doi:10.3772/j.issn.1002-0470.2014.09.009

众核处理器上的高性能网络入侵检测系统①

姜海洋②* ** 谢高岗**

(* 中国科学院大学 北京 100190)

(** 中国科学院计算技术研究所 北京 100190)

摘 要 为提高网络入侵检测系统(NIDS)在互联网流量和网络攻击数量增长下的性能,进行了在多核处理器上利用并行结构提高 NIDS 处理能力的研究。首先实现了 NIDS 在 TILERA-GX36 众核处理器上的数据并行(RTC)和任务并行(SPL)这两种并行机构方法,实验结果表明众核处理器上丰富的计算资源支持大量并行的 NIDS 实例,但同时也带来严重的资源竞争和冲突,系统并行化开销大大增加。为此,提出了一种基于共享的 RTC 方法,即 SRTC 方法,和已有方法相比,SRTC 方法解决了 RTC 模型内存占用线性增长的问题,同时避免了 SPL 模型中的线程间通信开销。以开源 NIDS 软件 Snort 为基础,在 TILERA-GX36 众核处理器上对 SRTC 方法进行了实现和验证,实验结果证明采用 SRTC 的并行系统获得了类似线性的加速比,当加载超过7000 条 NIDS 真实规则条目时,系统能够处理包长为 1K 字节的 10Gbps 的网络流量。

关键词 众核处理器, 网络入侵检测系统(NIDS), 并行结构

0 引 言

随着计算机网络的发展和广泛应用,网络入侵检测系统(network intrusion detection system, NIDS)在计算机网络安全领域的作用越来越重要[1]。利用在规则文件中描述的已知网络攻击特征,NIDS 通过对数据包进行规则匹配来检测互联网流量中是否包含攻击。由于网络攻击手段越来越复杂,攻击种类越来越多,NIDS 的规则库持续快速增长。以著名的开源入侵检测系统 Snort[2] 的官方规则集为例,最新发布的规则集中有超过 10000 条的规则条目。由于规则条目的快速增加与网络速率的提高,NIDS 性能问题日益突出,如何提升 NIDS 的处理能力,是面临的严峻挑战,在多核处理器上提高 NIDS 的性能则成为研究的热点。在 NIDS 中,数据包规则匹配阶段的时间开销达到 70% 以上[3],提升 NIDS 性能的主要方法是利用硬件或者软件手段,消除这个性能瓶颈。硬件的方法主要是利用现场可编程门阵列(field-programmable gate array, FPGA)、专用集成电路(application specific integrated circuits, ASIC)和三态内容寻址存储器(ternary content addressable memory, TCAM)等芯片来实现规则匹配[4,5]。由于硬件设备的灵活性差、价格昂贵、扩展性差,当前对高性能 NIDS 的研究主要采用软件方法,在多核处理器上利用并行结构提升 NIDS 的性能。这种方法的优点是灵活度高,相对采用硬件芯片的系统,开发和维护的成本也低很多。

NIDS 在多核处理器上,主要有数据并行(run-to-complete, RTC)和任务并行(software-pipeline, SPL)两种并行结构[6]。RTC 的优点是实现简单,SPL 的优点是程序局部性好。和单进程 NIDS 相比,已有研究中采用这两种模型的系统性能都有所提升。然而受限于通用多核处理器的计算资源,这些系统的加速比并不好,获得的最好性能为 3Gbps 左右[7],不能满足需求。TILERA、CAVIUM 以及 INTEL 等公司发布的众核处理器,比多核处理器拥有更丰富的计算资源,为进一步提高 NIDS 的性能提供了新的机会。在众核处理器上设计高性能 NIDS 的关键是最大程度利用处理器上的计算资源。基于

① 863 计划(2013AA013501)和国家自然科学基金(61133015)资助项目。

② 男,1982 年生,博士生;研究方向:高速数据包处理,网络安全;联系人,E-mail: bupt.jsea@gmail.com

(收稿日期:2014-04-14)

