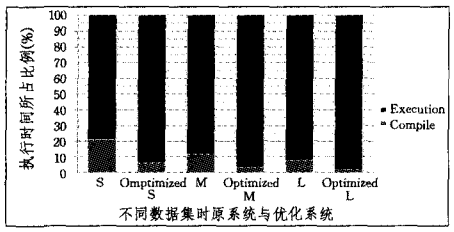

图 3　矩阵乘法算法系统优化效果

结束语　本文的核心内容是实现一种面向海量数据高性能计算的 CPU、GPU 协同计算方法，其实现形式是设计一个可以同时利用 CPU 和 GPU 计算能力的、基于计算机集群的平台。该方法可以以便捷的方式整合计算机集群中的 CPU、GPU 计算资源，从而提高计算机集群的海量数据处理性能。本文提出的计算框架已经作为核心计算方式应用于具体科研项目中，在北京市科技计划课题“能源行业海量数据成像云计算系统产业化”中用于地震数据的叠前偏移计算。实践结果表明，原本采用 CPU 计算需要数小时、采用 GPU 加速后需要 20 分钟的数据量，在此框架中（5 节点）仅需要约 5 分钟即可计算完毕，可见此框架在不增加程序设计难度的前提下，将 Hadoop 的海量数据处理能力和 GPU 的高性能计算能力良好地结合在一起，具有较好的应用价值。

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Communications of the ACM. vol. 51, 2008: 107-113

[2] Vecchiola C, Pandey S, Buyya R. High-performance cloud computing: A view of scientific applications[C]// 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks. 2009: 4-16

[3] Yoo R M, Romano A, Kozyrakis C. Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system[C]// IEEE International Symposium on Workload Characterization, 2009 (IISWC 2009). 2009: 198-207

[4] Fang W, He B, Luo Q, et al. Mars: Accelerating MapReduce with Graphics Processors[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22: 608-620

[5] Catanzaro B, Sundaram N, Keutzer K. A map reduce framework for programming graphics processors[C]// Workshop on Software Tools for MultiCore Systems. 2008

[6] Hong C, Chen D, Chen W, et al. MapCG: writing parallel program portable between CPU and GPU[C]// Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. New York, NY, USA: ACM, 2010: 217-226

[7] Zhai Yan-long, Su Hong-yi, Zhan Shou-yi. A Data Flow Optimization based approach for BPEL Processes Partition[C]// IEEE International Conference on e-Business Engineering (ICEBE 2007). HongKong, China, 2007: 410-413

(上接第 73 页)

[13] Bloom B. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426

[14] Dharmapurikar S, Krishnamurthy P, Sproull T, et al. Deep packet inspection using parallel Bloom filters[J]. IEEE Micro, 2004, 24(1): 52-61

[15] Dharmapurikar S, Lockwood J. Fast and scalable pattern matching for network intrusion detection systems[J]. IEEE Journal on Selected Areas in Communications, 2006, 24(10): 1781-1792

[16] Snort. Home Page[EB/OL]. http://www.snort.org/, 2012-07-10

[17] Morris R. Scatter storage techniques[J]. Communication of the ACM, 1968, 11(1): 38-44

[18] Broder A, Mitzenmacher M. Network applications of bloom filters: A survey[J]. Internet Mathematics, 2003, 1(4): 485-509

[19] MIT DARPA Intrusion Detecion Data Sets[DB/OL]. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999/training/week1/monday/outside.tcpdump.gz, 2011-12-02

[20] 刘威，郭渊博，黄鹏. 基于 Bloom filter 的多模式匹配引擎[J]. 电子学报，2010，38(5)：1095-1099

[21] 李伟男，鄂跃鹏，葛敬国，等. 多模式匹配算法及硬件实现[J]. 软件学报，2006，17(12)：2403-2415