密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：物联网服务系统运行时验证系统的研究与实现

学　　号：2015111609

姓　　名：陈　宽

专　　业：计算机科学与技术

导　　师：章　洋

学　　院：网络技术研究院

2018 年 01 月 21 日

Secret level： Confidentiality period：

Beijing University of Posts and Telecommunications

Master Thesis

Title: Research and implementation of the runtime verification system for IoT service system

Student Number: 2015111609

Name: Chen Kuan

Major: Computer of Science and Technology

Tutor: Zhang Yang

Institute: Institute of Network Technology

2018.01.21

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

物联网服务系统运行时验证系统的研究与实现

摘　要

随着物联网的快速发展，物联网服务系统在给人们带来便利的同时，也带来了极大的安全隐患，在运行中系统需要更多的安全保障。已有的运行时验证方案往往嵌入到物联网服务系统中，当系统受到入侵时因为缺乏相应的隔离性而不能对系统进行可靠的保障。

本文介绍了一种运行时验证系统，基于高安全级别的虚拟化技术，将物联网服务系统部署在虚拟机中。通过控制虚拟机与外界通信渠道，从网络、串口、内存三个角度提取物联网服务系统运行事件，并对事件进行多来源融合，然后进行事件验证，最后根据安全策略反馈调节服务系统。

本文的创新点首先在于事件采集于服务主机外部，对服务本身是透明的，具有高度的隔离性，能在物联网服务系统受到入侵时有更高的可靠性。在此之上本文创新性地将多来源提取的事件进行融合，提出一系列事件融合验证与事件重复、遗漏应对算法，解决了运行时验证系统与物联网服务系统分离带来的事件获取不精确问题。最后，本文将定理证明与运行时验证相结合，通过运行时实例化状态机节点变量来实现状态机的转移验证，规避了传统静态模型检测的效率问题。

本文通过多种测试和实验来验证本文所述方法可靠性和高效性。

关键词　物联网　运行时验证　虚拟化　转移验证

RESEARCH AND IMPLEMENTATION OF THE RUNTIME VERIFICATION SYSTEM FOR IOT SERVICE SYSTEM

ABSTRACT

With the rapid development of the IoT (Internet of Things), the IoT service system brings convenience to people as well as great security risks. The system needs more security during operation. Existing runtime verification schemes are often embedded in the IoT service system, which cannot be reliably guaranteed because of lack of corresponding isolation when the system is invaded.

This paper describes a runtime verification system based on high security virtualization technology that deploys IoT service systems in virtual machines. By controlling the communication channel between the virtual machine and the outside world, IoT service system operation events are extracted from the perspectives of network, serial port and memory, and the events are fused from multiple sources. Then event verification is carried out. Finally, the service system is fed back according to the security policy.

The innovation of this paper lies in the fact that the event is collected

outside the service host and is transparent to the service itself. It is highly isolated and has higher reliability when the IoT service system is compromised. Above this innovative fusion of multiple sources of extracted events, a series of event fusion verification and event duplication, omission coping algorithms, to solve the runtime authentication system and the Internet of Things service system to separate the event acquisition is not accurate problem. Finally, this paper combines the theorem proving and runtime verification, and realizes the state machine transfer verification by instantiating state machine node variables at run time, which avoids the efficiency problem of traditional static model testing.

This article through a variety of tests and experiments to verify the reliability and efficiency of the method described in this article.

KEY WORDS Internet of Things; runtime verification; virtualization; transfer verification

目录

第一章 绪论

1.1 背景

物联网服务系统是新一代信息技术的重要组成部分，也是“信息化”时代的重要发展阶段。其发展正在促进着人们的生产和生活等方式向着更加现代化与更加智能化的方向发展。但是随着物联网服务系统与外界进行通信更加频繁与其本身模式变得更加开放，整个物联网服务系统的运行环境将更加复杂化[1]。在这样的背景之下，物联网服务系统的通信安全问题变得越来越重要。如何在保持物联网开放的同时提高其通信安全并对整个系统进行安全保障是急需解决的问题。

对物联网服务系统运行时验证的研究主要指监控服务系统在运行中的行为，判断其行为是否满足属性策略。Barringer 等人在 2004 年对软件系统运行时验证技术进行了综述[2]，概述了基于自动机和基于规则的运行时监控器研究方法。其中一个主要进展是引入了具有编辑能力的安全自动机，可对被监控的系统违反策略规定的行为进行编辑修改。Ligatti 等人在 2010 年提出了 MRA(Mandatory Result Automata, 强制性结果自动机)框架[3]，在该框架中，运行时监控器从目标获取请求，向执行环境发送动作，以执行系统的结果响应目标。我们的研究方法也遵照这种框架，但是该框架要求在运行时监控器和目标系统之间有可靠的通信信道。在我们研究的物联网环境中，这种假设不成立，即被隔离的物联网服务系统和运行时监控器之间缺少可靠的通信信道，我们采取多种方式对物联网的运行状态进行观测，并将这些可能相互矛盾的观测结果进行融合。

有一些研究人员针对物联网应用系统提出了运行时保护方法。如 Gamage 在 2010 年提出的物联网服务属性保障的框架[4]。其工作旨在检查自动机是否遵循期望的交互描述以及有界模型检查。虽然该方法完全基于现代 SMT 求解器的先进特性，如可满足性的验证判断，但它缺少将隔离的服务运行环境进行建模和隔离验证的过程。Mitsch 和 Platzer 在 2016 年将系统模型的离线验证与运行时验证相结合，通过使用模型监视器以保证物理系统的安全[5]。他们提出了 ModelPlex 方法，将控制器监视器与预测监视器综合起来，以保护实际系统免受已验证模型的偏离。但是没有考虑如何将不同观察源的观测结果融合到隔离的运行时监控器中。

从现阶段学术界对该领域的已有研究分析可以看出，物联网服务系统的保障在沿着运行时验证更高效更先进的同时，缺少相应的隔离验证方法，从而难以在对物联网服务系统保障的同时对运行时验证系统进行相应的保障[6]。

近年来，虚拟化技术的使用为隔离式物联网保障系统提供新的解决方案。通过将物联网服务系统部署在隔离的虚拟化环境中，在服务主机外部控制物联网服务系统与外界的通信通道，可以以高度的隔离性获取服务的运行状态。之后将多观察源的事件进行融合验证，可以对隔离性事件获取带来的事件重复、遗漏等情况进行最大化规避。最后在宿主机中对物联网服务系统进行控制反馈，从而能以更高的可靠性与安全性对物联网服务系统提供完整的安全保障[11]。

本文在虚拟化技术的基础上，提出一种完整的物联网服务系统运行时验证系统。该系统实现了服务系统的高度隔离化，融合了三个观察源采集到的服务系统状态，提出一系列事件融合验证算法避免了事件重复、遗漏等情况，最后将事件采集与事件反馈相结合，为物联网服务系统的安全提供完善的保障服务。

1.2 研究工作

通过对传统的物联网服务系统保障系统进行深入研究，我们可以发现物联网服务系统保障系统在系统原始数据获取、数据分析验证、反馈保障等各部分均存在很多的缺陷与不足，不能很好地对原有物联网服务系统进行保障工作。首先，传统的物联网服务系统保障系统与被保障的物联网服务系统耦合过于紧密，主要表现在保障与被保障系统均部署在同一操作系统中，能够被操作系统同时感知。这样当恶意软件入侵到该操作系统中时，不仅会对被保障的物联网系统进行数据修改，还可能会对保障系统自身进行数据修改，从而造成保障系统从数据采集到数据分析各个步骤的结果都不可靠。其次，传统的保障系统往往对检测到的物联网服务系统异常仅仅提供告警功能，由用户手动进行排除异常，造成异常情况处理的不及时。再次，数据分析验证行为方法的简单性与低效性也是传统的保障系统的弊病，在物联网系统运行繁忙网络与串口通信数据量较大的情况下，整个物联网系统的运行效率将被分析验证的效率所掣肘。并且随着物联网系统的复杂化，对数据验证环节的需求将持续增加，传统的验证方法将难以适应新需求的提出。最后，传统的物联网服务系统保障系统往往存在缺乏相应的系统扩展性与易用性，非专业人员难以配置使用，在需求变更时难以及时适应等问题。

对于以上描述的问题，本于提出了一整套完整的物联网服务系统运行时验证系统的解决方案。一方面使用虚拟化技术并且对数据获取部分进行改进和添加获取方法，提高数据获取的准确性与运行效率，增加数据来源的多样性；一方面对数据分析验证进行标准化、模式化，使其适用于大多数物联网服务系统的环境，并最大程度地对验证效率进行优化提高；另一方面将数据分析后反馈与数据获取相结合，以高度地准确性对非法数据进行拦截处理[8]。

首先，采用虚拟化技术，将被保障的物联网服务系统部署在虚拟机中，将运行时验证系统部署在宿主机中，实现了保障与被保障系统的分离。此时即使虚拟机被恶意软件入侵，仅可以修改物联网服务系统的数据内容，并不会影响到运行时验证系统，极大地提高了运行时验证系统在数据获取、数据分析验证等环节的可靠性。数据获取将扩大数据来源，将传统物联网服务系统保障系统中容易忽略和极少涉及的串口通信、内存变化纳入其中，同时改进网络通信的数据获取，并且将物联网服务系统中的数据抽象成事件进行统一处理。三个来源的事件获取均在宿主机中进行，通过相应的宿主机访问虚拟机技术，提供高可靠的运行中事件。

其次，运行时验证系统将用户对物联网服务系统的事件和事件变化提出的要求表示成可视化的状态机模型，在事件采集后输入状态机模型进行事件的融合后转移验证分析，得到相应的转移结果。该验证过程不仅能够综合三个事件来源的事件，而且可以实现事件的重复、遗漏等情况的应对。由于状态机模型是由用户预先定制，因此验证系统可以有很强的可定制性与扩展能力[9]。

第三，将事件验证后反馈与事件获取相结合，使事件获取部分不仅有对物联网服务系统通信数据简单读取的能力，同时具备一定的可修改能力。从而在虚拟机内物联网服务系统无感知的情况下在宿主机中进行高效率的事件反馈。

第四，为了提高系统的扩展性和用户友好性，本文实现了一整套对应的界面系统供用户使用，包括可视化的状态机图形绘制界面、运行时系统参数定制与运行展示界面。用户可以通过简单地操作实现状态机的自定义，系统参数的输入，同时在启动运行时验证系统后可以方便地查看事件的获取与反馈拦截等关键信息。

本文将提出的一整套包含事件采集、事件验证、事件反馈等组成的理论模型与实际应用环境相结合，开发出可实际应用的物联网服务系统运行时验证系统，并在实际环境下进行全方面的功能测试和性能测试。测试结果可以表明，本文提出的解决方案不仅可以实现高度的运行时验证系统隔离性，同时可以高效地实现对物联网服务系统验证保障，并对其造成最小程度的影响。

综合以上工作可知，本文所提出的整个运行时验证系统不仅具有创新性的理论支撑，更能以实际的表现真正应用到真实的生产环境中为物联网的发展作贡献。

1.3 论文组织结构

根据以上研究结果可知，本文基于虚拟化技术和状态机事件转移模型提出了一整套的物联网服务系统运行时验证系统。文章的组织结构如下：

第一章：绪论，概述了文章的研究背景和研究内容；

第二章：相关技术概述，对在实现物联网服务系统运行时验证系统过程中实际需要使用的关键技术进行了简要介绍。

第三章：系统需求分析，主要阐述了目前物联网服务系统对运行时验证的需求，以及运行时验证系统需要哪些功能以完整地实现这些需求。

第四章：系统概要设计，主要从理论设计上阐述各模块的实现方法、相应的算法细节以及主要流程的运行过程。

第五章：详细设计与实现，主要从实际开发应用的角度上对运行时验证系统各模块如何开发以及各模块间如何衔接等进行详细阐述。

第六章：系统测试，设计系统功能和性能测试用例，得出相应的测试数据结果，并进行分析总结。

第七章：总结与展望，对目前已有的工作成果进行总结分析，指出现有工作的不足并提出下一步的改进方向。

1.4 本章总结

本章概述了物联网服务系统运行时验证系统的技术背景以及本文的主要研究工作，并且针对传统方案缺陷与不足提出更高效合理的解决方案，确定了本文研究的运行时验证系统整体组成与各部分功能。同时，本章给出了论文的组织结构，使得读者能够清晰地了解论文的展开与系统设计实现过程。

第二章 相关技术概述

由第一章的绪论内容可知，本文论述的整体物联网服务系统的运行时验证系统是基于多种事件采集技术以及事件验证和反馈技术的。其中事件的采集与事件反馈依托于现行的虚拟化技术，在虚拟机所在的宿主机添加网络流量捕获模块实现网络事件的采集，在虚拟机监控器利用虚拟的伪终端设备实现串口通信的中继与事件采集，在宿主机利用虚拟机内存访问技术对虚拟机进行无感知的内存读取进行内存事件的采集。事件的验证依托于定理证明技术，将采集到的事件代入相应的模型进行事件证明验证。此外，整个验证系统内部还使用了多线程和 I/O 复用技术来满足事件采集来自多个来源的协调问题。为了使后续系统概要设计描述更为清楚，本章对本文涉及到的关键技术进行简要介绍。

2.1 虚拟化技术概述

2.1.1 虚拟化与虚拟机

虚拟化，是指通过软件的方法重新定义划分计算机资源，将一台计算机虚拟为多台逻辑计算机的技术。其中计算机资源包括 CPU、内存、外存、外设、网络等。虚拟化技术实现了对计算机资源与实体结构的分割，可以根据实际需要来分配与使用这些资源，从而达到最大化的利用资源。同时对资源的新虚拟视图并不受地理位置、实现或者底层资源的物理配置所限制，这就极大提高了资源使用的灵活性与高效性。

虚拟机是使用虚拟化技术实现的模拟具有完整硬件系统功能且运行在严密隔离环境中的完整操作系统。虚拟机是虚拟化技术的一种实现应用，可以像物理计算机一样拥有独立的操作系统和软件，并通过硬件虚拟化技术虚拟出 CPU、内存等资源置于虚拟机中。从虚拟机内部来看，这些资源与实体硬件资源无异，即虚拟机内部无法感知自身是处于虚拟机中还是真实物理机器之中。虚拟机与真实物理机器的不同仅在于虚拟机的资源均由其所在的宿主机通过软件模拟出来，而宿主机的资源需要真实存在。因此，虚拟机在封装性与隔离性上有着真实物理机器无法比拟的独特优点[10]。