开源 NIDS 产品 Snort 和 Suricata[8]，本研究首先在众核处理器上实现了 RTC 和 SPL 这两种并行模型。实验结果表明，众核处理器上丰富的计算资源可以支持大量并行的 NIDS 实例，但同时也会带来严重的资源竞争和冲突，系统并行化开销大大增加。为此，本研究提出一种改进的并行方法，即共享的 RTC（shared-RTC，SRTC）方法，和已有方法相比，SRTC 解决了 RTC 模型中内存资源占用线性增长的问题，同时避免了 SPL 模型中的线程间通信开销。以 Snort 为基础，我们在 TILERA-GX36 众核处理器[9]上对 SRTC 进行实现和验证，实验结果证明，采用 SRTC 的 NIDS 系统可获得类似线性的加速比；当加载包含超过 7000 条表项的 NIDS 规则集合时，系统能够处理超过 10Gbps 的网络流量。

1 相关工作

1.1 TILERA-GX36 众核处理器

众核处理器的芯片上集成了几十到上百个处理器核心，相比于多核处理器，大大提高了处理能力。和硬件芯片相比，在众核处理器上编程灵活，用户开发和维护的成本显著降低。

本文采用的 TILERA-GX36 众核处理器的芯片上集成了 36 个同构的处理器核心。不同于基于总线互联的多核处理器，TILERA-GX36 芯片的 36 个处理器核心之间通过一个 6＊6 的二维片上网络互联。为了加速数据包的处理能力，在 TILERA-GX36 芯片上还集成了多核可编程智能包引擎（multicore programmable intelligent packet engine，mPIPE），为系统提供线速的数据包捕获、分类、负载平衡和缓存管理功能[9]。

1.2 NIDS 并行模型

NIDS 的并行模型分为 RTC 和 SPL 两种[6]。RTC 是数据并行模型，SPL 是任务并行模型。

在 RTC 模型中，NIDS 的包处理流程被复制并绑定到各个处理器单元。每个流程独立完成数据包捕获、解码、检测任务，整个并行系统同时处理多个数据包。在 RTC 模型中，由于每个线程都复制一份 NIDS 的数据结构，系统的内存占用随线程数量线性增长，导致 RTC 的性能和扩展性较差。在文献[10]中，英特尔的 RTC 模型在配置为 4 个处理器内核的服务器上性能只有 29Mbps。

在 SPL 模型中，数据包处理流程按照功能被划分成几个阶段，阶段之间通过队列来传递消息[11]。依据各个阶段在数据包处理流程中的耗时情况为其分配相应的计算资源。通过为耗时多的阶段分配更多的资源，运行更多的处理线程来消除性能瓶颈，获得系统性能提升。SPL 模型中每个线程负责数据包处理流程的一部分，和 RTC 模型相比，程序的访问局部性和系统的缓存命中率都得到提高。英特尔提出的 Supra-linear 模型利用 SPL 方法对 Snort 并行加速[12]。在配置 8 个处理器核心的服务器上，和单进程 Snort 相比，系统性能提高了 75%。Supra-linear 模型中，Snort 处理流程划分为两个阶段：数据包抓取和数据包处理。由于包处理阶段耗时远大于包抓取阶段，Supra-linear 模型分配一个处理器内核运行数据包抓取线程，其余的计算资源运行数据包处理线程。为避免共享数据造成的锁开销，Supra-linear 中所有数据包处理线程都复制了 Snort 中包处理阶段的相关数据结构，造成系统内存占用线性增长，导致系统的可扩展性差。基于 Supra-linear 模型，陈新民等人在并行的数据包处理线程间共享只读数据结构，减少了内存占用，系统的性能提高到 800Mbps[13]。在此基础上，Derek 等人提出的 Multi-Snort 模型通过改进并行的包处理线程间的负载均衡策略，在 8 个处理器核心的服务器上，获得了 3.1Gbps的处理性能[7]。为了提高 SPL 模型中的程序访问局部性，在开源 NIDS 软件 Suricata 中，对 NIDS 进行了更细粒度的划分和并行。如图 1 所示，NIDS 的数据包处理流程按照功能划分为 4 个流水级，为了消除系统的性能瓶颈，规则匹配模块（图 1 中的第 4 个流水级）中的线程数量是可配置的。

