基于运行时验证的边缘服务器 DoS 攻击检测方法

于斌，张南，陆旭，段振华，田聪
（西安电子科技大学计算机科学与技术学院，陕西 西安 710071）

摘 要：针对边缘计算系统中边缘服务器面临的拒绝服务（DoS）攻击问题，提出了一种基于并行运行时验证的 DoS 攻击检测方法。首先，使用命题投影时序逻辑（PPTL）公式形式化描述边缘服务器预期行为和 DoS 攻击特征；进而，针对待验证 PPTL 公式，采用并行运行时验证框架，充分利用边缘服务器的计算与存储资源，对程序运行状态进行异常检测和误用检测。利用所提方法，对一个实际的基于边缘计算的点对点（P2P）网络智能停车系统进行模拟 DoS 攻击和攻击检测。对比实验表明，所提方法能够有效检测出边缘服务器异常行为和所受 DoS 攻击类型。

关键词：边缘计算；边缘服务器；命题投影时序逻辑；拒绝服务攻击；运行时验证
中图分类号：TN92
文献标识码：A
DOI: 10.11959/j.issn.1000−436x.2021169

Runtime verification approach for DoS attack detection in edge servers

YU Bin, ZHANG Nan, LU Xu, DUAN Zhenhua, TIAN Cong
School of Computer Science and Technology, Xidian University, Xi’an 710071, China

Abstract: Aiming at the DoS (denial of service) attacks against edge servers in an edge computing system, a parallel runtime verification approach for DoS attack detection was proposed. First, PPTL (propositional projection temporal logic) formulas were utilized to formally describe expected behaviors of an edge server and DoS attack characteristics. Then, for the PPTL formulas to be verified, a parallel runtime verification framework was adopted to make use of the computing and storage resources of an edge server to conduct anomaly detection and misuse detection. The proposed attack detection approach was performed for an actual P2P (peer-to-peer) network based on smart parking system using edge computing which was supposed to suffer from a DoS attack. Experiments show that the proposed method can accurately and efficiently identify abnormal behaviors of edge servers and types of DoS attacks.

Keywords: edge computing, edge server, propositional projection temporal logic, DoS attack, runtime verification

1 引言

边缘计算系统利用终端设备及终端附近的边缘服务器的计算和存储资源，提前对数据进行部分处理，减少要发送到云数据中心的数据量，从而减少网络带宽使用量，并使系统能够在较低数据时延的情况下立即响应数据[1]。边缘计算系统总体框架如图 1 所示。在边缘计算系统中，边缘服务器处于整个系统的中间层，起到连接云服务器和终端设备的作用，并承担云服务器的部分计算任务。随着服

收稿日期：2021−01−21；**修回日期**：2021−03−25
通信作者：张南，nanzhang@xidian.edu.cn
基金项目：国家重点研发计划基金资助项目（No.2018AAA0103202）；国家自然科学基金资助项目（No.61732013, No.61806158）；中央高校基本科研业务费专项资金资助项目（No.XJS210305）；陕西省自然科学基础研究计划基金资助项目（No.2021JQ-208）
Foundation Items: The National Key Research and Development Program of China (No.2018AAA0103202), The National Natural Science Foundation of China (No.61732013, No.61806158), The Fundamental Research Funds for the Central Universities (No.XJS210305), The Natural Science Basic Research Program of Shaanxi (No.2021JQ-208)

务器性能的不断提升，越来越多的数据处理任务依托边缘服务器来完成。

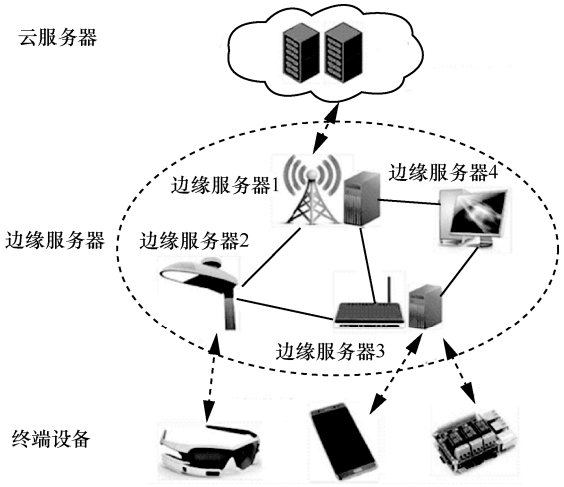


图 1 边缘计算系统总体框架

然而，边缘服务器的特点使其面临新的挑战[2-5]。首先，边缘服务器常位于网络边缘，面临更复杂的网络环境。此外，边缘服务器往往是地理分布式的，难以做到集中的设备安全保护，且每个边缘服务器是不同的，此差异性给攻击者带来了更多的可选择性。更严重的是，由于计算能力和存储空间的限制，大多数边缘服务器只关注业务逻辑的实现，而对可能受到的攻击缺少防范和检测。由以上分析可知，虽然边缘服务器在边缘计算中发挥重要作用，但其易攻击的特征使其面临严峻的攻击威胁。

由于边缘服务器的计算和存储资源受限，极易被耗尽，因此，在边缘计算系统面临的多种攻击中，针对边缘服务器的拒绝服务（DoS, denial of service）攻击很广泛[6-10]。在 DoS 攻击中，恶意节点通过无限制地访问指定的被攻击边缘服务器，向其发送大量的无效数据包，耗尽正在运行的服务器资源，阻碍边缘服务器的正常运行。由于在边缘计算架构中，边缘服务器用于实现数据的部分处理以及数据与云服务器的传输，一旦边缘服务器的通信链路堵塞或资源被耗尽，将直接造成预期的正常业务无法完成。

作为保障网络安全的一种重要手段，入侵检测能够实时监测网络活动，判断设备状态，及时发现 DoS 中的受攻击设备，主要包括异常检测和误用检测 2 种方法。其中，异常检测方法基于被监控对象的正常行为，其假设入侵是与正常行为必然不同的异常，因此，任何明显偏离正常的行为都被认为是侵入性的；误用检测对正在监视的事件进行匹配，符合已知攻击特征的操作均被视为入侵行为。在现有针对边缘计算系统的入侵检测方法中，大部分方法利用模式识别或机器学习手段[11-18]，虽然能够取得较好的检测结果，但需提前训练模型，在基于训练好的模型进行实时检测时需占用较大内存。与之对应，以模型检测和运行时验证为代表的形式化验证方法利用时序逻辑性质，对网络入侵的行为特征进行识别，不需要提前训练模型，但由于需要构造自动机等理论模型，检测效率往往不高，虽然在云计算等传统网络中得到应用[19-24]，但在边缘计算系统中的研究仍处于初步阶段。

鉴于边缘计算系统中边缘服务器的重要性和易攻击性，以及面向边缘服务器 DoS 攻击的广泛性和威胁的严重性，如何构建一个高效准确的入侵检测方法，是一个亟待解决的重要问题。作为轻量级形式化验证方法，运行时验证能够监视和分析真实系统的执行情况，以检查系统的运行轨迹是否满足形式化描述的属性，已逐渐被用于网络入侵检测中[19-20]。在边缘计算系统中，由于边缘服务器需要保持运行状态，不断响应终端设备的服务请求并完成期望的逻辑功能，因此，对其进行运行时验证是必要和可行的。