目前比较流行的虚拟机软件包括 VMware Workstation、Virtual Box、XEN、Hyper-V、KVM、QEMU 等。

KVM(kernel-based virtual machine, 内核级虚拟机)是 x86 架构且硬件支持虚拟化技术的 linux 全虚拟化解决方案。它使用 linux 自身的调度器进行管理，已经成为学术界的主流虚拟机之一。KVM 目前是 linux 内核的一个功能模块，在

linux2.6.20 之后的任何 linux 分支中都被支持，其通过调用 linux 内核功能，可以实现对 CPU 和内存的底层虚拟化，其系统架构如图 2-1 所示[7]：

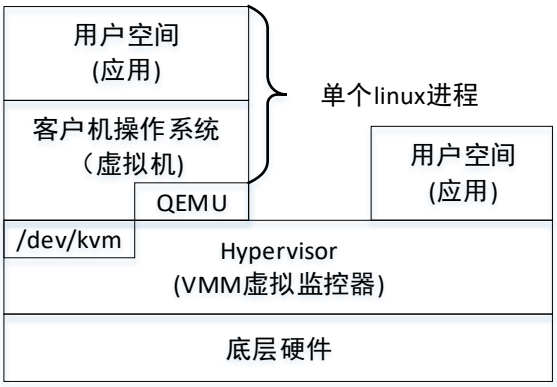

图 2-1 KVM 系统架构

QEMU 是一个通用开源的机器模拟器和虚拟化软件可以独立模拟出整台计算机，包括 CPU、内存、I/O 设备等。

QEMU-KVM 是 QEMU 与 KVM 的结合，使 KVM 运行在内核态，提供 CPU 和内存的虚拟化，而 QEMU 运行在用户态，模拟创建、管理各种虚拟硬件，包括磁盘、网卡、显卡等设备。

从软件开源、流行程度以及兼容性上综合考虑，本文采用 QEMU-KVM 虚拟化技术来实现整个物联网服务系统运行时验证系统。在实际应用中，QEMU-KVM 通常被简称作 KVM，本文将沿用此简称，下文中所提到的 KVM 均表示 QEMU-KVM。

2.1.2 VMM

VMM(Virtual Machine Monitor, 虚拟机监控器)是一个负责维护多个互相隔离的独立虚拟机的系统软件。严格意义上的 VMM 是指在物理资源与虚拟机之间的程序，其向下负责管理真实计算机的资源，向上为虚拟机提供资源的统一访问接口，同时 VMM 提供资源上的规划、虚拟机的部署管理等功能。广义上的 VMM 是将狭义的 VMM 与虚拟机合并称为 VMM。借助于 VMM 的功能与接口，我们一方面可以同时运行管理多个虚拟机，另一方面可以从接口上直接对虚拟机进行相应的程序内访问与操作。

本文在使用 KVM 作为虚拟机基础上，同时使用 libvirt 和 virt-manager 作为 VMM 选择方案。

libvirt 是一套免费、开源的支持 linux 下主流虚拟化工具的 C 函数库。其旨在为包括 KVM 在内的各种虚拟化工具提供一套方便、可靠的 API 编程接口。

virt-manager 是用于管理 KVM 虚拟机的主要工具，其使用 Unix socket 调用 libvirt 来实现对 KVM 的管理。此外，virt-manager 还通过一个嵌入式虚拟网络客

户端查看器为虚拟机提供一个完整的图形控制台，极大的方便了用户对 KVM 的管理操作。

2.1.3 KVM 网络模式

KVM 有四种简单的网络模式，分别是路由模式、NAT 模式、隔离模式、桥接模式。

路由模式是指 VMM 建立虚拟路由，将 KVM 都作为虚拟路由下子网中一台设备。当 KVM 与外界通信时，虚拟路由根据目的地址进行转发。但虚拟路由并不修改 KVM 发出的 IP 数据报中的源地址，因此源地址仍然为虚拟路由下子网的私有 IP，这将导致外界返回的 IP 数据报可能无法回传给 KVM。

NAT(Network Address Translation, 网络地址转换)模式与路由基本相同，其区别在于虚拟路由转发 KVM 发出的 IP 数据报时，将源 IP 地址(即 KVM 私有 IP 地址)转换为为外的物理网卡 IP 地址，在接收到外界回传到 KVM 的 IP 数据报时，转发给相应的 KVM。NAT 模式的原理如图 2-2 所示[12]：

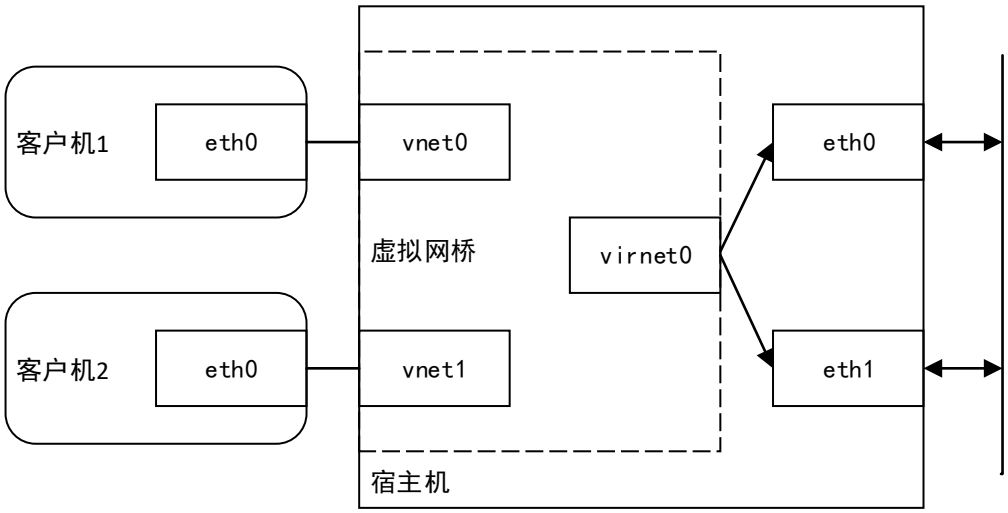

图 2-2 NAT 模式

隔离模式是指 VMM 建立虚拟网桥，将 KVM 都连接到此网桥上，但该虚拟网桥并未桥接物理网卡。因此相当于 KVM 之间组建了私有网络，无法与外界进行通信。

桥接模式是指 VMM 建立虚拟网卡作为宿主机对外的通信网卡，将 KVM 与宿主机均桥接到该虚拟网卡之上。此时该虚拟网卡作为虚拟网桥具有交换机功能，KVM 与宿主机在虚拟网桥下处于同等地位。因此 KVM 可以获取原宿主机所在网段的相应 IP 地址。桥接模式原理如图 2-3 所示[13]：

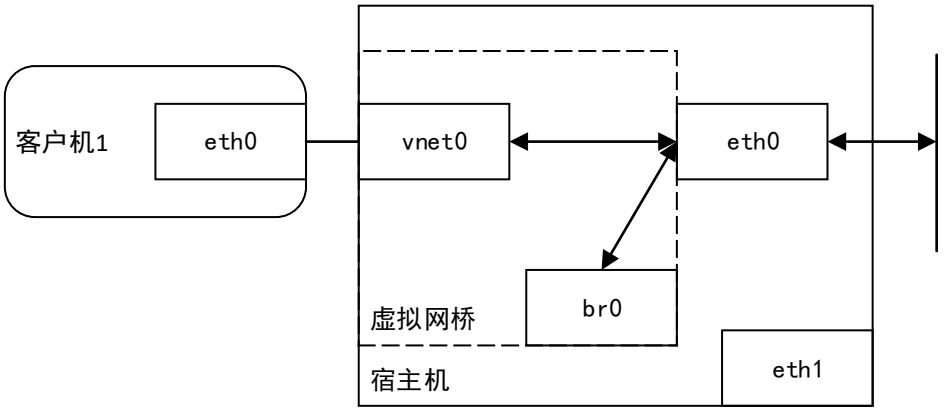

图 2-3 桥接模式

本文所研究的物联网服务系统需要 KVM 与外界进行网络交互通信，因此可选的网络模式为 NAT 与桥接。其中 NAT 模式下 KVM 持有的是私有 IP 地址，除非借助 NATP(Network Address Port Translation)等端口映射技术，外界无法主动发起与 KVM 的连接。同时在大多数的使用场景下，物联网服务的部署要求部署机器拥有所在网段的 IP 地址。因此本文首选研究的网络模式为桥接。

2.2 内核 hook 技术概述

2.2.1 Netfilter 内核模块

Netfilter 是 linux2.4.x 引入的一个子系统，是内核中的一个软件框架，用于管理网络数据包。它作为一个通用的、抽象的框架，提供了一整套的 hook 函数机制，可实现安全策略应用中的许多功能，如数据包过滤、数据包处理、地址伪装、透明代理、NAT，以及基于用户及 MAC(Media Access Control, 媒体介入控制层)地址的过滤和基于状态的过滤、包速率限制等。利用运行于用户态的应用软件，如 iptables 等，可以控制 Netfilter 进而管理 linux 操作系统的网络数据包[14]。

Netfilter 的架构就是在网络流程的若干位置放置了一些 hook 点，而在每个 hook 点上挂载了一些处理函数进行处理。

Netfilter 在网络层的挂载点有 5 个，分别是 PRE(pre_routing)、IN(local_in)、FWD(forward)、OUT(local_out)、POST(post_routing)，代表着 linux 内核中 ip 数据包的流经路线，如图 2-4 所示：

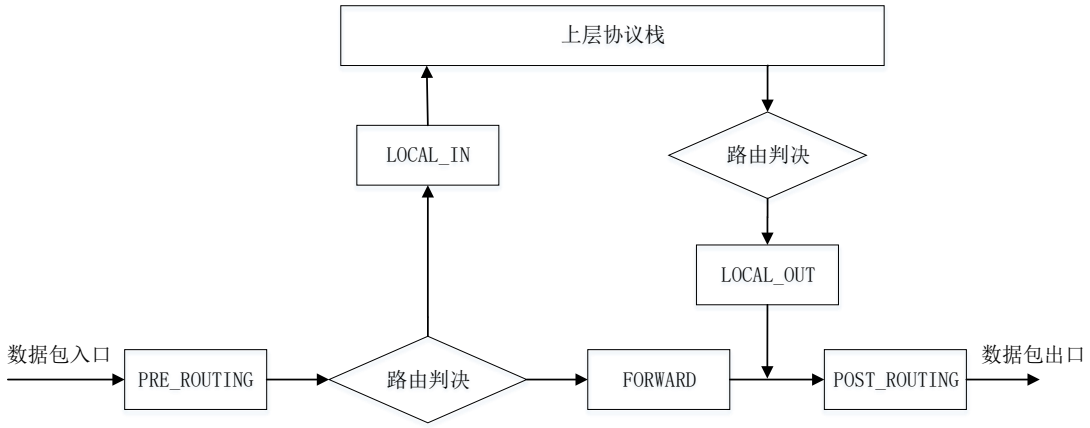

图 2-4 Netfilter 在网络层的 hook 点

与此同时，Netfilter 在数据链路层的挂载点有 6 个，分别是 brouting、pre_routing、local_in、forward、local_out、post_routing，代表着 linux 内核中以太网帧的流经路线，如图 2-5 所示：

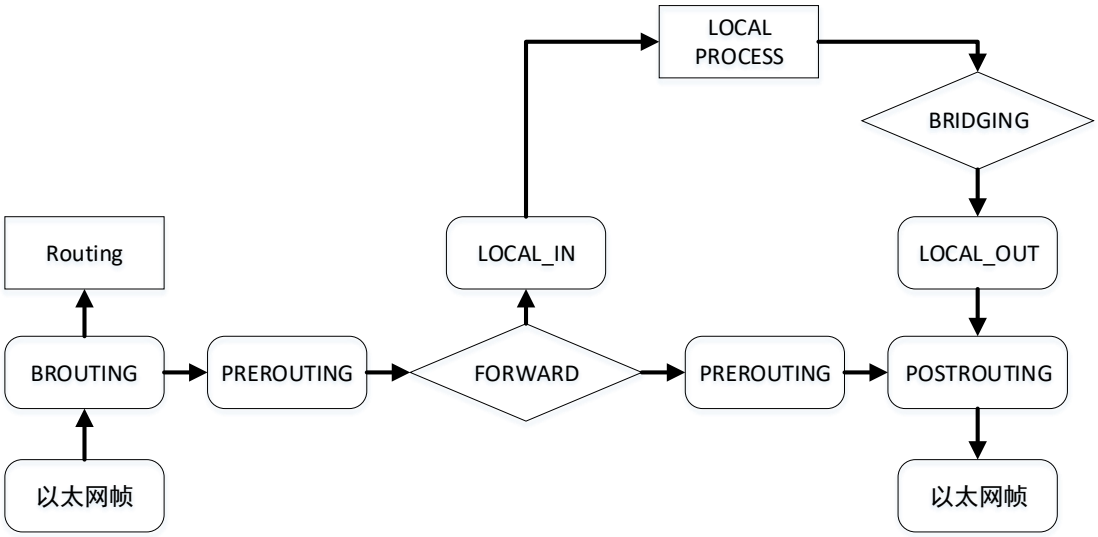

图 2-5 Netfilter 在数据链路层的 hook 点

2.2.2 Netlink 内核态与用户态通信

Netlink 套接字是以实现用户进程与内核进程通信的一种特殊的 IPC(Interprocess communication, 进程间通信)，也是网络应用程序与内核通信的最常用接口。其不像普通网络套接字一样可以用于主机间通讯，Netlink 只能用于同一主机上进程通讯，并通过 PID(Process Identification, 进程标识符)来标识它们。

Netlink 用于替代复杂和不够灵活的 ioctl 通讯方式，在 linux 内核 2.2 版本中作为字符设备被首次提供使用。Netlink 套接字可以使用标准的套接字 API 来创建与使用，用户也可以方便地在内核中添加自定义的 Netlink 协议进行通信，借此可以使用 Netlink 连接用户自定义的内核模块[15]。