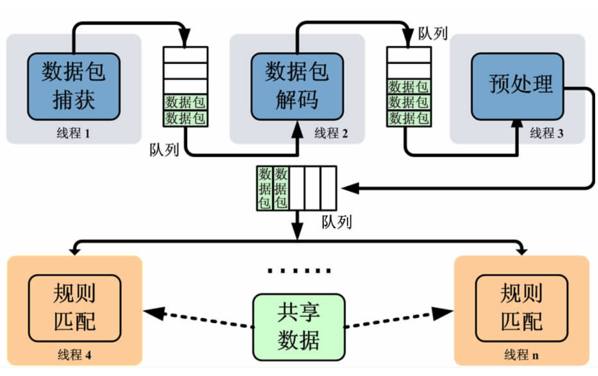

图 1 开源 NIDS 软件 Suricata 的流程图

2 SRTC 并行方法及其实现

在介绍 SRTC 并行方法的设计之前，首先在众核处理器上实现 RTC 和 SPL 方法，并对其性能进行

分析。我们使用实验室开发的 TCP 流量测试仪对并行系统进行性能测量。该测试仪表发送的 TCP 流量速率为 10Gbps,发送的数据包大小设置为 128 字节。并行 NIDS 系统加载 Snort 官方规则集,集合中包含 7571 条规则条目。

2.1 基于 RTC 的 NIDS 在众核处理器上的性能

RTC 方法的实现形式是复制 NIDS 的数据包处理流程,并绑定到处理器核心上。如图 2 所示,我们在众核处理器上直接运行多个 Snort 的实例对 RTC 模型进行实现和验证。

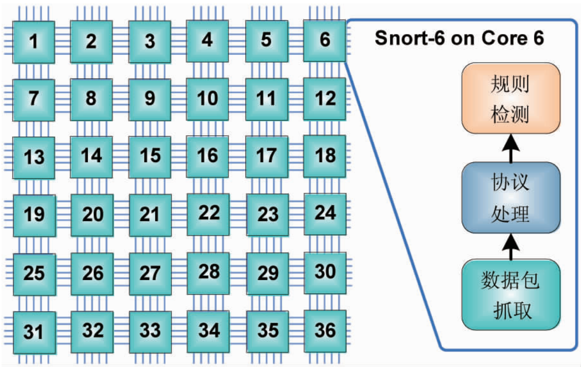

图 2 基于 Snort 的 RTC 模型在 GX36 处理器上的实现

采用 RTC 模型的并行 Snort 在众核处理器上的性能情况如图 3 所示。当运行 1 个 Snort 进程时,系统性能为 0.15Mpps(153.6Mbps),当运行 36 个进程时,并行系统的性能为 1.95Mpps(2.0Gbps)。从图中的加速比曲线可以看出,系统性能虽然随着并行的进程数递增,但是加速比并不是线性。当系统运行 6 个 Snort 进程时,加速比为 6;当进程数增长为 36 时,加速比为 13。

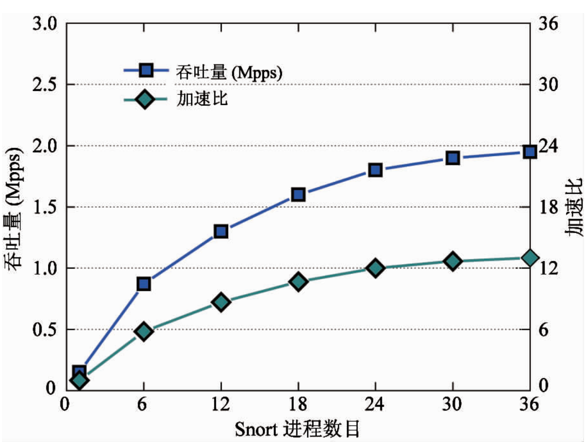

图 3 采用 RTC 并行方法的 Snort 在 GX36 众核处理器上的性能和加速比

为了进一步分析影响并行系统加速比的原因,我们利用 Linux 下的性能评测工具 Oprofile 对系统的缓存使用情况进行测量[14]。如图 4 所示,随着处理器上并行的 Snort 进程增多,系统的内存占用和二级缓存的失效率都随之增长。例如只运行一个 Snort 进程时,系统的内存占用约为 180Mbyte,二级缓存的失效率为 6.35%。当运行 36 个 Snort 进程时,系统的内存占用达到 6.48Gbyte,二级缓存的失效率也随之升高到 32% 左右。由于处理器的运算速度远大于内存的访存速度,大量的缓存失效会显著降低处理器的处理能力[15]。所以,RTC 模型中内存占用的线性增长是影响其性能的主要原因。