考虑到边缘服务器具有较强的计算能力，本文采用运行时验证方法对 DoS 攻击进行检测。为了充分利用边缘服务器计算资源以提高验证效率，解决目前运行时验证方法中线性时序逻辑[25]和计算树逻辑[26]表达能力不足的问题，本文结合误用检测和异常检测，提出基于命题投影时序逻辑（PPTL, propositional projection temporal logic）[27-29]的并行运行时验证方法。首先，基于统一建模语言（UML, unified modeling language）顺序图，使用 PPTL 公式描述系统正常状态下的行为特征；然后，使用 PPTL 公式描述 DoS 攻击中针对 TCP/IP 协议族最常见的 Smurf 攻击、SYN Flood（synchronize flood）攻击以及 Land 攻击的行为特征；进而，将上述两类 PPTL 公式作为待验证性质，在边缘计算系统运行过程中，在边缘服务器上部署 PPTL 公式的运行时验证方法，基于搜集到的信息，利用边缘服务器的多个线程，共同完成 DoS 攻击的入侵检测。在此过程中，若发现第一类性质不满足，则表明边缘服务器无法完成正常的请求响应；若发现第二类性质中某性质满足，则表明检测到某一特定类别的 DoS 攻

击。最后，为了检验上述方法的有效性，本文进行了性能对比实验，并对一个实际的基于边缘计算的点对点（P2P, peer-to-peer）网络智能停车系统进行针对边缘服务器的模拟 DoS 攻击和攻击检测，实验结果表明，所提方法能够有效检测出边缘服务器的异常行为和所受 DoS 攻击类型。

本文贡献主要包括以下三方面。

1) 采用 PPTL 公式描述基于 UML 顺序图的边缘服务器正常行为特征和不同 DoS 攻击下边缘服务器的异常行为特征，其中，PPTL 具有完全正则的表达能力，可直接用于描述边缘服务器上不同模块调用的区间相关与周期重复的性质。

2) 基于 PPTL 性质描述，采用运行时验证方法对边缘服务器进行实时监控，实现异常检测和误用检测，在此过程中，基于并行运行时验证框架，充分利用边缘服务器的空闲计算与存储资源，提高验证效率，及时发现攻击。

3) 将所提运行时验证方法部署在实际智能停车系统，对其进行边缘服务器监控，实验结果表明，边缘计算系统中的 DoS 攻击能够被有效识别。

2　相关理论

2.1　命题投影时序逻辑

PPTL 公式 P 归纳定义为

$$P ::= p \mid \mathrm{O}P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, \cdots, P_m)\ \mathrm{prj}\ P \mid P^+$$

其中，Prop 为一个可数的原子命题集合，$p \in \mathrm{Prop}$ 表示原子命题，$P_1,\cdots,P_m$ 和 P 是 PPTL 公式。PPTL 的语义已在文献[27-29]中明确给出。PPTL 派生的时态公式为

$$P_1;P_2 \triangleq (P_1, P_2)\ \mathrm{prj}\ \varepsilon \qquad P \triangleq \mathrm{true};P$$
$$\Box P \triangleq \neg\Diamond\neg P \qquad P^* \triangleq P^+ \vee \varepsilon$$
$$\mathrm{fin}(P) \triangleq \Box(\varepsilon \rightarrow P) \qquad \mathrm{len}(n) \triangleq O^n\varepsilon$$

任何 PPTL 公式都能够转换为对应的范式[29]，基于范式，能够构造出其对应的带标签的范式图（LNFG, labeled normal form graph）。LNFG 中的有穷路径 $\pi=<n_0, p_0, \cdots, \varepsilon>$是从初始节点到 ε 节点的节点和边的交替序列，其一定是可接收的。无穷路径 $\pi=<n_0, p_0, \cdots, (n_i, p_i, \cdots, n_j, p_j)^{\omega}>$是节点和边的无穷交替序列，若路径上无限出现的节点的集合（用 $\mathrm{Inf}(\pi)$表示）不存在被所有节点同时拥有的标签，则该路径是可接收的。

图 2 展示了 PPTL 公式$\Diamond(p\wedge\Box(\neg q))$对应的 LNFG，图中每个节点代表一个 PPTL 公式；每条边为一个状态公式；额外引入的命题 l_k 用于识别一条路径是否为可接收的。例如，有穷路径 π_1=<1, $p\wedge\neg q$, 3, $\neg q$, 4>为可接收路径；对于无穷路径 π_2=<1, true, (2, true) $^{\omega}$ >，由于 $\mathrm{Inf}(\pi_2)=\{2\} \subseteq L_1=\{2\}$，意味着无穷循环的节点拥有相同的标记 l_1，因此这条路径是不可接收的。

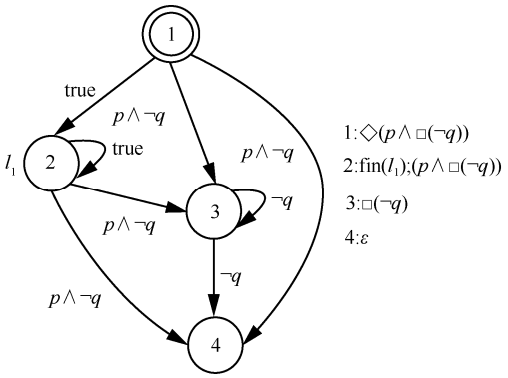
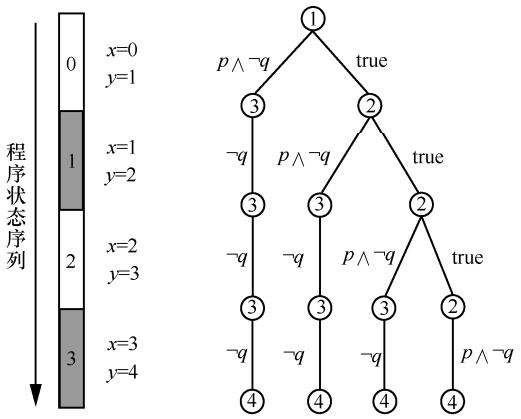


图 2　PPTL 公式$\Diamond(p\wedge\Box(\neg q))$对应的 LNFG

2.2　传统运行时验证方法

对于待验证程序 M，若采用传统的运行时验证方法验证其是否满足期望的 PPTL 性质 P，需要首先对程序 M 进行插桩，并构建$\neg P$ 对应的 LNFG G；然后，在插桩后的程序 M 执行过程中，每个状态上出现在 P 中的变量的值被记录下来，基于这些变量的值，可以得到 G 中对应边上的状态公式的真值；进而，探索 G 中的可接收路径，若能够在 G 中发现一条可接收路径，则表明程序的当前执行路径不满足性质 P；若不存在可接收路径，则表明程序的当前执行路径满足性质 P。