2.3 伪终端技术概述

2.3.1 伪终端

伪终端(Pseudo Terminal)是成对的逻辑终端设备，与实际物理设备并不相关，可分为伪终端主设备和伪终端从设备，对主设备的操作会反映到从设备上。如果把一个伪终端主设备看作是一个串行端口设备，则它对该串口的读写操作都会反映在该伪终端设备对应的另一个伪终端从设备上，而伪终端从设备则是另一个程序用于读写操作的逻辑设备[16]。

2.3.2 KVM 串口与伪终端映射

KVM 可通过多种方式映射成串口设备，包括宿主机字符设备、伪终端、文件、TCP/UDP 网络控制台、Unix 套接字。

宿主机字符设备是宿主机自身连接的物理设备，如果需要 KVM 访问物理串口与外界进行直连通信，则宿主机字符设备应该是宿主机上对应的串口设备路径。

伪终端映射串口可以实现在宿主机中读写该伪终端，KVM 内读写该伪终端映射的串口完成虚拟的串口链路通信过程。

文件、TCP/UDP 网络控制台以及 Unix 套接字等可以实现 KVM 读写虚拟串口的数据记录、远程读写物理串口等功能。

2.4 虚拟化监控与内存取证

2.4.1 虚拟化监控

伴随着迅速发展的虚拟化技术，VMM 在对虚拟机安全的监控上有了新的解决办法。由于虚拟机自身拥有高度的隔离性，虚拟机内部对外部处于无感知状态，因此可以实现监控系统与被监控系统的分离，即将原有的监控系统从部署在被监控系统同一操作系统中转变成将监控系统部署在被监控系统所在虚拟机外部的宿主机中。

在将监控系统分离之后，原有的直接访问被监控系统信息的渠道被切断，因此需要新的从宿主机直接访问虚拟机信息的技术。

VMI(virtual machine introspection, 虚拟机自省)技术是指通过在虚拟机外部来实现对虚拟机内部运行状态监控的方法。由于虚拟机的内存是宿主机内存的一部分，因此利用 VMI 技术可以实现宿主机实时访问到虚拟机的内存。

LibVMI 是一个 C 函数库，它提供了对正在运行的虚拟机运行细节进行监视的功能。监视的功能是由观察内存细节，陷入硬件事件和读取 CPU 寄存器来完成的。其原理如图 2-6 所示：

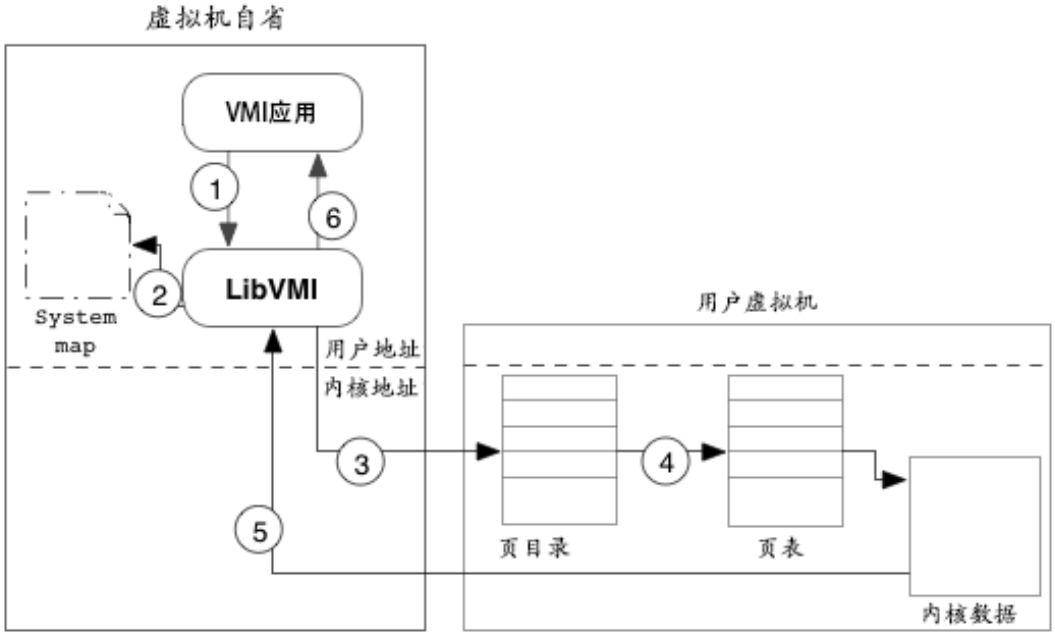

图 2-6 LibVMI 原理图

2.4.2 内存取证

内存取证是指从计算机物理内存和页面交换文件中查找、提取和分析易失性数据，是对传统文件系统取证的重要补充。

当操作系统处于活跃状态时，物理内存中保存着系统运行的关键信息。将这些关键信息进行重构，即可还原出原先操作系统中状态信息。

Volatility 是一款开源的内存取证框架，能够对导出的内存镜像进行分析，通过获取内核数据结构，使用相应的插件获取内存的详细情况以及系统运行状态。

将内存取证与 VMI 进行结合，即可在虚拟机处于运行状态下提取虚拟机内操作系统中的数据结构，从而获取其中关键运行信息[17]。

2.5 多线程与 I/O 复用

2.5.1 多线程

多线程(multithreading)是指从软件或者硬件上实现多个线程并发执行的技术。线程是操作系统调度的最小单位，一个进程至少拥有一个线程。

由于单进程应用往往需要在保持主干流程不阻塞的情况下处理后台任务，因此需要创建子线程进行后台任务的执行，从而在宏观上表现为并发处理。

2.5.2 非阻塞 I/O 与 I/O 复用

在网络通信以及外存读写等常见 I/O 场景下，linux 提供了 5 种 I/O 模型，分别是阻塞 I/O(Blocking I/O)、非阻塞 I/O(Non-Blocking I/O)、I/O 复用(I/O Multiplexing)、信号驱动的 I/O(Signal Driven I/O)、异步 I/O(Asynchrnous I/O)。

由于 CPU 读写速度与相应的 I/O 设备读写速度不匹配，如果在 I/O 读写上选择常规的阻塞 I/O 模型，往往会造成程序表现为 I/O 状态下失去响应。因此，在一些包含 I/O 读写的程序中会大量使用到非阻塞 I/O，即程序在发出 I/O 请求时会立即返回，而不是阻塞调用等待请求的完成。

在实际应用场景中，程序可能包含多个 I/O 相关的文件描述符，可以采用 I/O 复用模型将多个文件描述符进行集合式关联，从而达到同时操作统一响应的目的。

2.6 有限状态机与模型表述

2.6.1 有限状态机

FSM(Finite-state machine, 有限状态机)又称 FSA(Finite-state automaton, 有限状态自动机)是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。有限状态机拥有有限数量的状态，每个状态可以迁移到零个或多个状态，输入决定执行哪个状态的迁移。有限状态机可以表示为一个有向图，是自动机理论的研究对象[18]。

在有限状态机中，状态存储关于节点的信息，转移表示状态间的变化，用节点间的转移函数来表示节点关系。状态节点包含起始状态和终结状态，状态机在运行时从初始状态开始，逐一读取输入，根据当前状态、当前输入和转移函数来决定状态机的下一个状态。

有限状态机分为确定的有限状态自动机和非确定的有限状态自动机，二者的区别在于确定的有限状态自动机对于每一个可能的输入只有一个状态的转移，而非确定的有限状态自动机对于每一个可能的输入都可以有多个状态转移，接受到输入时从这多个状态转移中非确定的选择一个。

2.6.2 SMV 模型表述

SMV(Symbolic Model Verifier, 符号模型检测)是指将实际中的系统建模为有限状态系统，系统所要满足的性质由时序逻辑进行表示。

SMV 设计了一种语言称为 SMV 语言，可以方便的描述有限状态系统和时序逻辑，处理这种语言的词法和语法分析程序是借助于经典的工具 Yacc 和 Lex 生成的。

2.7 定理证明器概述

2.7.1 定理机器证明

定理机器证明(Automated theorem proving, 简称 ATP)是指把人证明数学定理和日常生活中的演绎推理变成一系列能在计算机上自动实现的符号演算过程的技术。

定理机器证明是人工智能的重要研究领域，它的成果可应用于问题求解、自然语言理解、程序验证和自动程序设计等方面。数学定理证明的过程尽管每一步都很严格有据，但决定采取什么的证明步骤，却依赖于经验、直觉、想象力的洞察力，需要人的思维过程。因此，数学定理的机器证明和其他类型的问题求解，就成为了人工智能研究的起点[19]。

2.7.2 Z3 求解器

Z3 求解器是由微软公司开发的一个优秀的 SMT(Satisfiability modulo theories)求解器，它能够检查一个或多个逻辑表达式的可满足性。Z3 为软件分析和验证工具提供了良好的适配，因为一些常见的软件构造直接能映射到其所支撑的理论中。

2.8 本章总结

本章简要介绍了构建整个物联网服务系统运行时验证系统中所要用到的关键性技术。其中虚拟化技术是整个系统的基础，内核 hook 技术、KVM 串口映射技术、虚拟化监控与内存取证技术是事件采集的关键，多线程与 I/O 复用技术保障事件采集到验证的中间衔接，有限状态机是对验证系统模型的描述，定理证明器技术则用来实现事件的验证分析。以上技术的详细应用，将在后续章节展开详细阐述。

第三章 需求分析

由前述章节内容可知，本文主要以将物联网服务部署在虚拟机为基础，通过内核 hook 技术、KVM 串口映射技术、虚拟机自省技术获取 KVM 中的事件，经过定理证明器的事件验证，最后完成事件反馈，形成高效全面的闭环式物联网运行时验证系统。系统整体不仅支持事件采集的监控展示，还可以实现事件验证结果与事件反馈的实时查看，对 KVM 在运行和对外通信中的非法事件进行检测与告警。同时在系统运行之前，利用可视化界面实现验证模型的绘制、事件模板的配置等。因此本章目的在于确定系统各模块需求，以指导整体设计与实现。

3.1 功能性需求

本文论述的物联网服务系统运行时验证系统，要求在满足基本功能的基础上，还要具备高效性与健壮性。各模块应当能高效实现模块需求，并能处理各类异常情况。整个运行时验证系统的用例图如图 3-1 所示：

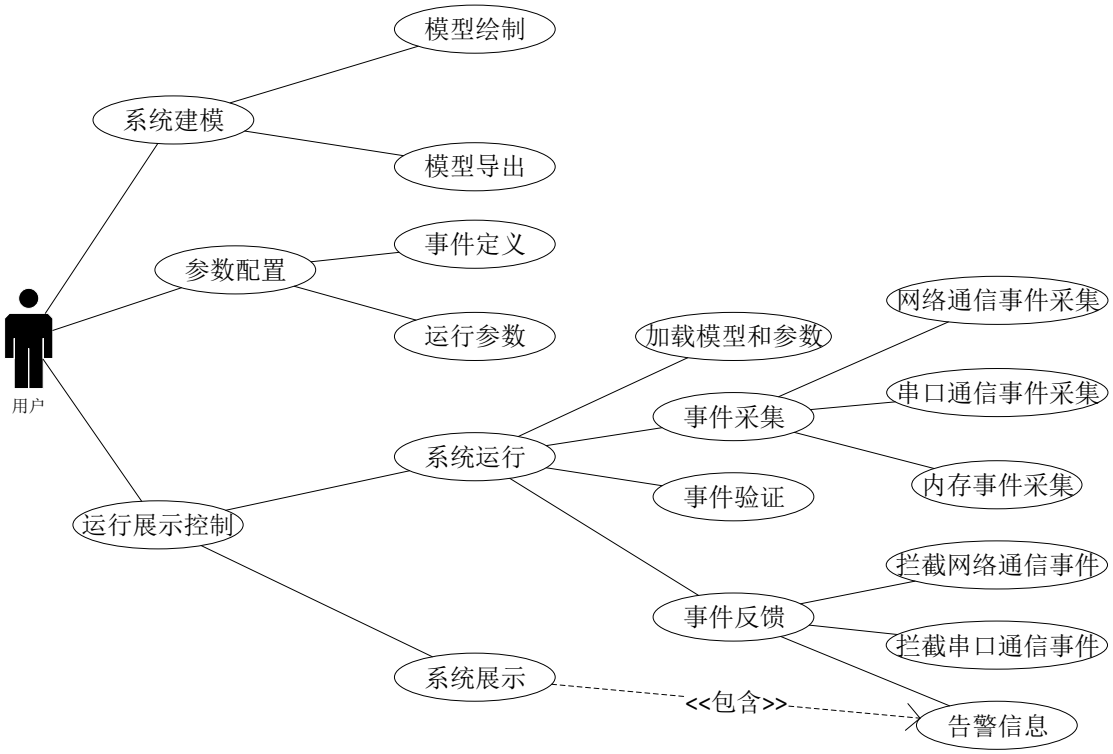

图 3-1 系统整体用例图

3.1.1 事件采集

3.1.1.1 网络通信事件采集

物联网服务系统需要与外部设备进行频繁网络通信来实现对物联网设备的控制，因此网络通信是物联网服务系统自身功能实现的一大重点。网络通信事件的采集能力代表着对整个物联网服务系统的事件采集能力。

鉴于耦合性与安全性的考虑，物联网服务系统需要使用虚拟化技术搭建在KVM 之中，网络通信事件的采集需要部署在宿主机之中，并且不应对 KVM 内部产生依赖。

KVM 的网络模式以桥接方式为主，因此 KVM 会获得与宿主机同等网段的IP 地址。但 KVM 和宿主机对外进行网络通信的数据会流经同一物理网卡，所以在进行网络通信事件采集时要对 KVM 和宿主机的数据进行严格区分。

物联网服务系统与外界通信主要使用传输层的 TCP 协议和 UDP 协议，由于本文所述运行时验证系统将会在事件采集验证后进行反馈拦截，而 TCP 协议是可靠的协议，自身存在超时重传机制，因此若对 TCP 协议的 IP 数据报进行拦截将导致通信链路中断。本文主要针对 UDP 协议进行事件采集和反馈拦截。此外，UDP 报文过长时，其所在的 IP 数据报将可能超过通信链路上数据链路层的MTU(Maximum Transmission Unit, 最大传输单元)，从而导致 IP 数据报分片，使得 UDP 报文无法在一次采集中完整获取。由于实际通信过程中，UDP 过长产生的 IP 数据报分片将使网络通信效率降低，所以绝大多数 UDP 报文被人为控制在一定长度以内。本文主要针对无 IP 数据报分片行为下的 UDP 协议内事件采集。