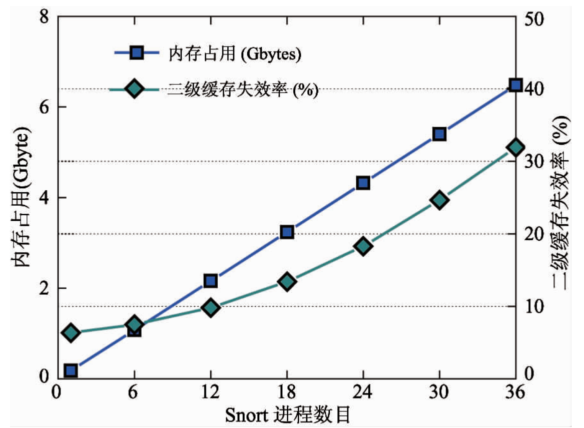

图 4 采用 RTC 并行方法的 Snort 在 GX36 上内存和二级缓存的使用情况

2.2 基于 SPL 的 NIDS 在众核处理器上的性能

我们在众核处理器上直接运行 Suricata 验证 SPL 模型的性能情况。如图 1 所示,Suricata 中,第 4 个流水级的计算资源是可配置的。在试验中,我们设置系统中并行的规则检测线程数目,为规则检测模块分配更多的计算资源从而获得系统性能的提升。

图 5 中记录了 Suricata 在众核处理器上的性能情况。从图中可以看出,在规则检测线程数较少时,系统的性能随着线程的增多而升高。例如,当系统中有一个规则检测线程时,处理速度是 0.13Mpps(133Mbps),当线程数增加到 4 个时,处理速度增长到 0.45Mpps(460Mbps)。但是当超过 4 个规则检测线程后,系统的处理速度不再升高(例如当 5 个规则检测线程时,系统的处理速度为 0.44Mpps),如果系统中并行线程数目继续增加,性能还会进一步下降(例如当 10 个规则检测线程时,系统的处理速度仅为 0.23Mpps)。由于规则检测模块是 NIDS 的

性能瓶颈,Suricata 在数据包捕获、解码和预处理阶段都只运行一个线程,剩余的计算资源都用来完成数据包检测任务。随着规则检测线程的增多,当规则检测阶段的处理能力不再成为系统的性能瓶颈后,受限于前面三个阶段的处理能力,更多的规则检测线程不能带来系统性能的继续提升。

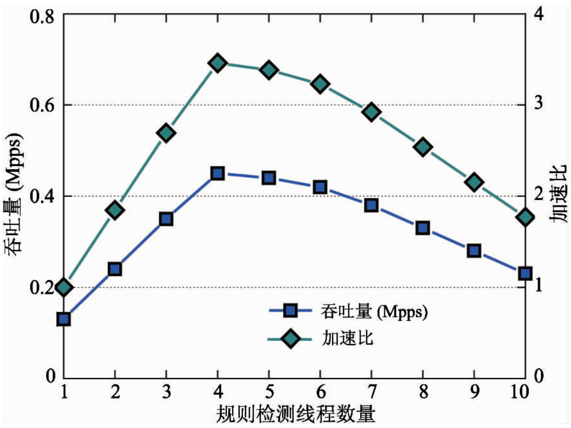

图 5 Suricata 在 GX36 众核处理器上的性能和加速比

为了进一步研究图 5 中,在性能曲线的后半段,当线程数继续增多系统性能反而快速下降的原因,我们对 Suricata 的内存占用和二级缓存的使用情况进行测量。从图 6 中可以看出,由于在线程之间共享数据结构,线程的增加对于 Suricata 的内存使用影响很小。系统的二级缓存丢失率在图中存在一个明显的分界线:在规则匹配的线程数小于等于 4 个时,由于内存占用较小,系统的二级缓存丢失率保持在一个比较低的水平上;当线程数超过 4 个后,二级缓存丢失率明显上升,在线程数为 10 个时,丢失率已经升高到了 30% 以上。结合图 5 中系统的性能变化情况,可以认为二级缓存的大量丢失造成了