图 3 中的例子展示了传统运行时验证方法面临的挑战。图 3(a)展示了一个程序执行产生的状态序列，共包含 4 个状态。假设待验证的性质为 $\Box(p\rightarrow\Diamond q)$，其中原子命题 p 和 q 分别代表 x<=3 和 y>5。性质$\Box(p\rightarrow\Diamond q)$的非，即$\Diamond(p\wedge\Box(\neg q))$的 LNFG G 如图 2 所示。当程序执行时，G 中的路径探索结果如图 3(b)所示，图 3 中的节点编号与图 2 中的节点编号保持一致。由于在 G 中存在多条从节点 1 和节点 2 出发的边，在图 3(b)的前 3 步中，越来越多的路径产生。由于性质对应的 LNFG 中路径的探索过程相当于程序解释执行的过程，其速度原本就远小于程序编译后生成状态的速度，再加上 LNFG 中不断产生分支，因此在传统的运行时验证方法中，串行的验证方法限制了验证效率的提高。



(a) 程序执行产生的状态序列　　(b) G 中的路径探索结果

图 3　传统运行时验证方法面临的挑战

3　DoS 攻击检测方法

为了有效检测针对边缘服务器的 DoS 攻击，本文提出基于运行时验证的入侵检测方法，该方法可部署在需要监控的边缘服务器上，在边缘服务器提供服务的同时，实现针对边缘服务器的异常检测和误用检测。在检测过程中，为了充分利用边缘服务器的计算和存储资源，本文提出了基于多线程的并行运行时验证框架，以提高入侵检测效率，及时发现攻击的存在。攻击检测示意如图 4 所示。

在性质描述方面，本文采用与完全正则式表达能力相同的 PPTL 公式，其能够直接表达区间相关和周期重复的性质。为了提供异常检测所需性质，针对 UML 顺序图中表达的模块之间函数调用关系，使用 PPTL 公式描述正常情况下不同函数的执行序列；为了提供误用检测所需性质，针对不同 DoS 攻击的行为特征进行分析，并使用 PPTL 公式形式化描述此行为特征。

在程序执行和验证方面，首先针对上述待验证 PPTL 性质中涉及的程序变量与函数，利用底层虚拟机（LLVM, low level virtual machine）平台完成对被监控程序的插桩，进而编译成 LLVM 统一的中间代码（IR, intermediate representation）。在边缘服务器程序运行过程中，状态序列被分为多个子片段，并交由边缘服务器中创建的不同线程，以实现验证的并行化。在此过程中，为了实现多个线程的高效合作，需建立验证任务分配和验证结果合并的调度机制。

本节将对性质描述与程序执行和验证方法进行具体描述。

3.1　基于 PPTL 公式的性质描述

3.1.1　基于 UML 顺序图的 PPTL 性质描述

作为详细设计阶段的一种重要描述方法，UML 顺序图用于描述函数调用的时间顺序和消息传递，能够表达函数间的动态交互关系。本文在 UML2.0 中增加了包括选择（alt）和循环（loop）在内的交互片段。

针对边缘服务器的 DoS 攻击通过耗尽正在运行的边缘服务器的计算、网络和存储等关键资源，使其性能不断下降，直到不能提供正常服务，乃至系统崩溃，此时，UML 顺序图表达的系统正常行为将不能符合预期地被执行。为此，本文将重点关注边缘计算系统中边缘服务器上不同函数是否能够符合预期地被调用，在此过程中，将忽略 UML 顺序图上有关变量的数据流传递，只考虑模块间的函数调用以及模块内的函数调用。为了将系统预期满足的性质用时序逻辑语言形式化表达，本文利用

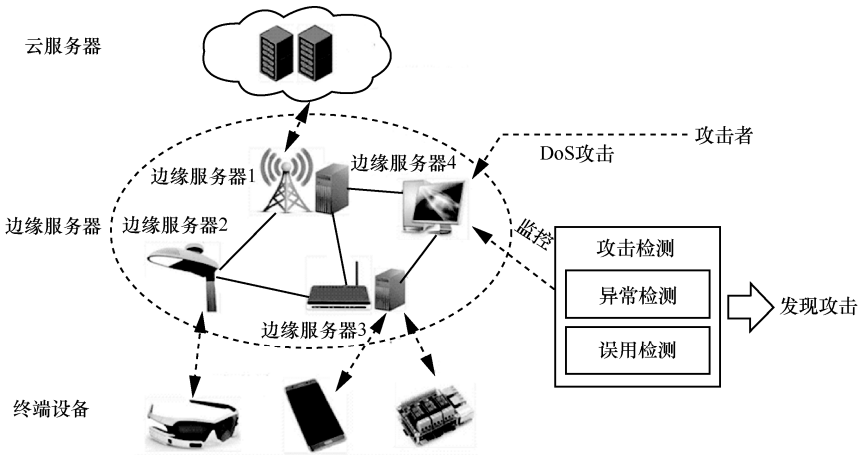


图 4　攻击检测示意

PPTL 公式描述UML 顺序图中表达的函数间调用关系。

将 UML 顺序图形式化的工作已经广泛开展，例如，文献[30]使用确定事件有限自动机描述 UML 顺序图，文献[31]使用线性时序逻辑（LTL, linear-time temporal logic）公式形式化表达顺序图。相较于 LTL 公式，PPTL 公式能够表达区间相关与周期重复性质，因此，本节将重点介绍采用 PPTL 公式描述 UML 顺序图中的循环片段。UML 顺序图如图 5 所示。

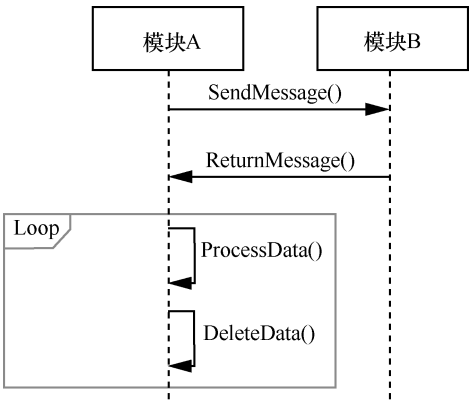


图 5　UML 顺序图

图 5 展示了 2 个模块中函数的调用关系。模块 A 调用 SendMessage 函数将相关消息发送给模块 B，模块 B 通过 ReturnMessage 函数返回消息后，模块 A 循环调用 ProcessData 和 DeleteData 函数来处理和删除冗余数据。基于此 UML 顺序图，可用如下 PPTL 公式形式化描述模块 A 中的函数调用序列。

$$\Diamond(\text{SendMessage};\ (\text{ProcessData};\ \text{DeleteData})^{+})$$

其中，原子命题 SendMessage、ProcessData 和 DeleteData 表示对应函数被调用，子公式 $(\text{ProcessData};\ \text{DeleteData})^{+}$ 表示 ProcessData 和 DeleteData 被循环调用。上述整体 PPTL 公式的含义为存在某个状态，从该状态开始，SendMessage 函数被调用，进而，ProcessData 和 DeleteData 函数被循环调用。若模块 A 被攻击而无法提供服务，将导致系统的执行路径违反上述公式。