网络通信事件的采集本身干扰了原有的正常通信过程，因此在事件采集过程中，要求仅处理需要处理的网络通信数据，对不相关的数据做到无任何修改、丢失。同时整个事件采集的过程要做到高效快速，最大程度降低因事件采集造成的网络带宽下降。

在稳定性方面，由于网络通信事件采集涉及到 linux 内核底层，因此整个模块要求极高的健壮性，对可能的异常情况做相应的防御性异常处理，以防止运行中的异常造成操作系统的崩溃。

网络通信事件采集后，需要将事件发送到用户态进行事件验证。所有事件将根据事件属性分为关键事件和非关键事件，其中仅关键事件需要等待事件验证结果进行事件反馈，非关键事件将直接通过。由于此模块将在宿主机内核中运行，等待事件验证结果这一过程需要有等待超时上限时间，以防通信链路陷入阻塞。

表 3-1 网络通信事件采集功能需求表

需求名称	需求内容	需求说明
KVM 网络流量采集	数据捕获	捕获流经物理网卡的所有网络通信数据
	数据区分	筛选出 KVM 与外界通信的数据
	协议解析	依次解析 IP 协议、UDP 协议提取应用层数据
	事件析取	从应用层中解析出事件
	结果等待	仅关键事件需等待事件验证结果，等待有超时机制
	低依赖性	不用对 KVM 内部产生依赖
	高效性	对 KVM 对外通信过程影响要小
	稳定性	保障宿主机操作系统的稳定性
采集事件发送	通信可靠	用户态与内核态通信需要保证中间不丢失数据
	及时反馈	内核态收到处理结果后要及时处理

3.1.1.2 串口通信事件采集

串口是一种独占设备，即任何时候无论是虚拟机还是宿主机，均只有其中一个能够独占此设备进行通信。因此，该模块需要借助于伪终端等技术实现串口映射，将映射为 KVM 内部串口的伪终端与物理串口进行连接，实现无缝的串口中继转发功能。

该模块首先要求能对两端波特率、停止位、奇偶位等串口参数进行协调统一，其次要能够实现串口中继过程中数据的零丢失和零异常。

由于串口传递的是无状态的字节流信息，因此在串口通信事件采集过程中，对于非串口事件部分的字节流信息要及时转发，提高串口通信的及时性。

同时，该模块连接的两端均为 I/O 设备，要满足信道的全双工通信，需要在模块内部使用非阻塞 I/O 模型，实现串口通信的高效性。

串口通信事件采集后，需要进行事件验证。其中仅关键事件需要等待事件验证结果进行事件反馈，非关键事件直接转发。

表 3-2 串口通信事件采集功能需求表

需求名称	需求内容	需求说明
串口中继	端口衔接	连接两端设备，协调通信协议
	数据转发	转发串口两端的数据到对端，全双工通信
	高效性	对整体通信效率影响较小，数据及时转发
	可靠性	数据转发过程中无遗漏和重复
串口事件采集	事件析取	从串口中继器数据缓冲区中解析事件
	结果等待	仅关键事件需要等待事件验证结果

3.1.1.3 内存事件采集

物联网服务系统在与外界通信时，其所在的线程栈总会在运行时保留其当前栈上的变量信息。该模块要求在 KVM 运行中在宿主机进行虚拟机自省，获取物联网服务系统运行时的关键信息构成事件，完成内存事件采集的过程。

由于内存事件并不是物联网服务系统本身产生，而是由宿主机中运行的虚拟机自省程序获取生成的，因此对整个内存事件采集的高效性有较高的要求。

此外，内存事件采集的过程中，需要保证虚拟机自省程序对 KVM 内部不造成任何影响，以满足整个运行时验证系统的隔离性。

内存事件采集后，进行相应的事件验证，无需等待事件验证结果。

表 3-3 内存事件采集功能需求表

需求名称	需求内容	需求说明
虚拟机自省	KVM 内存访问	从宿主机中访问 KVM 中指定进程的内存
	JVM 解析	解析进程 JVM 中的数据结构
	事件构造	根据 JVM 中数据构造事件
内存事件发送	通信可靠	事件发送过程需要无遗漏与重复

3.1.2 系统建模与可视化模型绘制

用户需要对物联网服务系统在运行中预期的运行状态进行定制，状态与状态之间的变化也要通过相应的事件来推动。每一个状态中将有相应的状态表达式，用于表示此状态的约束信息。此外，由事件推动产生的状态变化轨迹可以产生更多轨迹上的约束，即状态本身的约束用于单次状态转移的判断，状态与状态间的时序上约束用于实际运行中状态变化轨迹的合法性判断。

系统的建模是根据 SMV 语言的规范来进行的，用户对系统建模需要有相应的图形化表示，即用户需要能够简单方便地对物联网服务系统运行中运行状态的表示、状态间的关系、以及状态变化轨迹的时序上约束进行语义表达，从而完成对整个系统建模的过程。

运行时验证系统需要对事件进行相应的验证处理，验证过程需要代入到既定的模型之中。因此在运行整个系统之前，需要一个可视化界面用以绘制相应的模型。

为了方便用户的操作，同时提高系统的扩展性，该模块需要在可视化界面中提供图元与图元绘制界面以表示模型。用户只需要在本地或者远程访问该可视化界面，进行简单地图元拖拽、内容输入等操作即可绘制整个模型。

该模块需要提供可视化界面的序列化表示，以供验证系统的读取解析。同时序列化表示需要实现界面表示的较高还原性与可靠性。

表 3-4 系统建模与可视化模型绘制需求表

需求名称	需求内容	需求说明
系统建模	SMV 规范表述	建立符合 SMV 规范的有限状态系统
	时序逻辑约束	需要对有限状态系统添加时序逻辑的约束
模型绘制	可拖动图元	可简单操作即可绘制模型的用户界面
	图元序列化	可将绘制的模型导出成规范文本

3.1.3 事件验证

通过各个事件采集渠道采集而来的关键事件需要通过此模块进行事件的验证。

该模块首先要将预先输入的模型进行预处理，生成相应的模型结构与验证公式。当采集到的事件输入到该模块时，需要对事件进行解析，提取其中的变量信息，然后代入模型进行验证分析。事件验证后将会产生验证通过与验证未通过两种结果。

事件的验证需要融合三个观察源的事件，形成互相补偿的作用。事件在验证过程中需要进行相应的状态机节点转移，首先要满足事件转移是符合状态机规范的，其次由于在物联网系统实际运行中从事件通信到事件采集均可能出现事件重复与遗漏的情况，因此事件转移需要对重复与遗漏有所应对。最后事件转移需要对时序上的约束进行验证，以保障事件的验证结果是符号模型需求的。

由于整个被监控系统是处于运行状态，事件的采集速度与事件验证速度可能出现较大的不匹配，从而导致事件验证成为系统瓶颈，降低了事件采集甚至物联网服务系统对外通信的带宽。因此，事件验证的及时性与高效性将是整个运行时验证系统的重要环节。

此外，事件验证模块要达到高度的验证准确性，以保障系统在对事件处理上可能的误判过程。

表 3-5 事件验证需求表

需求名称	需求内容	需求说明
事件验证	事件转移	按有限状态机规范进行转移
	事件重复与遗漏应对	需要在常规无法转移时在全局进行跳跃转移
	时序约束验证	需要事件转移后满足时序上的约束
	高效性	事件验证需有较高的运行效率

3.1.4 事件反馈

3.1.4.1 非法事件告警

从三个事件采集源采集到的事件经过事件验证之后，将会得到验证通过与未通过两种结果。用户需要能够随时获取这些处理结果，同时对未通过的情况要进行相应的警告输出作出异常情况的提醒。

3.1.4.2 网络与串口事件反馈拦截

网络通信事件采集与串口通信事件采集模块均要求不仅要有事件采集能力，同时要能对事件进行相应的拦截丢弃处理。

对于网络通信事件采集模块，要求该模块可接收来自事件验证结果的反馈。如事件验证通过则网络通信事件采集模块对该采集到的事件进行网络上的放行通过，或者事件验证未通过时则网络通信事件采集模块对该采集到的事件进行网络上的丢弃处理。

对于串口通信事件采集模块，要求该模块能根据事件验证的结果对该事件进行转发或者不转发的相应操作，达到事件的拦截丢弃效果。

此外，对于内存事件采集模块，由于 KVM 无法直接修改内部操作系统的数据结构，因此内存事件无法实现反馈拦截，仅可进行告警处理。

表 3-6 事件反馈需求表

需求名称	需求内容	需求说明
非法事件告警	告警到用户界面	将运行中出现的非法事件通过界面告知用户
非法关键事件拦截	网络通信事件拦截	对网络通信中非法关键事件进行丢弃
	串口通信事件拦截	对串口通信中非法关键事件不转发处理

3.1.5 界面系统

界面系统是针对用户设计与实现的。其目的是简化用户操作，降低系统操作门槛，使得用户只需要通过界面系统进行简单地操作即可以对物联网服务系统运行时验证系统进行操作与查看。界面系统包括建模过程中的可视化图元绘制界面、事件参数等定制界面和控制展示界面，具体需求如下。

3.1.5.1 事件与配置输入

界面系统将由用户输入事件采集的标准、KVM 的相关信息等，因此需要相应的输入框供用户输入信息。

事件采集的标准包括事件的匹配头和匹配尾。KVM 的相关信息包括 KVM 的 IP 地址、KVM 与外界通信的设备 IP 地址、映射成 KVM 串口的伪终端路径、物理串口的路径等信息。

3.1.5.2 控制与运行展示

用户需要在此界面以简单地点击操作实现整个系统的开始运行和结束运行。同时，点击开始运行后，运行过程应放在后台进行处理，界面不应被后台处理任务阻塞导致影响用户体验。

当用户想要查看一些系统异常和事件拦截等关键信息时，如果直接去查找历史日志，将会非常麻烦。因此，当一些重要日志产生时，需要及时反映到此界面供用户查看，方便用户即时知晓物联网服务系统可能出现的一些异常情况，进而进行异常排查等后续操作，为以后的保障与扩展改进做更多的准备。

表 3-7 界面系统需求表

需求名称	需求内容	需求说明
事件与配置输入	可定制事件格式	可对事件析取规范进行定制
	参数输入	可输入事件采集各模块的参数
控制与运行展示	快速可控的系统运行与结束	对系统的开始运行和结束运行需要快速可控
	运行展示	对系统中关键信息进行用户界面的输出

3.2 非功能性需求

除以上文所述功能性需求之外，高可靠性和高性能也是物联网服务系统运行时验证时必不可少的重要要求。

3.2.1 高可靠性

物联网服务系统本身需要高度的稳定性以保障其长期投入生产运行，而对其进行的运行时验证系统同样需要自身具备高可靠性。一方面需要运行时验证系统能长时间运行不出现异常情况，各模块内部具有较高的健壮性，能够抵挡运行中出现的各种异常情况。另一方面运行时验证系统在物联网服务系统进行事件采集和事件反馈的过程中不会因为异常情况而影响到物联网服务系统自身的正常运行。因此，对整个系统反复的测试和调优，增加系统的稳定性与可靠性是运行时验证系统的核心目标之一。

3.2.2 高性能

物联网服务系统运行时验证系统各模块衔接较为紧密，同时事件采集与事件反馈对通信链路存在着一定的干扰，因此各模块的执行效率将影响着整个运行时验证系统的效率。各模块的性能表现需要一定的匹配程度，同时各模块的连接部分需要较高的性能以提高整体表现。

事件验证将是在事件采集后最为核心的一块，而这部分也将是预期最为耗时的部分。在大流量、高并发的网络环境下，我们需要对事件验证过程进行高度优化与改进，以满足整体高性能的要求。

3.3 本章总结

本章从物联网服务系统运行时验证的思路出发，将功能进行分割细化，详细阐述了各模块自身需要满足哪些要求以及各模块需要怎样联合实现整个系统的功能。在功能性需求方面，主要包括网络通信、串口通信和内存三个方面的事件采集、事件验证、事件反馈以及高用户体验的界面系统；在非功能性需求方面，主要包括维持运行时验证系统和物联网服务系统平稳运行的高可靠性以及降低运行时验证系统带来影响的高性能表现。

第四章 系统概要设计

第三章从功能性和非功能性两方面详细描述了一个完整的物联网服务系统运行时验证系统的需求和目标。本章将对这些需求和目标进行一一分解，从系统整体架构出发，详细阐述每一个模块功能的设计思想和实现方式，再从功能流程出发，详细介绍每一个流程完成的过程和模块接合关系。

4.1 系统架构

本文所述运行时验证系统包括事件多观察源采集、事件融合验证、事件反馈三个主要部分，集高可靠性、高性能、隔离性于一体。图 4-1 为整体系统架构图。由图可知，物联网服务系统部署在隔离的虚拟机中，其与外界物联网设备的交互包括网络通信与串口通信两部分。运行时验证系统部署在宿主机中，对物联网服务系统的通信通道进行管控。各观察源以不同角度从物联网服务系统中析取事件，最大化的还原运行状态。析取出的事件交由运行时验证系统进行融合验证，采用事件重复与遗漏应对算法并结合定理证明器进行求解。最后对物联网服务系统的通信通道进行相应的反馈，达到高隔离性下对违法策略规定事件的拦截。

具体到各个模块设计上，网络通信事件采集模块运行在宿主机内核中，能够获取 KVM 与外界通信的所有数据包，通过既定的事件模板对事件进行采集，然后发送到事件验证模块。串口通信事件采集模块包含有一个完善的串口中继器，将宿主机的物理串口与映射成 KVM 内部物理串口的伪终端连接起来，同样通过既定的事件模板对事件进行采集，同时肩负内外部通信连接功能，然后将事件发送到事件验证模块。内存事件采集模块通过虚拟机自省技术从 KVM 内部物联网服务系统进程中线程栈上获取变量信息，然后根据既定的事件模板构造事件发送到事件验证模块。事件验证模块轮询收集三个来源的事件，对事件进行逐一验证，然后将关键事件的验证结果进行事件反馈。各个来源对事件反馈的表现不一，其中网络通信事件采集模块对事件反馈表现为数据包通过或者丢弃，串口通信事件采集模块表现为数据包的转发或者丢弃，内存事件采集模块表现为向用户发出警告。