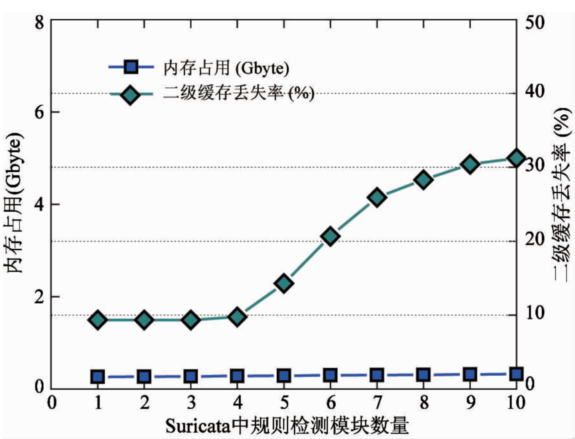

图 6 Suricata 在 GX36 上内存和二级缓存的使用情况

系统的性能下降,其根本原因是规则匹配线程的过度分配,加剧了线程间对共享数据的竞争和冲突,是导致系统性能下降的原因。所以,在 Suricata 采用的 SPL 模型中,如果为前三个流水级分配更多的线程,依然会很容易造成流水级之间的过度资源竞争,进一步造成系统的性能下降。

2.3 SRTC 并行方法

从图 3 看出,当 NIDS 的进程数目较少时,RTC 方法可以获得类似线性的加速比。从图 4 看出,在并行系统的进程数目较多时,由于系统内存占用的增多导致性能下降。如图 7 所示,和 RTC 相比,SRTC 方法的主要改进包括两个方面:一是利用线程的方式来优化 RTC,把 NIDS 的数据包处理流程以线程方式进行复制并绑定到处理器核心,线程的创建、维护、调度和内存占用的开销都远小于进程,使用线程的方式有利于提高并行系统的性能;二是由于所有的并行 NIDS 线程共享相同的地址空间,SRTC 模型中把系统中的只读数据结构以及共享冲突较小的数据结构从 NIDS 的处理流程中分离出来,设计为所有线程都可以访问的公共变量,从而解决 RTC 方法内存占用线性增加的问题。

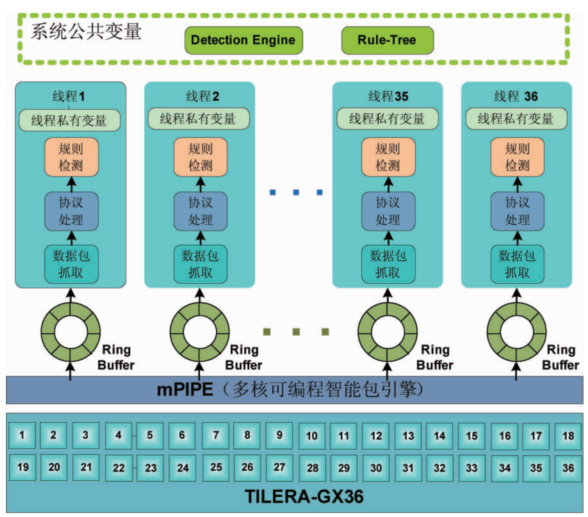

图 7 SRTC 方法及其在 GX36 众核处理器上的实现

图 8 给出了单进程 NIDS 的数据包处理流程和相关数据结构的关系。在抓取阶段,从链路上获得的数据包被保存在 raw-packet 结构中,供后续模块进行处理和检测。数据包解码阶段对数据包的 MAC、IP、TCP、UDP 以及应用层协议头进行解码,并将协议信息保存在 packet 结构中送入下一阶段。根据 packet 结构中保存的协议信息,预处理阶段首先在 flow-table 中查找数据包的流(TCP/UDP session）信息。如果没有找到，程序会创建一条新表项加入 flow-table 中；如果找到该流，则根据当前的数据包对流的状态做出更新。根据 flow-table 中的流状态信息，预处理阶段对 TCP 数据包进行重组。最后，重组的数据包被送入规则检测阶段，根据 detection-engine 结构对数据包的内容进行检测。在程序初始化时，NIDS 根据 rule-tree 结构中保存的规则条目生成 detection-engine。