3.1.2　基于 DoS 攻击特征的 PPTL 性质描述

由于每种 DoS 攻击均具备不同特征，因此可以对不同的 DoS 攻击进行原理分析，得出当攻击发生时，被攻击节点发生的动作序列，并在此基础上建立不同攻击对应的 PPTL 时序逻辑公式。本节将对最典型的 3 种针对 TCP/IP 协议族攻击的 DoS 攻击行为，即 Smurf、SYN Flood 和 Land 进行形式化描述。

1) Smurf 攻击

网际控制报文协议（ICMP, Internet control message protocol）为了判断线路能否正常通信，首先向目标主机发送请求，目标主机在接收到该请求后，自动回复应答给源主机。

入侵攻击者可利用 ICMP 的工作原理制造针对边缘计算系统的 Smurf 攻击，其将控制的设备接入边缘计算网络中，并将被攻击边缘服务器的主机地址作为请求回应的源地址，进而利用伪造的源地址向边缘计算网络中连续发送请求命令。接收到请求回应数据包后，被广播的网络主机均会发送回复应答，造成被攻击的边缘服务器的网络拥塞甚至系统崩溃。Smurf 攻击示意如图 6 所示。

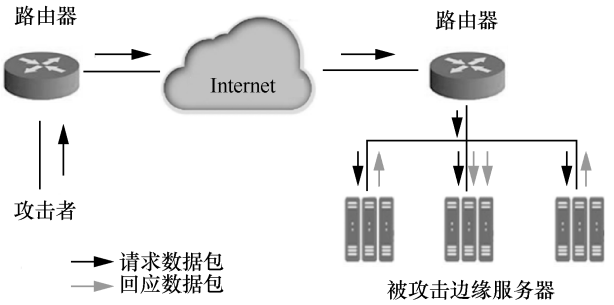


图 6　Smurf 攻击示意

Smurf 攻击的行为特征可总结如下。

边缘服务器没有向该子网的广播地址发送数据，但收到的数据均来自子网主机，可用如下 PPTL 公式形式化描述以上行为特征。

$$\Diamond(\neg\text{send} \rightarrow (\Diamond\text{receive})^{+})$$

其中，原子命题 send 表示被攻击的边缘服务器发送数据，receive 表示被攻击的边缘服务器接收响应数据。子公式$(\Diamond\text{receive})^{+}$表示 receive 行为不断发生，即不断接收到其他节点返回的数据包。上述整体 PPTL 公式的含义为存在某个状态，从该状态开始，该设备虽然没有发送数据包，但不断接收到数据包。

2) SYN Flood 攻击

基于 TCP/IP 互联网服务中，TCP 协议采用三次握手方式建立可靠连接，连接过程如图 7(a)所示。入侵攻击者利用上述三次握手工作原理，发起 SYN Flood 攻击。具体来说，攻击者伪装成边缘计算系

统中的多个终端或边缘服务器，在经过第一次握手和第二次握手后，故意不向边缘服务器发送第三次握手中所需的 ACK 信息，使边缘服务器不得不多次重新发送 SYN-ACK 应答数据包；同时，边缘服务器需维护大量连接队列，造成大量消耗资源的现象。边缘服务器一旦资源耗尽，就会出现速度极慢、无法连接边缘服务器等情况。除了在第三次握手中故意不向被攻击设备返回 ACK 信息，攻击者还可以将 SYN 数据包的源地址直接伪造成不存在的地址，向目标边缘服务器发起攻击，也可以达到同样的攻击效果。SYN Flood 攻击示意如图 7(b)所示。

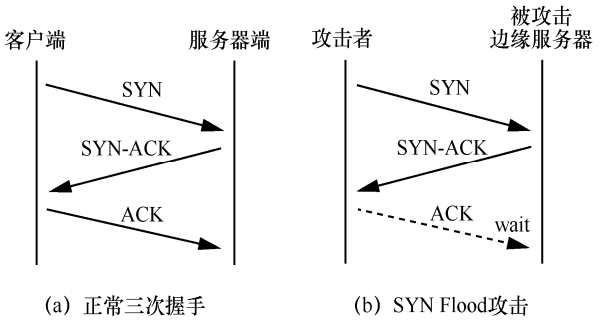


图 7 SYN Flood 攻击示意

SYN Flood 攻击的行为特征可总结如下。

边缘服务器收到某个主机发来的 SYN 数据包，并且向该主机返回一个 SYN-ACK 数据包后，一直未接收到对方再次返回的 ACK 数据包，可用如下 PPTL 公式形式化描述以上行为特征。

◇((receive.SYN; send.SYNACK)→□(¬receive.ACK))

其中，原子命题 receive.SYN、send.SYNACK 和 receive.ACK 分别表示被攻击的边缘服务器接收到 SYN 数据包、发送 SYN-ACK 回应数据包和接收到 ACK 数据包。公式使用(receive.SYN; send.SYNACK)表示接收 SYN 数据包和发送 SYN-ACK 数据包的行为先后发生。上述整体 PPTL 公式的含义为存在某个状态，从该状态开始，边缘服务器接收 SYN 数据包并进而发送 SYN-ACK 数据包，但是在之后的所有状态，都没有接收到 ACK 数据包。

3) Land 攻击

与 SYN Flood 攻击类似，Land 攻击同样利用 TCP 的连接建立过程。与正常报文不同的是，在向被攻击的边缘服务器发送的 SYN 报文中，源地址和目的地址均为被攻击边缘服务器的 IP 地址。

此操作将导致被攻击的边缘服务器首先向本机发送 SYN-ACK 数据包，然后返回 ACK 消息并创建一个空连接。值得注意的是，这些空连接将一直保留至超时。大规模的 Land 攻击将导致边缘服务器服务不可用。Land 攻击示意如图 8 所示。

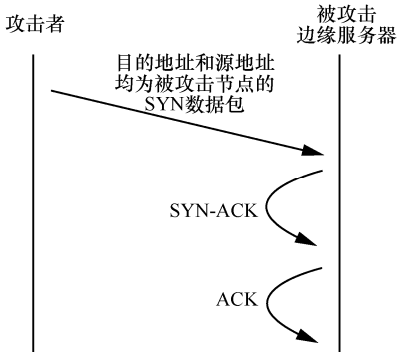


图 8 Land 攻击示意

Land 攻击的行为特征可总结如下。

被攻击边缘服务器收到 SYN 数据包后，不断试图与自己建立连接，即不断发送 SYN-ACK 数据包和 ACK 数据包，可用如下 PPTL 公式形式化描述以上行为特征。

◇(receive.SYN →◇(send.SYNACK; send.ACK)$^{+}$)