系统的架构设计要保证各模块衔接合理，将模块的职责划分清楚，提高模块的内聚性，降低模块间耦合性。同时系统架构要兼顾模块衔接导致的额外系统开销和为降低系统开销将模块过度耦合带来的扩展性问题。

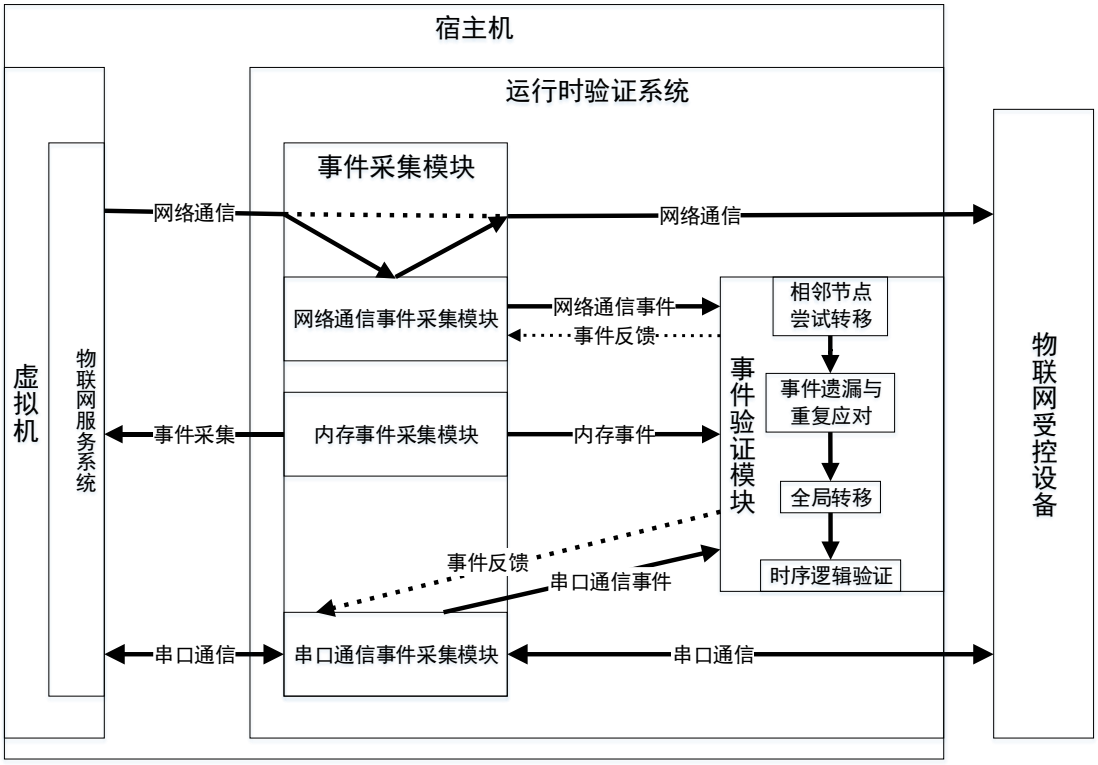

图 4-1 整体系统架构

4.1.1 网络通信事件采集

网络通信事件采集模块首先需要满足隔离性，即不能在 KVM 内部部署相应的网络流量捕获程序。由于 KVM 在对外进行网络通信时，虽然其拥有由宿主机虚拟出来的虚拟网卡，但其通信数据包总会经过宿主机的物理网卡。因此，在宿主机的网络底层部署相应的程序分析流经宿主机物理网卡的数据包，必然可以获取 KVM 对外通信的全部数据。

在第二章介绍中已经提到，本文主要研究 KVM 在桥接模式下的网络通信事件采集。普通的宿主机对外通信时使用的网络连接是以太网或者无线局域网等直接使用物理网卡的连接方式。当 KVM 需要设置成桥接模式时，首先要更改宿主机的网络连接方式。即新建一个桥接连接，将宿主机原来的以太网或者无线局域网等连接桥接到这个新建的桥接连接上，此时桥接连接获得宿主机所在网段的 IP 地址。然后通过本文所采用的 VMM virt-manager 配置 KVM 的网络连接模式为桥接到宿主机的桥接连接上。

为了在宿主机内即可采集 KVM 内网络通信数据，需要在 KVM 的数据包流经宿主机网络底层桥接处理之前进行捕获。因此 Netfilter 内核模块的 hook 点需要选取的尽量靠近网络数据流经起点。由于桥接处理是在网络协议层的数据链路层进行，Netfilter 内核模块需要 hook 到数据链路层的 6 个挂载点上，本文选择 6 个挂载点的第一个挂载点 pre_routing 进行挂载。

Netfilter 在 hook 到相应挂载点会在运行中有数据流经挂载点时触发相应的回调函数，在回调函数中可根据内核数据结构获取该次数据包中的 IP 数据报。原始的 IP 数据报需要经过多重筛选才能获得实际需要处理的数据。基于高性能要求，我们将 IP 数据报的筛选由以下几个步骤先后进行[20]：

1. 判断是否有用户态客户端已经连接到内核的 Netfilter 模块，如果没有则直接通过 IP 数据报。

2. 将无效的和空的 IP 数据报直接通过。

3. 解析以太网报文类型，将非发往其他主机的数据直接通过。

4. 解析 IP 数据报首部，获取源 IP 地址和目的 IP 地址，与预先配置要求的源 IP 地址和目的 IP 地址相比较，如果有任何一个不相同，直接通过此 IP 数据报。

5. 从 IP 数据报首部获取传输层协议，如果是非 UDP 协议，则直接通过此 IP 数据报。

6. 定位到 UDP 报文 body 部分，对预选配置的事件匹配头和事件匹配尾在 body 部分中进行匹配，如果匹配失败，则直接通过此 IP 数据报。

网络通信数据包筛选流程如图 4-2 所示：

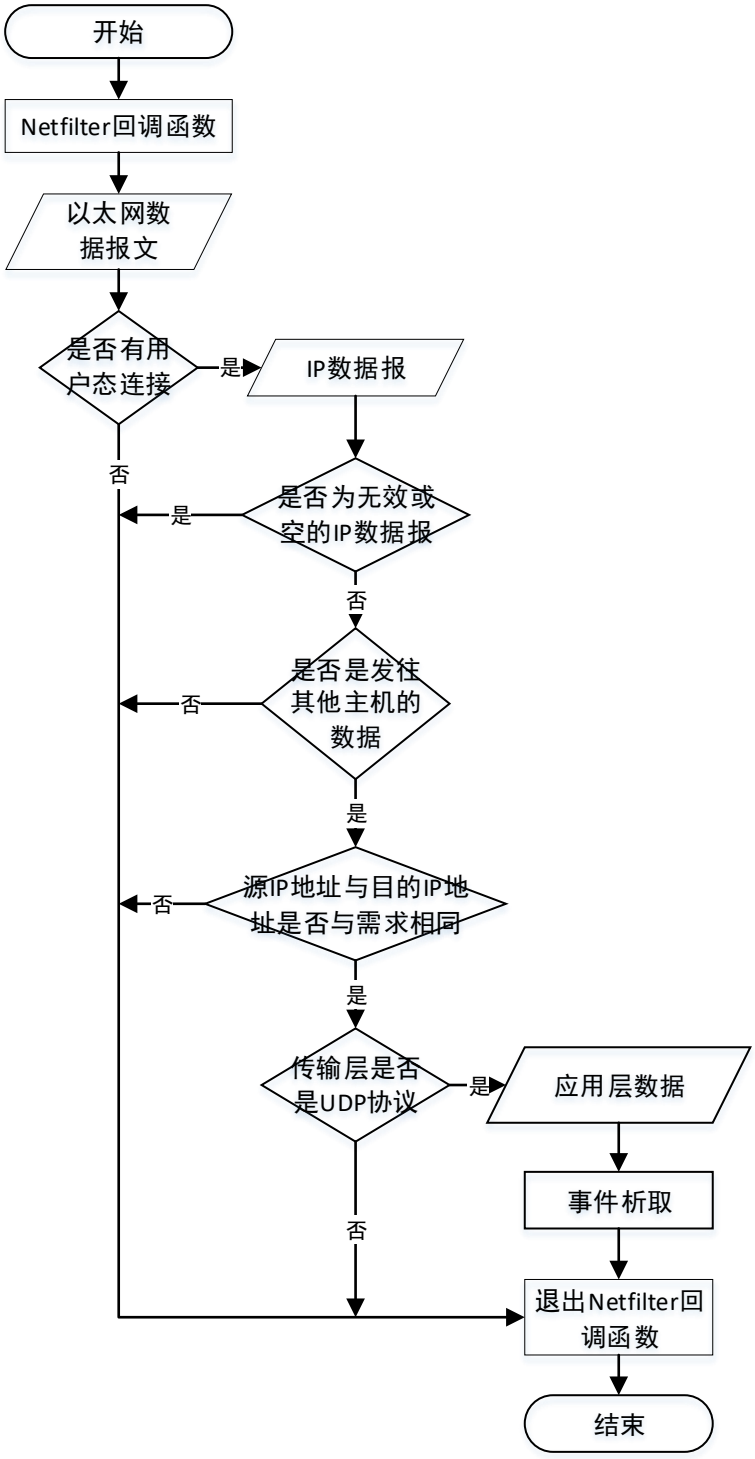

图 4-2 网络通信数据包筛选流程

经过以上几个步骤筛选之后，此时在该 IP 数据报已经能够确定匹配的事件数据部分。然后确定该事件是否为关键事件，并将该事件使用 Netlink 发送到用户态事件验证模块。如果该事件不是关键事件，则 Netfilter 直接将该 IP 数据报通过。如果该事件是关键事件，则 Netfilter 在此对用户态的事件验证结果进行超时等待。只要在既定时间内等到了事件验证的结果并且结果为拦截该事件，则 Netfilter 将该 IP 数据报丢弃。网络事件析取与处理的流程如图 4-3 所示：

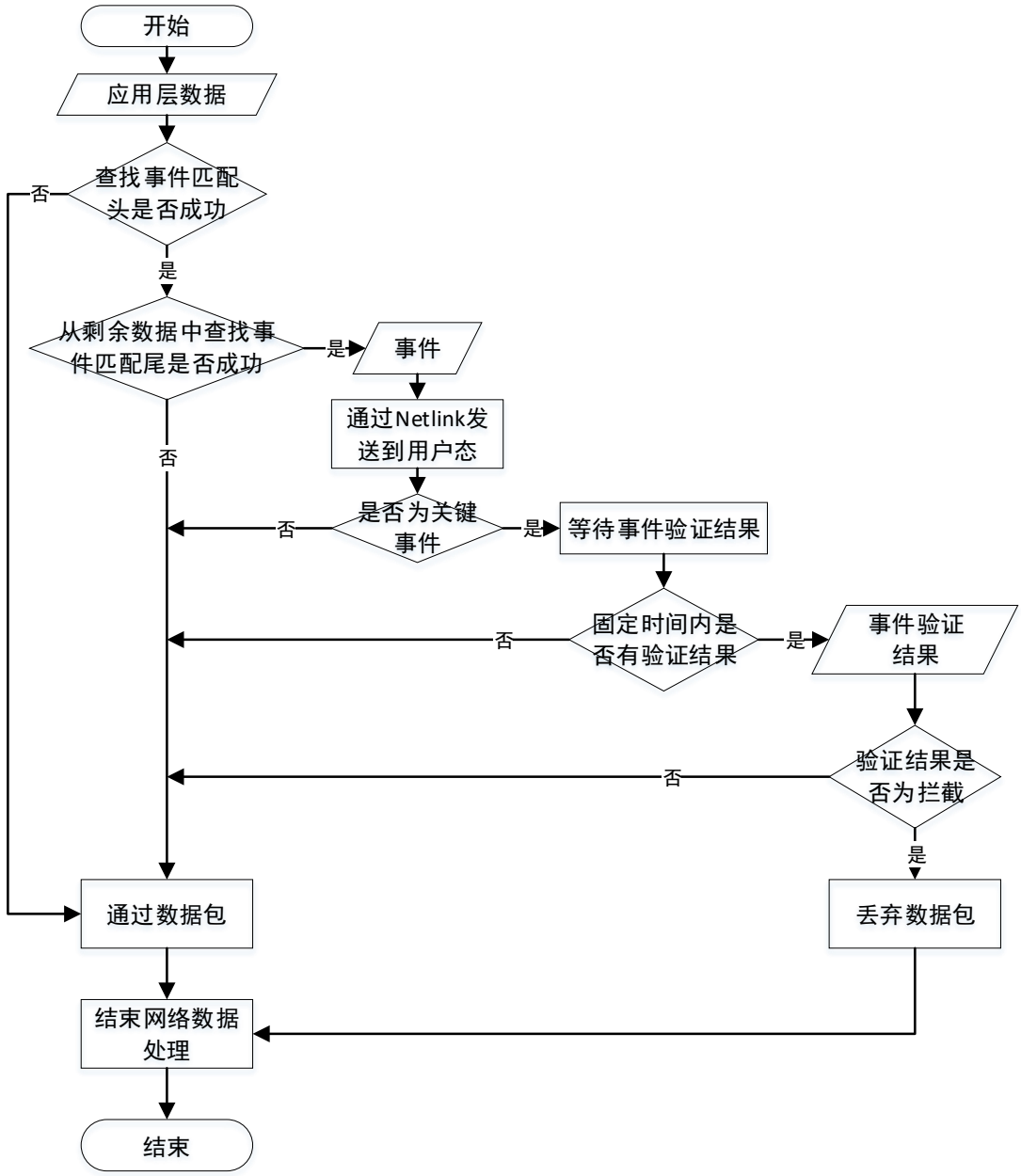

图 4-3 网络事件析取与处理流程

4.1.2 串口通信事件采集

串口是一种独占设备，如果按照传统的方式让宿主机的物理串口直接映射成 KVM 内部的物理串口，那么 KVM 在使用串口与外界进行通信时，宿主机将无法同时使用自身的该物理串口。此时无法对串口通信事件进行采集。

借助于 linux 伪终端技术，改变 KVM 的串口映射方式，将宿主机的一个伪终端映射成 KVM 内部的串口。同时在宿主机中运行串口中继程序，将该伪终端与物理串口连接，对两端通信的数据进行转发，这样在对 KVM 内部和外部设备无感知的情况下就可以实现串口数据的采集与拦截[21]。