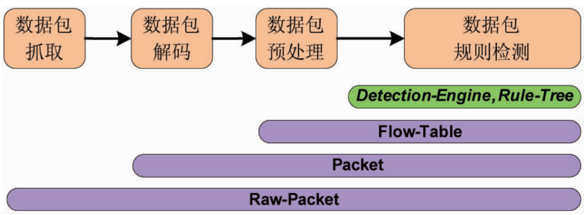

图 8　NIDS 中数据包处理流程及相关数据结构

如图 7 所示，SRTC 模型中的每个线程都运行完整的 NIDS 数据包处理流程。线程的私有变量只有本线程才能访问，而所有线程都可以访问系统的公共变量。为了确定变量是否可以共享，我们对图 8 中的数据结构进行分析。并行的 NIDS 线程独立地对网络流量进行捕获、解码、预处理以及规则检测，因此，每个线程需要创建私有的 packet 和 raw-packet 结构来完成数据包的处理。线程把抓取到的数据包保存在私有的 raw-packet 结构中，相关的协议处理信息则保存在私有的 packet 结构中。flow-table 中记录了系统正在处理的 TCP/UDP 流信息，在 NIDS 的预处理和规则匹配阶段都会对当前数据包所属的流信息进行读取和更新操作。当系统存在多个并行 NIDS 线程时，对 flow-table 结构的共享会造成线程间共享资源的冲突，因此也需要为每个线程创建私有的 flow-table 结构。在 NIDS 的初始化过程中根据加载的规则文件生成 rule-tree 结构，继而生成 detection-engine 结构。在数据包的处理流程中，这两个结构都是只读的。随着 NIDS 中规则条目的增多，rule-tree 和 detection-engine 的内存占用也越来越大[13]。如图 7 所示，我们在 SRTC 中把这两个数据结构设计成公共变量，可以显著降低并行系统的内存开销。

2.4　SRTC 模型在众核处理器上的实现和优化

基于开源 NIDS 软件 Snort，我们在众核处理器上对 SRTC 模型进行实现和优化。我们把 Snort 的数据包处理函数作为并行线程的入口函数，使用遵循 POSIX（可移植操作系统接口）标准的 pthread 线程库对 Snort 的包处理流程进行并行化。为了保证各个线程的正确运行，利用 GNU 编译器套装（GNU compiler collection，GCC）的扩展属性 _ thread 把 Snort 的变量私有化，即将 raw-packet、packet 和 flow-table 等结构变成线程私有变量。包括 rule-tree 和 detection-engine 结构在内，系统初始化阶段根据配置文件和规则文件生成的只读变量则是全局共享的。为了避免各个 Snort 线程在处理器之间迁移，设置线程的亲和性，为每个 Snort 线程绑定不同的处理器核心。

我们使用 TILERA-GX36 上的 mPIPE 收包引擎替换原始 Snort 中的 LibPcap 库。在 LibPcap 库的收包过程中，CPU 的多数时间都用来把网卡收到的数据包经过内核队列发送到用户空间。即从网卡缓冲区拷贝到系统内核缓冲区，再从内核缓冲区拷贝到用户空间这两个步骤，耗费了大量的 CPU 时间[16]。为此，我们使用 mPIPE 收包引擎来进行数据包的抓取。如图 7 所示，每个 Snort 线程把预先分配的 raw-packet 结构保存在自己的环状缓冲区中，mPIPE 直接将数据包写入缓冲区，并通知 Snort 进行处理。使用 mPIPE 避免了内存的两次拷贝，可以提高并行 NIDS 处理能力。

3　试验验证和性能分析

3.1　试验平台

我们的实验平台是一块安装在英特尔服务器上的 TILERA-GX36 PCIe 众核处理器板卡。板卡集成了 36 个频率为 1.2GHz 的处理器核心，每个核心上有 64kbyte 的一级缓存和 256kbyte 的二级缓存。板卡上安装了 8Gbyte 的 DDR3 内存和 4 个 10G 以太网端口。在软件方面，实验中使用的 Snort 版本是 2.8.4，程序加载包含 7571 条规则条目的 Snort 官方规则库。试验中，我们使用开源软件 Oprofile 对系统的缓存情况进行记录和统计。