其中，原子命题 receive.SYN、send.SYNACK 和 send.ACK 分别表示被攻击的边缘服务器接收到 SYN 数据包、发送 SYN-ACK 数据包和发送 ACK 数据包。子公式 (send.SYNACK; send.ACK)$^{+}$表示发送 SYN-ACK 数据包和发送 ACK 的行为不断发生。上述整体 PPTL 公式的含义为存在某个状态，从该状态开始，边缘服务器接收到 SYN 数据包，之后，不断发送 SYN-ACK 数据包和 ACK 数据包。

3.2 入侵检测方法

针对边缘计算系统中边缘服务器 DoS 攻击入侵检测的并行运行时验证方法基本框架如图 9 所示，包括执行模块和验证模块 2 个同时运行的模块。

执行模块首先基于 PPTL 公式表达的时序逻辑性质，对边缘服务器上待验证的程序进行插桩，并基于 LLVM 平台将其编译为 IR 程序；然后在程序运行时，不断产生的状态序列被分为多个子片段。验证模块中，调度线程将不同序列片段的验证任务分发给同时运行的不同线程，并将不同片段被验证后的结果进行合并，得出最终验证结果。

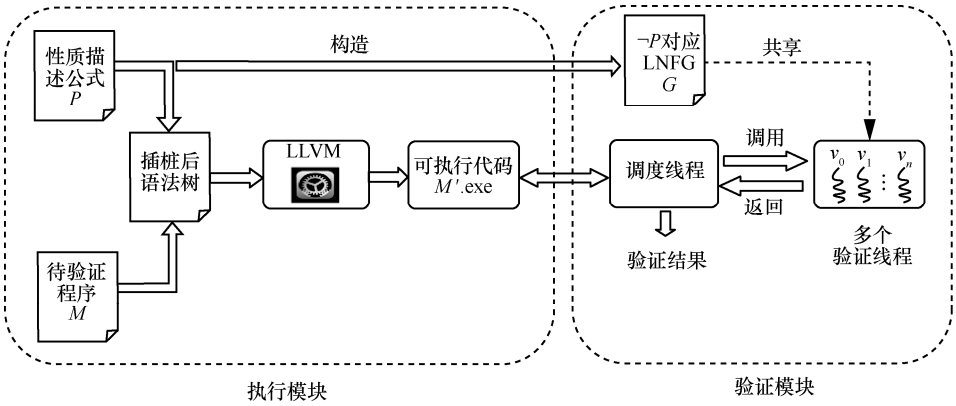


图 9　并行运行时验证方法基本框架

3.2.1　基于 PPTL 性质的代码插桩

边缘计算系统通过用 C、C++、Java 等多种语言实现，而这些语言均能通过 LLVM 平台转成统一的 IR，进而由 LLVM 完成中间代码的优化以及面向不同平台的最终可执行代码的生成。LLVM 框架如图 10 所示。

为了保证入侵检测方法的通用性，本文将基于 LLVM 平台对程序语法树进行插桩。从 3.1 节可看出，形式化描述后 PPTL 公式重点关注程序变量和函数的调用情况。针对程序变量，由于其取值只有在其作为赋值语句左值时发生改变，因此，只有当性质 P 中出现的程序变量作为赋值语句的左值时，与该变量有关的原子命题作为条件的条件语句会被插入此状态之后。针对函数调用，只有当相关函数被调用时，才会将相应原子命题赋值为 1，并记录此时的程序状态下标。需要指出的是，此算法不会保存每个状态上原子命题的真值，而是在原子命题的真值可能发生改变时，将该状态的下标记录下来。由于在大多数程序中，原子命题的真值在很多状态下是保持不变的，故该算法能够在很大程度上节省存储空间。

3.2.2　并行运行时验证方法

在被监测的边缘服务器中，假设有 n+2 个线程可用，其中，线程 e 用于执行程序，线程 s 用于调度验证任务和合并验证结果，线程 v_i（1≤i≤n）用于验证不同的序列片段。在线程 e 运行程序过程中，状态序列首先被线程 s 分为若干片段，进而线程 s 将这些片段的信息发送给线程 v_i（1≤i≤n）完成相应验证任务；最后，不同片段的验证结果被返回给线程 s，由其完成验证结果的汇总，并得到最终验证结果。在此过程中，若待验证的序列片段较多，而提供验证功能的线程较少，线程 s 与线程 v_i（1≤i≤n）将发生多次信息交互。该并行运行时验证方法的步骤可概括如下。

1) 使用 PPTL 公式 P 表达需检测的性质，将性质取非后，利用 MSV 工具集[29]中的 P2G 工具构造对应的 LNFG G，该 LNFG 的信息将被验证线程 v_i（1≤i≤n）共享。

2) 线程 e 执行编译后程序 M'.exe，在此过程中，调度线程 s 和验证线程 v_i（1≤i≤n）并行运行。

3) 当新生成的 m 个状态能够组成一个新的待验证序列片段 sg_k 时，序偶(k; i)被调度线程 s 加入集

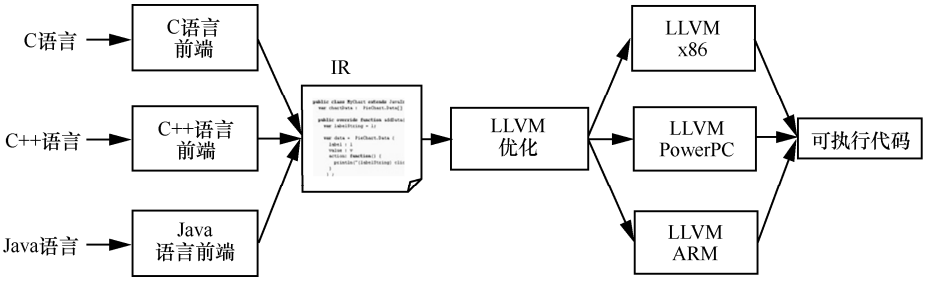


图 10　LLVM 框架

合 L 中，其中 i=min(I)，集合 I 存储目前没有执行验证任务的所有线程（初始值 I={1,2,⋯, n}）；然后，调度线程 s 将片段 sg_k 的信息发送给验证线程 v_i；此时，从集合 I 中删除 i；k=k+1。

4) 与线程 s 同时工作的验证线程 v_i（1≤i≤n）中，一旦其被调用来验证序列片段 sg_k，将基于 sg_k 中的状态信息，在¬P 对应的 LNFG G 上探索可扩展路径，并将可扩展路径集合 A_k={(in, en) | (in, en) 表示为至少存在一条从节点 in 到 en 的可扩展路径}；进而，线程 v_i 将(i, A_k)返回给调度线程 s。

5) 当线程 v_i 将序列片段 sg_k 的验证结果(k, A_k)返回给调度线程 s 后，(k, A_k)被加入 map 容器中；调度线程 s 通过 L=L−{(k;i)}操作，解除 sg_k 与 v_i 的对应关系；由于此时线程 v_i 再次空闲，调度线程将 i 加入集合 I 中；将 map 容器中的多个片段验证结果合并后，如果能找到一条对应 LNFG 中有穷路径，则一条反例路径被发现，即程序的当前执行路径不满足性质；若基于已有结果，能够断定 LNFG 中不存在从初始节点出发到达 ε 节点的路径，则表明程序当前执行路径满足性质；否则，调度线程 s 将继续等待更多序列片段的验证结果。