由于串口通信大多数是全双工通信，即使是半双工的情况下也要使串口中继器满足两端互发数据均能正确转发到对端。为了使串口中继器同时保障两条相对的通信链路，需要在内部使用 I/O 复用模型，将两端的文件描述符添加到集合中轮询处理。无论一端是否有发往另一端的数据，串口中继器均在一个最小周期内查询该端是否有数据，并进行超时等待。如果在超时未完成时等待到数据，即可处理该部分数据，然后再查询另一端是否有数据进行同样的处理。

串口通信传输的是字节流信息，实际应用中往往会抽象出高层的串口协议，或者直接使用字节流传输相应的格式化数据。本文主要针对串口通信的字节流信息中直接包含事件字符串的情况进行研究，如果要应用于具体的串口协议，只需更改此模块对于字节流信息提取事件的部分即可完成模块功能扩展。

整个串口通信事件采集模块如图 4-4 所示：

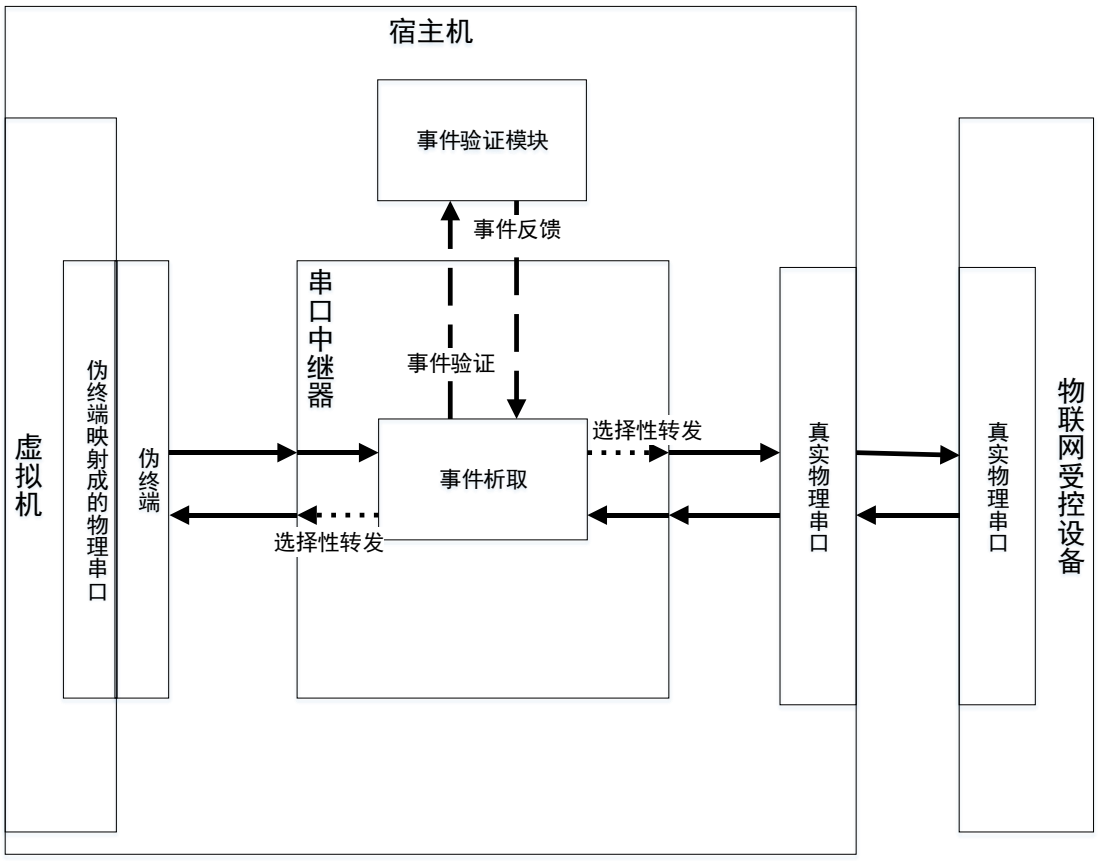

图 4-4 串口中继与事件采集结构图

对于串口中继器在两端获取的字节流，从中依次匹配预先配置的事件匹配头和事件匹配尾，确定事件的边界，将采集到的该事件交由事件验证模块进行验证处理。 同时对于关键事件需要等待事件验证模块的处理结果，将处理结果为拦截的事件进行丢弃，即在字节流中对应的字符串作丢弃不转发处理。

串口通信上事件析取与处理的流程与图 4-5 所示：

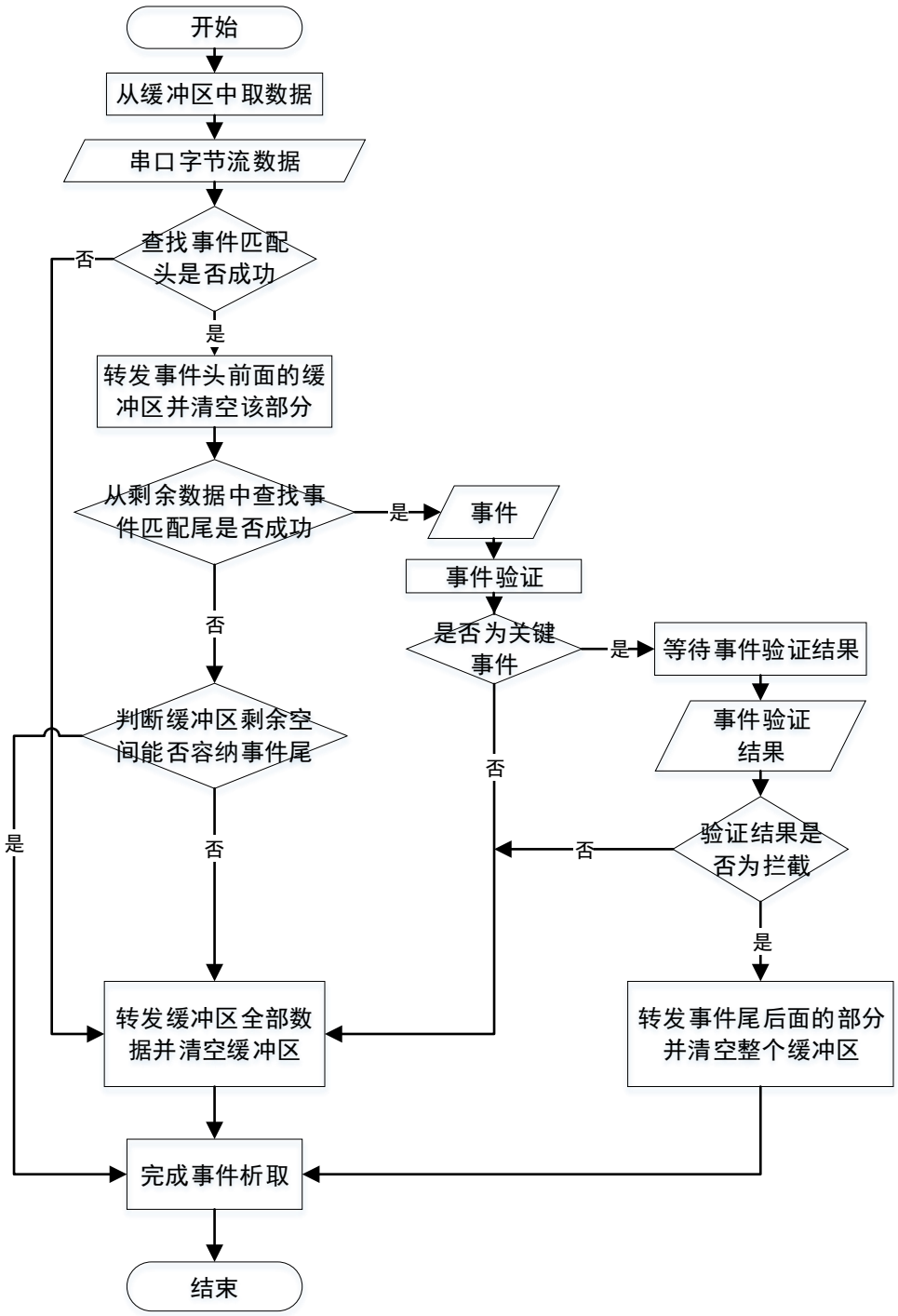

图 4-5 串口事件析取与处理流程

4.1.3 内存事件采集

内存事件采集是对运行中的 KVM 进行虚拟机自省以获取程序栈上变量信息从而构造事件的过程。KVM 内存的读取由 LibVMI 实现，内存中操作系统数据结构的重构由 Volatility 实现，将两者进行结合，即可在 KVM 运行中获取 KVM 内操作系统数据结构，从而分析得到进程的线程栈上变量信息。

由于现阶段大多数物联网服务系统程序使用 Java 进行开发，而 Java 在运行中由 JVM(Java Virtual Machine, Java 虚拟机)进行解释执行，所以程序在运行中一方面由 JVM 整体对操作系统表现成一个进程，另一方面 Java 程序在解释执行中使用的函数调用栈并非操作系统原生的栈，而是由 JVM 生成并管理的栈。这样就造成分析 Java 程序需要进一步根据 JVM 原理拆开 JVM 进程数据结构来获取实际执行的 Java 程序栈的信息[22]。

该部分功能由作者所在实验室实现，以模块形式经网络通信提供给整个运行时验证系统调用，本文将不对这部分功能实现细节作详细阐述。

4.1.4 系统状态机建模

物联网服务系统往往用于对物联网设备的远程控制，而在整个系统设计和使用时，这些控制过程中的控制量均有相应的标准与范围进行参考。同时人们又希望这些控制量有一定的约束，以防非法的控制指令给物联网设备造成预期外的负面影响。

我们将物联网服务系统中控制量提取出来形成具体的变量，变量作为系统的主要描述表示的是系统的特征。系统在运行过程中由一个或多个变量共同表示其运行状态。

物联网服务系统在运行中可以根据变量组的不同特征形成不同的运行状态，通过人为的划分运行状态，可以建立不同的状态节点。状态节点上以一组或多组变量的范围约束来表述该节点上的运行状态。

物联网服务系统在接收用户的控制量时会造成自身运行状态的转变，该转变可能是状态节点内部的变动，也可能推动状态节点从一个节点跳转至另一个节点，我们将由控制量变化指令产生的状态节点之间的变更称为转移。

由状态节点和转移构成的完整模型便形成一个有限状态自动机，该自动机能够描述物联网服务系统中的全部运行状态。同时参考 SMV 标准语法，对有限状态自动机表示的模型进行规范化，如加入起始节点与终止节点标记。

描述系统的表现之后，将预期希望的对系统添加的约束以时序逻辑的形式添加到系统中。时序逻辑描述的是系统在时间线上状态与状态之间的变化所需要的约束关系，体现的是用户预期的安全策略以及安全目标。

将有限状态自动机与时序逻辑相结合，基于 SMV 规范的系统便完成建模过程。

4.1.5 事件转移验证

传统的有限状态自动机分为确定的有限状态自动机和非确定的有限状态自动机，其中确定的有限状态自动机对于每一个可能的输入只有一个确定的转移。如果我们将物联网服务系统实际运行中产生的包含变量的事件作为输出，那么该

事件上则需要有一种标识与有限状态自动机上的转移进行对应，我们称之为事件名称，同时也可称为转移名称。事件转移验证如图 4-6 所示：

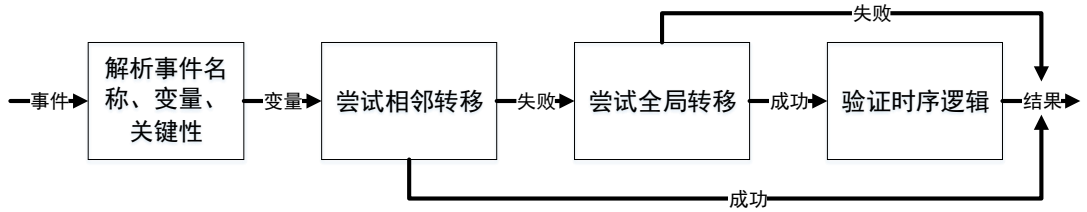

图 4-6 事件转移验证结构图

当采集模块采集到具体的事件时，将该事件加入状态机进行事件转移。状态机从起始状态出发，我们遍历当前状态出出的转移，判断该转移名称是否与事件名称一致。同时，在转移名称与事件名称保持一致的基础上，事件中具体的变量信息能否满足待定的下一状态的状态上约束也是需要考虑的问题。因此，需要额外对事件上变量与下一状态上的约束表达式进行联合验证。整体算法流程如算法 4-1 所示：

算法 4-1 单个事件直接转移算法 EVENTTRANSITION

输入：*Event* 单个事件，*State* 当前节点，*Solver* 求解器

输出：转移是否成功

```
1:  function EVENTTRANSITION(Event, State, Solver)
2:      for outTran in State.outTranList do
3:          if outTran.name != Event.name then
4:              continue
5:          end if
6:          nextState = outTran.destinationState
7:          for expression in nextState.expressions do
8:              Solver.add(expression)
9:          end for
10:         for variate, value in Event.variates do
11:             Solver.add(variate == value)
12:         end for
13:         if Solver.check() then
14:            return True
15:         end if
16:     end for
17:     return False
18: end function
```

物联网服务系统主要通过网络和串口与外界进行通信，实际运行中可能因为通信过程的异常等原因造成控制指令的部分丢失。此外，运行时验证系统在进行事件采集时也可能因为自身运行性能等问题发生事件采集遗漏、事件采集重复等异常情况。因此，在进行事件转移时要应对事件可能出现的丢失和重复，即如果当前节点不能根据当前事件进行合法转移，要在状态机全图中进行状态转移尝试。

当事件输入到状态机后，首先根据算法 4-1 所述进行简单的事件转移尝试。如果事件无法从当前状态转移到相邻状态，则有可能出现事件的丢失或者重复。此时，我们在状态机全局中查找与事件名称相同的转移名称，一方面对事件名称匹配，另一方面采用同样的验证方式将下一状态上的约束表达式与事件上变量进行联合验证。

此外，为了进一步提高执行效率，首先针对全局匹配中可能的一个状态被多个转移名称相同的转移所指向的情况，我们需要对全局匹配中尝试转移失败的状

态进行记录，以防止不必要的重复尝试。其次事件在全局匹配中可能出现多次失败，每当因匹配中表达式验证失败时，需要将该次尝试验证中添加的表达式全部弹出，而不是将求解器重置。如果事件转移成功，则新加入的表达式则无需弹出。