我们使用实验室开发的 TCP 流量测试仪对并行系统进行测试，测试仪向板卡的 10GE 端口发送线速 TCP 流量，发送的数据包大小可配置。

3.2　性能结果和分析

图 9 和图 10 中给出了采用 SRTC 模型的并行 NIDS 的性能、内存占用和缓存使用情况。

从图 9 可以看出，采用 SRTC 模型的系统中，吞吐量随着并行线程数的增多而增长，和 RTC 和 SPL 模型相比，系统的处理能力和加速比都得到提升。

以 128 字节长度的数据包为例,当运行 1 个 Snort 线程时,系统的性能为 153.6Mbps,当运行 36 个线程时,并行系统的性能提高到 4.2Gbps,获得的加速比为 27。和 RTC 方法相比(最高吞吐量为 2.0Gbps,最好加速比为 13),SRTC 模型的最高吞吐量和最好加速比都提高了一倍;和 SPL 方法相比(最高性能为 460Mbps,最好加速比为 3.5),SRTC 模型的最高吞吐量提高了 8 倍,最好加速比提高了近 7 倍。在实验中,为了验证 SRTC 模型的性能情况,我们使用了多种长度的数据包进行了测试,如图 9 所示,在数据包长度为 1024 字节时,并行系统可以处理 10G 线速的以太网流量。

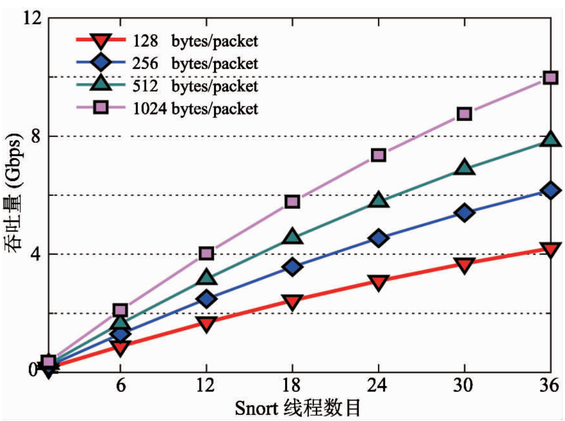

图 9　SRTC 模型在 GX36 众核处理器上的性能

如图 10 所示,由于在线程之间共享了大量只读的数据结构,系统的内存占用得到了显著降低。当系统中有 36 个线程时,内存占用为 1.26Gbyte,为 RTC 方法(36 个线程的内存占用为 6.48Gbyte)的 20%。系统内存的大幅度降低进一步优化了程序的局部性,系统的缓存失效率也得到了降低,当系统中有 36 个线程时,二级缓存失效率为 9.9%,为 RTC 方法(36 个线程时失效率为 32%)的 30%。同时,由于这些数据结构都是只读的,大量的线程对其进行同时读操作,也不会带来资源的竞争和冲突。由于目前对数据包的处理速度远未达到 TILERA-GX36 处理器的内存带宽,所以,对共享数据结构的读取不会带来额外的开销。