与传统验证方法中顺序寻找反例路径不同，在并行验证方法中，不同的状态序列片段是同时验证的，对于大部分状态序列的片段来说，当它们开始验证时，尚未得到之前片段的验证结果。上述步骤 4)对这些片段开展验证时，无法获知应从哪些节点开始探索可扩展路径。因此，第一个片段的初始节点的所有可达的非终止节点都应作为后续片段的初始节点。验证完成一个状态序列片段后，序偶(in,en)表示 LNFG 中至少存在一条从节点 in 到 en 的可扩展路径。一个序列片段被验证后，这些相邻片段上的序偶就会被连接以得到 LNFG 中更长的可扩展路径。

图 11 展示了一个程序状态序列满足性质的情况。假设在验证过程中，程序状态序列被分为 3 段，每段包含 2 个状态。若待验证的性质为 □(p→◇q)，其中原子命题 p 和 q 分别代表 x<=3 和 y>4。基于图 2 展示的 LNFG，对应于每一个序列片段，LNFG 中的可扩展路径如图 11 的中间列所示。在将每个片段的验证结果利用合成运算◦进行合并后，得到 A_0◦A_1◦A_2=∅，表明 LNFG 中不存在从初始节点到达 ε 节点的路径，即此状态序列满足性质□(p→◇q)。

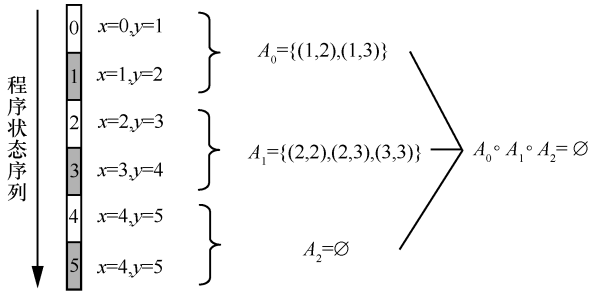


图 11 程序状态序列满足性质的情况

4 实验评估

KDD CUP99 是目前网络异常检测方法和系统评价中应用最广泛的数据集。为了评估本文所提的针对 Smurf、SYN Flood 和 Land 三类 DoS 攻击检测方法的可行性和有效性，从 KDD CUP99 数据集中获取相关记录，攻击行为数量分布如表 1 所示。

表 1 攻击行为数量分布

攻击类型	正常数量/个	攻击数量/个
Smurf	26 980	8 960
SYN Flood	12 560	5 800
Land	853	9

针对所获取的攻击数据，将本文方法与相关文献中模型检测和机器学习方法进行对比。文献[22]提出了一种攻击特征描述语言（ASDL, attack signature description language），该语言能够同时描述被监控程序与待验证性质，并在此基础上，提出了一种 ASDL 模型检测算法。文献[17]提出了面向边缘计算系统入侵检测的机器学习方法，并与其他机器学习方法进行了性能比较。本文实验使用 Windows 10 操作系统，CPU 型号为 Core i7-8700 @ 3.2 GHz，内存为 8 GB。需要指出的是，在使用本文方法进行入侵检测时，所创建的验证线程为 3 条。实验结果对比如表 2 所示。

表 2 实验结果对比

方法	准确率	误报率	训练时间/s	检测时间/ ms	内存/MB
本文方法	97.5%	1.2%	—	231	9
ASDL[22]	95.6%	2.1%	—	362	5
SS-ELM[17]	99.1%	0.4%	4.5	267	25
BP	84.2%	7.8%	387	321	76
SVM	94.3%	3.6%	15	365	37

分析实验结果，本文得出如下结论。

1) 与基于 RASL 的模型检测方法相比，由于本文使用的 PPTL 公式具有更强的表达能力，因此准确率较高；在检测时间和消耗内存方面，本文采用的多线程并行方法使验证效率更高，但同时会占用更多内存。

2) 与机器学习方法相比，本文方法比传统机器学习方法 BP（back propagation）神经网络和支持向量机（SVM, support vector machine）具有更高或相近的准确率，但与优化后的样本选择极限学习机（SS-ELM, sample selected extreme learning machine）相比尚有不足；在所需时间和消耗内存方面，由于不需要提前训练模型，并且在验证过程中采用了多线程并行方法，因此，本文方法能够节省模型训练时间，并在内存消耗方面有较大优势，同时需要更短的检测时间。

5　实例应用——智能停车系统

作为应用实例，本节将所提入侵检测方法应用于实际开发的基于边缘计算网络的智能停车系统中，以验证其在真实系统中的有效性。

5.1　模型概述

目前，智能停车系统已在智慧城市和智能交通系统中得到广泛应用。为了使其更加友好高效，文献[32]提出了一种基于边缘计算的 P2P 网络智能停车系统，其利用云计算、边缘计算和 P2P 网络技术提供大量的服务，包括停车位查询、导航、车牌识别、支付和事故查询等。智能停车系统共包括 5 个重要部分。1) 移动终端设备代表车辆，例如汽车或卡车，作为活动节点，其他节点都供该节点使用。终端设备不时地与其他节点通信并与之交互，例如云端服务器和边缘服务器。2) 停车场的边缘服务器用于本地边缘计算，完成所有与停车场有关的计算任务，例如云端注册、位置判断、车辆管理以及视频监控等。3) 第三方导航用于引导和服务车辆找到目的地停车场。4) 第三方付款用于处理停车活动产生的费用。5) 云端服务器用于与边缘服务器节点和终端设备进行交互，存储和提供一些数据量大且时延要求不高的数据。

5.2　性质表达

为了保证车辆在停车过程中边缘服务器能够正常提供所有服务，在边缘服务器提供服务的过程中，本节在边缘服务器部署并行运行时验证系统，对其行为进行入侵检测，包括异常检测和误用检测。为了实现异常检测，首先用 UML 顺序图对车辆从进入停车场开始的停车过程中各模块的行为进行建模，部分模型如图 12 所示。