依此改进后的整体算法流程如算法 4-2 所示：

算法 4-2 单个事件遗漏与重复应对转移算法 EventImprovedTransition

输入：Event 单个事件，State 当前节点，Solver 求解器，Module 整个模型

输出：转移是否成功

```
1:  function EventImprovedTransition(Event, State, Solver, Module)
2:      statesVisitedSet ← empty
3:      if EventTransition(Event, State, Solver) then
4:          return True
5:      end if
6:      for tran in Module.trans do
7:          if tran.name != Event.name then
8:              continue
9:          end if
10:         nextState = tran.destinationState
11:         if nextState in statesVisitedSet then
12:             continue
13:         end if
14:         Solver.push()
15:         for expression in nextState.expressions do
16:             Solver.add(expression)
17:         end for
18:         for variate, value in Event.variates do
19:             Solver.add(variate == value)
20:         end for
21:         if Solver.check() then
22:             return true
23:         end if
24:         statesVisitedSet.add(nextState)
25:         Solver.pop()
26:     end for
27:     return False
28: end function
```

在物联网服务系统的实际应用中，状态机节点和其节点间的转移关系仅能表示该物联网服务系统自身的特点，用户在对系统运行时往往希望能有更多时序上的节点间关系间约束。

节点之间由转移进行相互联系，而在实际运行中，转移的发生由具体的事件进行推动。事件中包含有相应的变量具体值，因此转移前后的节点可以有额外的变量关系约束。

节点间时序上的额外约束是发生在事件转移的过程中，随着事件的不断到达，可以将事件历史转移产生的节点变化情况称为节点转移轨迹，或者称之为事件轨迹。对事件轨迹上的状态节点间添加的额外约束即为时序上的约束。

事件的转移成功与否在使用算法 4-2 之后还要对节点间时序上的约束进行考虑，只有当事件可以转移且该次转移能够满足节点间时序约束，整个事件转移过程才能算作转移成功。整体算法流程如算法 4-3 所示：

算法 4-3 含节点间时序约束验证的单个事件遗漏与重复应对转移算法 EventVerificationTransition

输入：*Event* 单个事件，*State* 当前节点，*Solver* 求解器，*Module* 整个模型，*Specs* 时序验证表达式

输出：转移是否成功

```
1:  function EventVerificationTransition(Event, State, Solver, Module, Specs)
2:      for spec in Specs do
3:          Solver.add(spec)
4:      end for
5:      if EventImprovedTransition(Event, State, Solver, Module) then
6:          return True
7:      else
8:          return False
9:      end if
10: end function
```

除去事件遗漏与重复外，实际物联网服务系统在并发运行时还可能出现事件的失序情况。当后序事件在前序事件未被采集到时而先被采集到输入到验证系统中，可能出现时序验证逻辑的可求解而通过该事件。这种情况下，后序事件可能是非法的，但却被系统所接收。针对事件的失序，需要联合多个事件来源进行合并分析。

首先，验证系统在验证事件时仅验证了当前时序逻辑是否可满足，而实际上如果要事件验证后达到严格通过，则不仅要使当前时序逻辑满足，同时要使时序

逻辑取反后达到不可满足。如果仅能使当前时序逻辑满足，而在时序逻辑取反后仍然满足，说明该事件是可通过也可不通过的未知状态。因此，将时序逻辑与取反后的时序逻辑同时验证事件，可以根除前述后序事件可能非法，但因为失序后先被采集而通过验证系统的情况。

其次，当事件被验证为未知状态时，说明该事件可能是失序事件，此时验证系统记录事件并在固定时间内等待采集系统输入下一条采集的事件。如果在等待时间内验证系统能得到明确的通过或不通过结论，那么其记录的事件便可得到相应的处理结果。对于采集系统而言，当其从某个事件来源获取事件并输入到验证系统后，如果得到的返回结果为未知状态，则开始超时等待并在该时间内继续从其他来源获取事件，直到超时或者从验证系统获取时间段内所有事件处理结果。

4.1.6 非法事件告警

事件经过转移验证之后，将验证结果进行相应的输出展示。其中能够合法通过验证的事件预期占事件总体比例较大，因此无需输出。对于验证失败的事件，根据其是否为关键事件进行不同方式的输出，这样可以对用户提供较为明显的告警方式。

4.1.7 网络事件反馈拦截

当网络通信事件中的关键事件验证结果为未通过时，需要将验证结果通知回网络通信事件采集模块进行事件拦截。采集模块运行在 linux 内核中，而事件验证模块运行在用户态中。因此在用户态根据获取内核态采集事件建立的 Netlink 通信链路将该次关键事件的验证结果发回到内核态。

由于网络通信事件采集模块自身健壮性的需求，其在内核态运行时将关键事件发送到用户态验证时不能进行阻塞盲等，而是有等待时间上限。因此当用户态的事件验证结果发送到内核态时，可能出现内核态对该事件的结果等待已经超时，并且已经开启对下一个关键事件的超时等待。在这样的情况下，事件反馈将出现错序异常。为解决此问题，内核态在收到事件验证结果时需要对事件进行校验。

原则上对事件的校验需要用户态将事件验证结果与原事件一并发送回内核态进行事件比对，或者采用内核态计算事件摘要码发送到用户态，用户态在事件验证完成后将此摘要码随同事件验证结构发回内核态作为事件校验的方法。但在网络通信事件采集模块中，事件的捕获等待验证与事件的验证后反馈形成单生产者单消费者问题。因此，只需要在内核中维护一个生产者与消费者共享的变量来实现生产者消费者的同步，达到事件校验的效果。具体的来说，可以在内核中记录变量初始值为 0，每当有事件等待验证超时时将此变量加 1，当接收到事件验证结果时将此变量减 1，而事件的校验成功标识则为该变量保持为 0. 整体流程如下：

1. 内核态维护变量等待超时次数 n.

2. 内核态 Netfilter 在捕获到关键事件后将此事件发往用户态，然后开始进行超时等待验证结果。

3. 如果内核态等待超时，则将变量 n 加 1.

4. 用户态对事件进行事件验证，将验证结果通过 Netlink 发送回内核态。

5. 内核态的 Netlink 接收消息函数收到验证结果时，检查变量 n 的值是否为 0。如果 n 不为 0，表示事件不一致，将 n 减 1 后忽略该事件验证结果。如果 n 为 0，表示事件一致，则进行后续通过和拦截处理。

内核的 Netfilter 和 Netlink 在处理事件反馈上的相互同步流程分别如图 4-7 和图 4-8 所示：

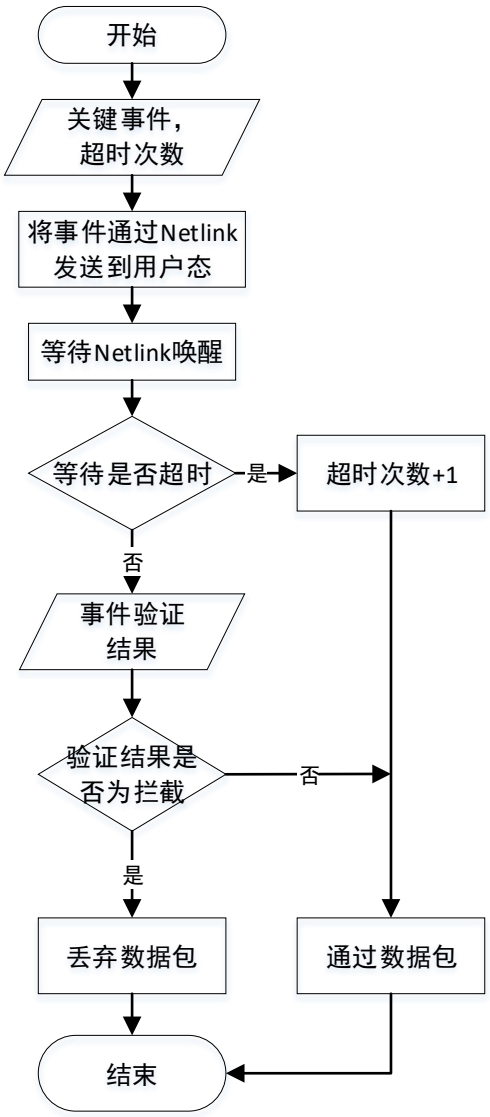

图 4-7 内核中 Netfilter 处理事件反馈的同步流程

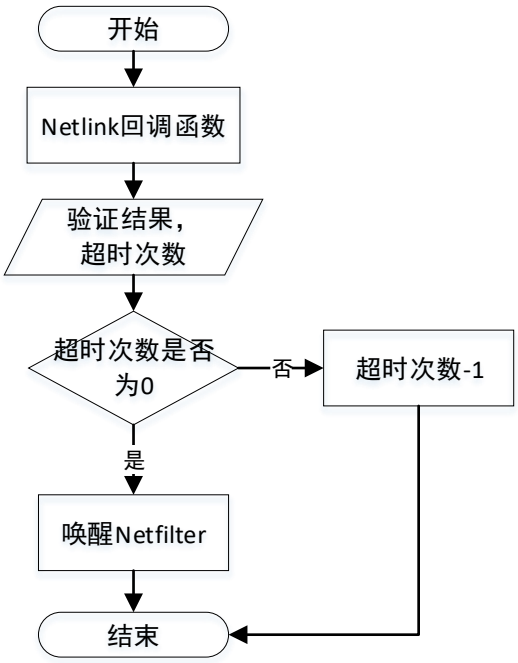

图 4-8 内核中 Netlink 处理事件反馈的同步流程

4.1.8 串口事件反馈拦截

串口事件反馈拦截与网络事件反馈拦截不同，在本文所述运行时验证系统中，串口中继器肩负着 KVM 内部物理串口和宿主机物理串口的中继转发功能，其双向通信链路并不是原来就有，而是需要串口中继器运行才能保持正常工作。同时，串口中继器本身运行在宿主机用户态，无需进行额外的内核态与用户态通信即可工作。因此，本文将串口事件采集模块与事件验证模块耦合，一方面轮询串口事件采集模块的两端文件描述符使其实现中继转发功能，另一方面在该模块采集到串口事件时将事件交由事件验证模块进行验证后再回到该模块继续进行处理。

对于串口事件中的非关键事件，在采集后直接由串口事件采集模块转发，同时事件验证模块在验证该非关键事件后也无需使用串口事件采集模块的接口将事件继续转发。而对于串口事件中的关键事件，在采集后根据事件验证的结果进行不同处理。如果验证通过，则调用串口事件采集模块的接口向另一端进行事件转发，如果验证失败，则直接丢弃该事件，不作转发处理。

串口中继器对事件反馈的处理流程如图 4-5 所示。

4.1.9 可视化模型绘制与控制展示界面

根据前文所述，事件的转移验证需要预定的状态机模型，事件在该状态机模型中推动节点沿着转移变化到另外的节点。状态机模型是一个多节点与节点间连线的图元结构，用户需要一个可视化的可绘制图元结构模型的界面。

mxGraph 是一个开源的 Web 前端 Javascript 绘图组件，适用于在网页中设计与编辑 Workflow/BPM 流程图、图表、网络图和普通图形的 Web 应用程序。它

允许用户在浏览器中进行快速方便的图元交互。本文对 mxGraph 进行源代码修改，以适用于运行时验证系统状态机绘制需求[23]。

mxGraph 同时包含有 Web 后端组件，可以实现对 Web 前端绘制页面的保存。图元结构的保存是一次序列化过程，序列化后的标识文本将在事件验证模块预处理中进行读取反序列化生成模型。mxGraph 的运行实例如图 4-9 所示：

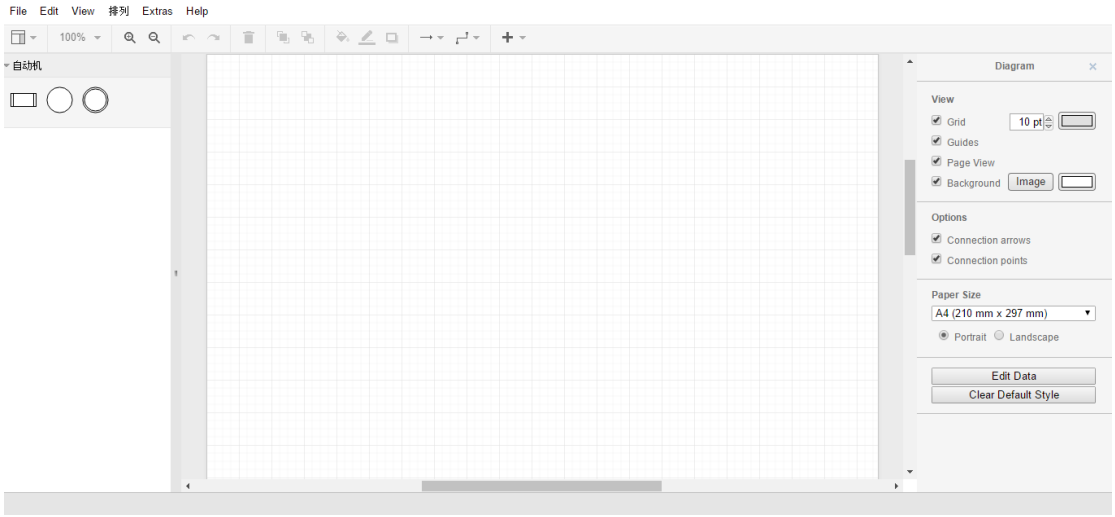

图 4-9 mxGraph 运行实例

除了状态机模型的绘制，整个运行时验证系统包含多个模块，需要多种自定义参数以满足系统的可扩展性。同时，系统运行中关键事件的捕获情况、拦截情况、各类异常情况等警告信息均需要有相应的界面向用户进行交互展示。

Qt 是一个 1991 年由 Qt Company 开发的跨平台 C++图形用户界面应用程序开发框架。它是面向对象的框架，很容易进行扩展，并且允许真正的组件编程。

本文使用 Qt 开发完整的交互式用户控制展示界面，包括可自定义的事件匹配头与匹配尾、网络通信事件采集限定的 KVM 内 IP 地址与外部通信设备 IP 地址、串口通信事件采集中映射成 KVM 内物理串口的宿主机伪终端路径和宿主机自身的物理串口路径。同时在通过用户界面启动整个运行时验证系统后，展示界面将会持续输出运行中的关键事件捕获、拦截、异常等各种信息，方便用户查看。