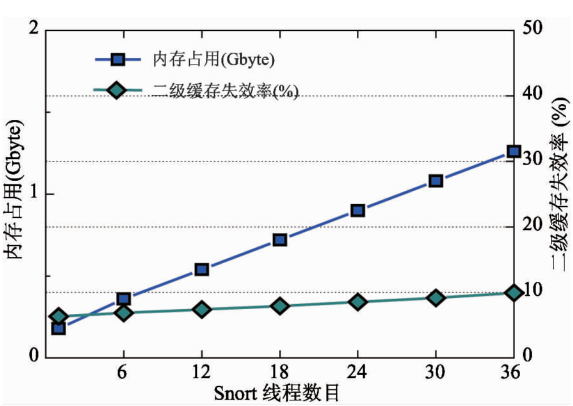

图 10　SRTC 模型在 GX36 众核处理器上的内存占用和缓存使用情况

4　结　论

为了充分利用众核处理器上丰富的计算资源,降低已有的 RTC 和 SPL 模型中的内存占用和资源竞争等并行开销,提高 NIDS 的处理能力,本文提出了一种改进的并行方法 SRTC(shared-RTC)。和已有方法相比,SRTC 解决了 RTC 模型内存资源占用线性增长的问题,同时避免了 SPL 模型中的线程通信开销。以开源 NIDS 软件 Snort 为基础,我们在 TILERA-GX36 众核处理器上对 SRTC 进行实现和验证。和 RTC 方法相比,采用 SRTC 模型后,系统性能和加速比提高了 1 倍;和 SPL 方法相比,系统的加速比提高了 7 倍,性能提高了 8 倍。采用 SRTC 方法后,并行系统获得了类似线性的加速比,当加载超过 7000 条 NIDS 真实规则条目时,系统能够处理包长为 1K 字节的 10Gbps 的网络流量。

参考文献

[1] Mukherjee B, Heberlein L T, Levitt K N. Network intrusion detection. *Network, IEEE*, 1994, 8(3): 26-41

[2] Snort. http://vrt-blog.snort.org/, 2014

[3] Sommer R, Paxson V, Weaver N. An architecture for exploiting multi - core processors to parallelize network intrusion prevention. *Concurrency and Computation: Practice and Experience*, 2009, 21(10): 1255-1279

[4] Lee J, Hwang S H, Park N, et al. A high performance NIDS using FPGA-based regular expression matching. In: Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, 2007. 1187-1191

[5] Kim S, Lee J. A system architecture for high-speed deep packet inspection in signature-based network intrusion prevention. *Journal of systems architecture*, 2007, 53(5): 310-320

[6] Verdú J, Nemirovsky M, Valero M. MultiLayer processing-an execution model for parallel stateful packet processing. In: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, USA, 2008. 79-88

[7] Schuff D L, Choe Y R, Pai V S. Conservative vs. opti-

mistic parallelization of stateful network intrusion detection. In: Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, USA, 2007. 138-139

[8] Suricata. http://www.suricata – ids.org, 2014

[9] TILE-Gx8036 product brief. http: //www.tilera.com/sites/default/files/productbriefs/TILE-Gx8036PB033-02web.pdf, 2014

[10] Intel Corporation. Supra-linear packet processing performance with intelmulti-core processors. Intel White Paper, 2006

[11] Suleman M A, Qureshi M K, Patt Y N. Feedback-directed pipeline parallelism. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, Vienna, Austria, 2010. 147-156

[12] Intel Corporation. Removing system bottlenecks in multi-threaded applications. Intel White Paper, 2008

[13] Chen X, Wu Y, Xu L, et al. Para-snort: a multi-thread snort on multi-core ia platform. In: Proceedings of The 21st LASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, USA, 2009. 137-146

[14] Profile. http://oprofile.sourceforge.net/download/, 2014

[15] Wulf W A, McKee S A. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer Architecture News*, 1995, 23(1): 20-24

[16] 王佰玲, 方滨兴, 云晓春. 零拷贝报文捕获平台的研究与实现. 计算机学报, 2005, 28(1): 46-52

High-performance parallel NIDS on ManyCore processors

Jiang Haiyang[*,**], Xie Gaogang[**]

([*] University of Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract

In order to improve the performance of a network intrusion detection systems (NIDS), a study on increasement of the processing capacity of a NIDS on multi-core processors by using the software based on parallel models was conducted. The two parallel designs of run-to-complete (RTC) and software pipeline (SPL) were implemented on the ManyCore processor of TILERA-GX36. For experiment demonstrated that this ManyCore processor's rich computation resources supported many parallel NIDS examples, but it brought serious resource competion and conflicts, leading to the system's great increasement of parallel overhead. Thus a Shared-RTC (SRTC) model for parallel NIDS was proposed. Compared with the models of RTC and SPL, the parallel overhead of the proposed SRTC model decreased largely and hence it took the full advantage of the ManyCore processor for the security task. Furthermore, the proposed design leverages particular features of the processor to break the bottlenecks. The proposed design was integrated into the open source NIDS Snort for performance evaluation. The prototype exhibits almost linear speedup and experimental results show that the parallel system can handle up to the 10 Gbps traffic with the packet size of 1kbytes, against a ruleset contains about 7K signatures.

Key words: ManyCore processor, network intrusion detection system (NIDS), parallel design