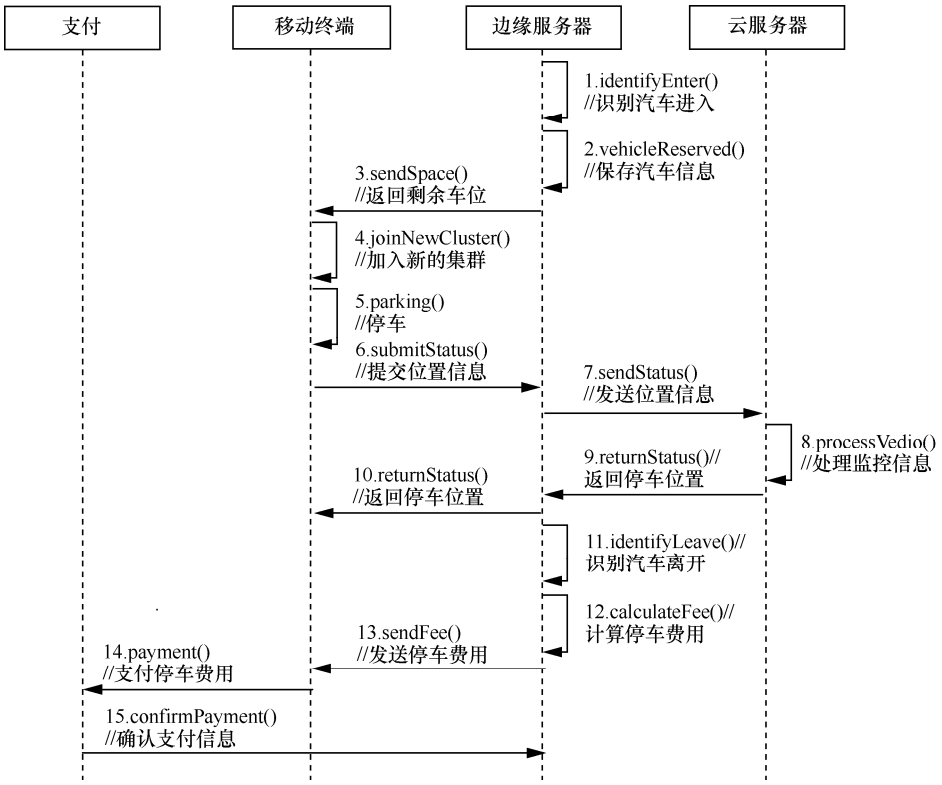


图 12　智能停车系统部分 UML 顺序图

由于本文只关注边缘服务器行为，因此只需分析边缘服务器生命线上的相关函数调用序列，可用如下 PPTL 公式对其进行形式化描述。

P_1≡◇(identifyEnter; vehicleReserved; sendSpace; sendStatus; returnStatus; identifyLeave; calculateFee; sendFee)$^+$

其中，操作符$^+$表示对于不同的车辆进入请求，边缘服务器将不断重复执行各个函数。总体来看，此 PPTL 公式表示从某一时刻起，边缘服务器将循环执行函数 identifyEnter、vehicleReserved、sendSpace、sendStatus、returnStatus、identifyLeave、calculateFee 和 sendFee。

针对 P_1 公式的运行时验证能够实现边缘服务器的异常检测，实时监控其各个函数是否被顺利调用执行，一旦发现 P_1 公式不被满足，可立即发出警告提示。在对边缘服务器进行异常检测的同时，可对 Smurf、SYN Flood 和 Land 攻击进行误用检测，即对 3.1.2 节中相对应的 PPTL 公式实施运行时验证，以判断攻击的类型，具体如下

$$P_2\equiv\Diamond(\neg\text{send}\rightarrow(\Diamond\text{receive})^+)$$

$$P_3\equiv\Diamond((\text{receive.SYN; send.SYNACK})\rightarrow\Box(\neg\text{receive.ACK}))$$

$$P_4\equiv\Diamond(\text{receive.SYN}\rightarrow\Diamond(\text{send.SYNACK; send.ACK})^+)$$

当 P_2、P_3 和 P_4 中某一个公式被满足时，即表明检测出相应类型的攻击。

5.3 系统验证

为了表明所提方法的有效性，针对智能停车系统，本节将运行时验证模块部署在边缘服务器以实时监控 P_1 公式是否被违反（即边缘服务器不能正常提供服务），以及 P_2～P_4 中某个公式是否成立（即检测到某个 DoS 攻击行为）。在实验中，使用安卓手机作为终端，5 台 PC 机作为边缘服务器，腾讯云提供云端服务。

实验共分为三组，分别使用边缘服务器中的单个验证线程来实现传统的运行时验证方法，以及 3 个和 5 个验证线程实现并行运行时验证方法，其中每次验证的序列片段包含 1 000 个程序状态。在每组实验中，将一个恶意节点加入边缘计算网络后，对边缘服务器实施 Smurf 攻击，为了证明所提方法的有效性以及验证效率的准确性，均重复攻击 10 次。针对 Smurf 攻击的检测结果如表 3 所示。

表 3 中，第一列为待验证的 PPTL 公式，包括用于异常检测的公式 P_1 以及用于误用检测的公式 P_2～P_4；第二列表示验证工具给出的验证结果，即程序是否满足相应性质，P_1 为×表明检测出边缘服务器没有正常提供服务，P_2 为√表明检测出攻击类型为 Smurf 攻击；第三列至第五列给出了在边缘服务器中，不同数量验证线程提供验证算力的情况下，每验证 1 000 个状态所需要的时间。比较第三列至第五列给出的验证时间可以发现，当 3 个验证线程提供验证算力时，验证效率最高，而当 5 个验证线程提供验证算力时，验证效率反而会降低。这是由于边缘服务器虽然能够支持一定数量的并行线程，但边缘服务器的 CPU 计算能力不强，5 个验证线程导致 CPU 的线程切换开销过大，影响整体性能。总体来看，3 个验证线程提供验证算力的情况下，在 Smurf 攻击实施 7 s 内，本文所提的并行运行时验证方法能够检测出入侵攻击。针对 SYN Flood 和 Land 攻击的检测结果能得到相似结论。

6 结束语

针对边缘计算系统中边缘服务器面临的 DoS 攻击，本文提出了一种并行运行时验证方法，该方法结合误用检测和异常检测，充分利用边缘服务器的计算与存储资源。首先使用 PPTL 公式，分别描述系统正常状态和 3 种典型的 DoS 攻击的行为特征；然后将上述两类 PPTL 公式作为待验证性质，在边缘服务器上部署并行运行时验证方法，在边缘服务器程序运行过程中，利用边缘服务器中的多个

表 3 针对 Smurf 攻击的检测结果

公式	性质是否满足	单个验证线程提供验证算力/ms	3 个验证线程提供验证算力/ms	5 个验证线程提供验证算力/ms
P_1	×	641	303	385
P_2	√	392	217	276
P_3	×	313	158	187
P_4	×	432	249	293

验证线程，完成 DoS 攻击的入侵检测；最后通过对比实验，验证了所提方法的可行性和有效性，并对一个实际的基于边缘计算的 P2P 网络智能停车系统进行模拟 DoS 攻击和攻击检测。实验表明，所提方法能够有效识别 Smurf、SYN Flood 和 Land 攻击。在今后的研究工作中，将进一步提高基于运行时验证的入侵检测效率，且对更多类型的入侵攻击进行形式化描述与检测。

参考文献：

[1] PANG H H, TAN K L. Authenticating query results in edge computing[C]//Proceedings of 20th International Conference on Data Engineering. Piscataway: IEEE Press, 2004: 560-571.

[2] GEORGE G, THAMPI S M. Vulnerability-based risk assessment and mitigation strategies for edge devices in the Internet of Things[J]. Pervasive and Mobile Computing, 2019, 59: 101068.

[3] ROMAN R, LOPEZ J, MAMBO M. Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges[J]. Future Generation Computer Systems, 2018, 78: 680-698.

[4] SHIRAZI S N, GOUGLIDIS A, FARSHAD A, et al. The extended cloud: review and analysis of mobile edge computing and fog from a security and resilience perspective[J]. IEEE Journal on Selected Areas in Communications, 2017, 35(11): 2586-2595.

[5] SHI W S, CAO J, ZHANG Q, et al. Edge computing: vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[6] MOUSTAFA N, HU J K, SLAY J. A holistic review of network anomaly detection systems: a comprehensive survey[J]. Journal of Network and Computer Applications, 2019, 128: 33-55.

[7] AHMAD I. A survey on DDoS attacks in edge servers[D]. Arlington: The University of Texas at Arlington, 2020.

[8] GAUTAM D, TOKEKAR V. An approach to analyze the impact of DDoS attack on mobile cloud computing[C]//2017 International Conference on Information, Communication, Instrumentation and Control. Piscataway: IEEE Press, 2017: 1-6.

[9] CAPROLU M, DI PIETRO R, LOMBARDI F, et al. Edge computing perspectives: architectures, technologies, and open security issues[C]//2019 IEEE International Conference on Edge Computing. Piscataway: IEEE Press, 2019: 116-123.

[10] SINGH J, BELLO Y, HUSSEIN A R, et al. Hierarchical security paradigm for IoT multiaccess edge computing[J]. IEEE Internet of Things Journal, 2021, 8(7): 5794-5805.

[11] RAZA S, WALLGREN L, VOIGT T. SVELTE: real-time intrusion detection in the Internet of things[J]. Ad Hoc Networks, 2013, 11(8): 2661-2674.

[12] MIDI D, RULLO A, MUDGERIKAR A, et al. Kalis—a system for knowledge-driven adaptable intrusion detection for the Internet of things[C]//2017 IEEE International Conference on Distributed Computing Systems. Piscataway: IEEE Press, 2017: 656-666.

[13] HODO E, BELLEKENS X, HAMILTON A, et al. Threat analysis of IoT networks using artificial neural network intrusion detection system[C]//2016 International Symposium on Networks, Computers and Communications. Piscataway: IEEE Press, 2016: 1-6.

[14] WANG Y, XIE L, LI W, et al. A privacy-preserving framework for collaborative intrusion detection networks through fog computing[C]//2017 International Symposium on Cyberspace Safety and Security. Berlin: Springer, 2017: 267-279.

[15] MENG W Z, WANG Y, LI W J, et al. Enhancing intelligent alarm reduction for distributed intrusion detection systems via edge computing[C]//2018 Australasian Conference on Information Security and Privacy. Berlin: Springer, 2018: 759-767.

[16] 肖阳, 白磊, 王仙. 基于朋友机制的移动 ad hoc 网络路由入侵检测模型研究[J]. 通信学报, 2015, 36(S1): 203-214.
XIAO Y, BAI L, WANG X. Friends mechanism-based routing intrusion detection model for mobile ad hoc network[J]. Journal on Communications, 2015, 36(S1): 203-214.

[17] AN X S, ZHOU X W, LYU X, et al. Sample selected extreme learning machine based intrusion detection in fog computing and MEC[J]. Wireless Communications and Mobile Computing, 2018, 2018: 1-10.

[18] LIN F H, ZHOU Y T, AN X S, et al. Fair resource allocation in an intrusion-detection system for edge computing: ensuring the security of Internet of things devices[J]. IEEE Consumer Electronics Magazine, 2018, 7(6): 45-50.

[19] NALDURG P, SEN K, THATI P. A temporal logic based framework for intrusion detection[C]//2004 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE). Berlin: Springer, 2004: 359-376.

[20] AHMED A, LISITSA A, DIXON C. TeStID: a high performance temporal intrusion detection system[C]//International Symposium on Telecommunications. Piscataway: IEEE Press, 2013: 20-26.

[21] AHMED A, LISITSA A, DIXON C. A misuse-based network intrusion detection system using temporal logic and stream processing[C]//2011 5th International Conference on Network and System Security. Piscataway: IEEE Press, 2011: 1-8.

[22] DENG M L, CAO H L, ZHU W J, et al. Benchmark tests for the model-checking-based IDS algorithms[J]. IEEE Access, 2019, 7: 135479-135498.

[23] OLIVAIN J, GOUBAULT-LARRECQ J. The orchids intrusion detection tool[C]//2005 International Conference on Computer Aided Verification. Berlin: Springer, 2005: 286-290.

[24] SUN Y, WU T, MA X Q, et al. Modeling and verifying EPC network intrusion system based on timed automata[J]. Pervasive and Mobile Computing, 2015, 24: 61-76.

[25] PNUELI A. The temporal logic of programs[C]//18th Annual Symposium on Foundations of Computer Science. Piscataway: IEEE Press, 1977: 46-57.

[26] CLARKE E M, EMERSON E A. Design and synthesis of synchroni-

zation skeletons using branching time temporal logic[C]//1981 Workshop on Logic of Programs. Berlin: Springer, 1981: 52-71.

[27] TIAN C, DUAN Z. Propositional projection temporal logic, büchi automata and ω-regular expressions[C]//2008 International Conference on Theory and Applications of Models of Computation. Berlin: Springer, 2008: 47-58.

[28] DUAN Z. An extended interval temporal logic and a framing technique for temporal logic programming[D]. Newcastle: Newcastle University, 1996.

[29] DUAN Z, TIAN C. A practical decision procedure for propositional projection temporal logic with infinite models[J]. Theoretical Computer Science, 2014, 554: 169-190.

[30] 张琛, 段振华, 田聪. 基于事件确定有限自动机的 UML2.0 序列图描述与验证[J]. 软件学报, 2011, 22(11): 2625-2638.
ZHANG C, DUAN Z H, TIAN C. Specification and verification of UML2.0 sequence diagrams based on event deterministic finite automata[J]. Journal of Software, 2011, 22(11): 2625-2638.

[31] SHEN H, ROBINSON M, NIU J. A logical framework for sequence diagram with combined fragments[R]. 2011.

[32] ZHANG N, LU X, TIAN C, et al. P2P network based smart parking system using edge computing[J]. Mobile Networks and Applications, 2020, 25(6): 2226-2239.

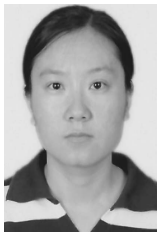
[作者简介]

于斌（1990− ），男，河南漯河人，博士，西安电子科技大学讲师，主要研究方向为模型检测、运行时验证。

张南（1984− ），女，天津人，博士，西安电子科技大学副教授、博士生导师，主要研究方向为形式化验证、模型检测。

陆旭（1985− ），男，河北承德人，博士，西安电子科技大学讲师，主要研究方向为可信软件、分离逻辑。

段振华（1948− ），男，陕西咸阳人，博士，西安电子科技大学教授、博士生导师，主要研究方向为时序逻辑、形式化验证。



田聪（1981− ），女，陕西合阳人，博士，西安电子科技大学教授、博士生导师，主要研究方向为形式化验证、模型检测。