4.2 功能流程

整个物联网服务系统运行时验证系统在各模块接合协作时会涉及到诸多工作流程，如事件采集流程、事件验证流程、事件告警与拦截流程、用户操作流程等。为了能够更好的阐述系统整体设计与模块关系，本节主要从纵向的角度将系统中主要的工作流程进行时序上的分析，从而明确各模块间的工作方式和数据流向。

4.2.1 整体流程

在用户通过界面启动整个运行时验证系统后，首先启动后台工作线程，初始化事件验证模块，因为只有该模块做好准备接收事件进行验证，事件的采集和反馈才可以正常进行。然后网络事件采集、串口事件采集和内存事件采集依次被初始化。

所有模块都初始化成功后，界面客户端开始无限轮询各个事件采集模块客户端。各个事件采集模块客户端以各自的方式与实际工作的事件采集模块进行交互，判断是否有事件被采集以及接受采集的事件。采集到的事件统一交由事件验证模块进行验证处理。验证结果通过事件采集模块客户端进行反馈处理，同时由界面进行输出展示。

当用户通过界面停止整个运行时验证系统时，首先通过后台工作线程结束无限轮询。后台工作线程接收到通知后再通过各个事件采集模块客户端停止各个事件的采集，并进行相应的清理工作。

整体运行流程如图 4-10 所示：

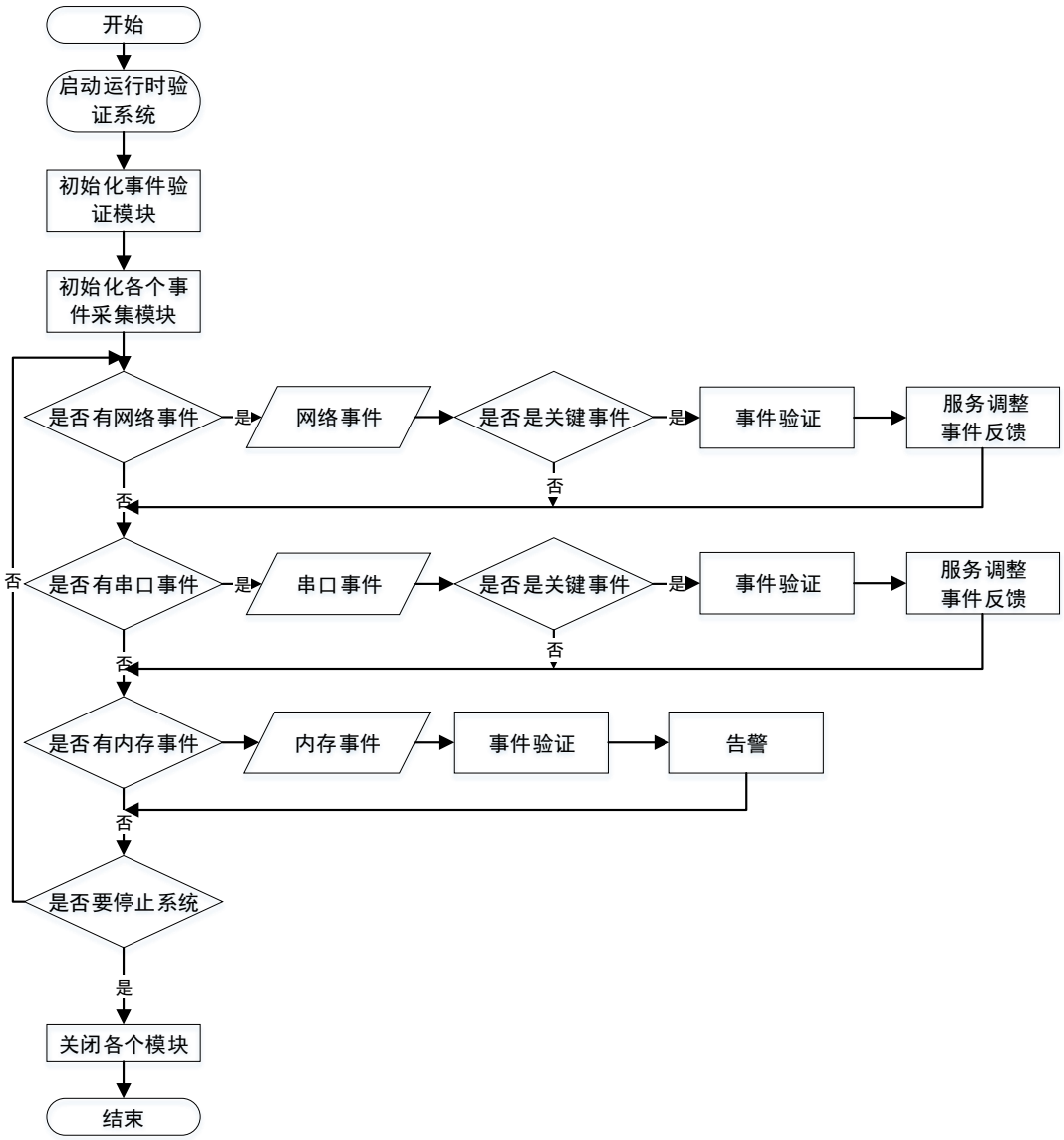

图 4-10 整体运行流程

4.2.2 事件采集流程

当用户配置好事件匹配头和匹配尾并开始对事件采集模块初始化时，网络通信事件采集模块和内存事件采集模块由于其均独立于运行时验证系统的客户端，因此需要在此先进行初始化。两个模块将事件匹配信息集成到自身的事件采集程序中，开始运行进行事件采集。

然后初始化的是串口通信事件采集模块，该模块集成在运行时验证系统的客户端中，初始化完成后便开始对映射成 KVM 内物理串口的伪终端和宿主机物理串口进行数据转发。

当客户端后台线程开始轮询三个事件来源时，依次查看该事件来源上有没有新的事件被采集到，如果发现新的事件，就对该事件进行事件验证处理。

4.2.3 事件验证流程

事件验证模块和串口通信事件采集模块一样被集成在界面系统客户端中。当用户启动运行时验证系统时，事件验证模块读取用户绘制的状态机图元转成的序列化文件描述，用来构造成自身的状态机结构。在完成事件验证模块的初始化之后，就可以随时接收事件采集模块发送过来的事件进行事件验证。

事件验证模块收到的事件是字符串表示的，首先需要对该字符串进行反序列化，提取出其中的变量信息。然后构造出自身的事件类型数据结构，将该结构添加到状态机结构中，利用上文所述事件验证模块的实现算法进行状态节点的转移验证。

事件在提取变量信息时还包括有是否为关键事件这一重要信息。对于非关键事件，事件验证只是为了改变状态机的状态，并不将事件验证结构发回到事件来源进行事件反馈。而对于关键事件，不仅要进行事件反馈，同时要进行相应的告警输出。

4.2.4 事件告警与拦截流程

关键事件在事件验证之后，将通过界面客户端的输出展示界面进行告警输出，同时反馈到事件的采集来源。对于网络通信事件和串口通信事件采集模块，两者在采集到关键事件时均会对事件验证结果进行等待。此时将关键事件的验证结果发回对应的模块，由模块完成事件的通过或者拦截操作。

4.2.5 用户操作流程

用户在配置整个物联网服务系统运行时验证系统时，需要按如下操作流程进行处理：

1. 将物联网服务系统部署在 KVM 内，配置 KVM 网络模式为桥接。

2. 在可视化状态机绘制界面上根据自定义需求绘制相应的状态机图形，并将其导出成序列化文本表示。

3. 启动运行时验证系统用户界面，编辑事件匹配头和匹配尾，配置各项参数。

4. 启动后台线程，开始运行时验证。

4.3 本章总结

本章主要论述了物联网服务系统运行时验证的概要设计。文章从系统架构出发，概要地介绍了各个模块内部的设计实现，然后从系统的各个功能流程角度对模块间衔接配合实现功能的方式进行了全面的说明。通过本章的阐述，整个物联网服务系统运行时验证系统的系统架构和设计有了清晰的概念，主要的算法细节与实现方式也有了详细的说明，接下来将对应用实践上的实现作更具体的阐述。

第五章 详细设计与实现

在第四章的论述中，我们从系统架构的角度详细阐述了各模块的功能设计与实现思路，通过各个主要流程描述了模块衔接工作过程。本章将从实际开发实践的角度上，具体到每一个细节详细阐述物联网服务系统运行时验证系统的实现方法。系统详细实现结构如图 5-1 所示：

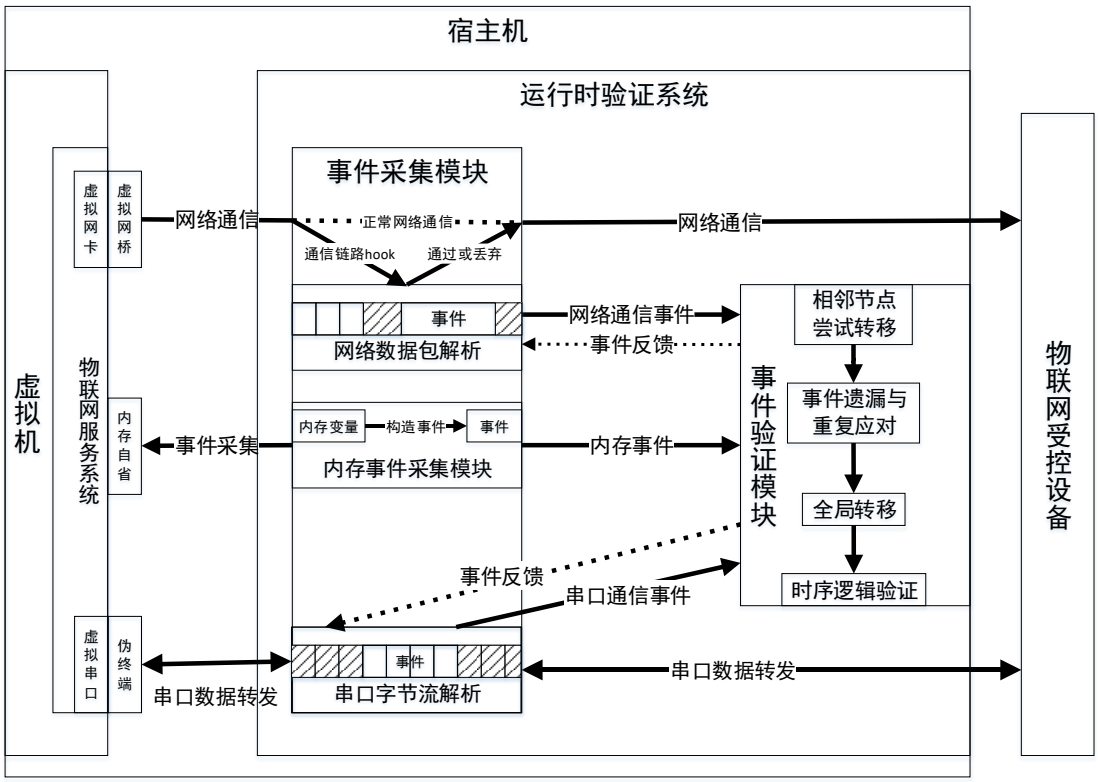

图 5-1 系统详细实现结构

5.1 事件采集

5.1.1 网络通信事件采集

5.1.1.1 Netfilter 的初始化

网络通信事件采集模块是由运行宿主机 linux 内核中的内核模块实现。

linux 内核是模块化组成的，它允许内核在运行时动态地向其中插入或从中删除代码。这些代码（包括相关的子线程、数据、函数入口和函数出口）被一并组合在一个单独的二进制镜像中，即所谓的可装载内核模块中。linux 终端提供简单的命令来完成内核模块的插入和删除，指令如下所示：

插入当前路径下的内核模块 xxx.ko

$ sudo insmod xxx.ko

删除内核模块 xxx

$ sudo rmmod xxx

linux 内核模块需要使用 GNU C 编译器编译，其语言与标准 C 语言有一些差别。内核模块中通过 struct nf_hook_ops 这个结构体表示 Netfilter 的函数钩子(hook)。在该函数钩子内可以指定 Netfilter 在网络上的挂载点为 NF_BR_PRE_ROUTING，表示数据链路层的第一个挂载点，以捕获虚拟网桥上的流量。

在初始化与结束模块时可以简单地使用函数 nf_register_hook(struct nf_hook_ops*)和 nf_unregister_hook()对 Netfilter 的函数钩子进行注册与卸载。

在 Netfilter 函数钩子中还可以指定其对应的 hook 函数，编写相应的 hook 函数即可在网络流量产生时触发调用此函数。该 hook 函数的原型如下：

unsigned int nf_hookfn(unsigned int hooknum, struct sk_buff *skb, const struct net_device *in, const struct net_device *out, int (*okfn)(struct sk_buff *));

其中 struct sk_buff 是 linux 网络底层最重要的数据结构，是整个网络传输的载体，该结构体关联着多个其他功能的成员字段，从中我们可以提取出该次网络通信过程中的当前数据包。

5.1.1.2 Netlink 的初始化

根据第四章所述的处理流程，首先要确定是否有用户态客户端连接，如果没有用户态程序通过 Netlink 与该内核模块连接，那么就可以直接通过此数据包减少对内核原有通信过程的影响。

由于 Netlink 通信协议控制字段需要包含有通信对端的进程 PID，因此在内核态使用用户态进程 PID 作为用户态客户端是否连接到内核的标识。该变量的定义如下所示：

__u32 userPid;

Netlink 在初始化时将 userPid 置为 0，在收到用户态客户端连接请求时，将 userPid 置为用户态进程的 PID。Netfilter 读取 userPid，如果其值为 0 即表示当前没有用户态客户端连接。Netlink 在收到用户态客户端断开连接请求时，重新将 userPid 置为 0。

Netlink 接受用户态消息的回调函数是由软中断实现的，因此在该内核模块中不会出现多线程同时读写变量可能异常的情况，所以模块内出现的用于 Netfilter 和 Netlink 公用的变量均不需要加锁处理。

5.1.1.3 数据链路层的数据包处理

物联网服务系统进行网络通信发出的事件是由网络协议层层包含的，其构成如图 5-2 所示。因此需要根据各层协议在 linux 内核中实现来拆解出事件数据。

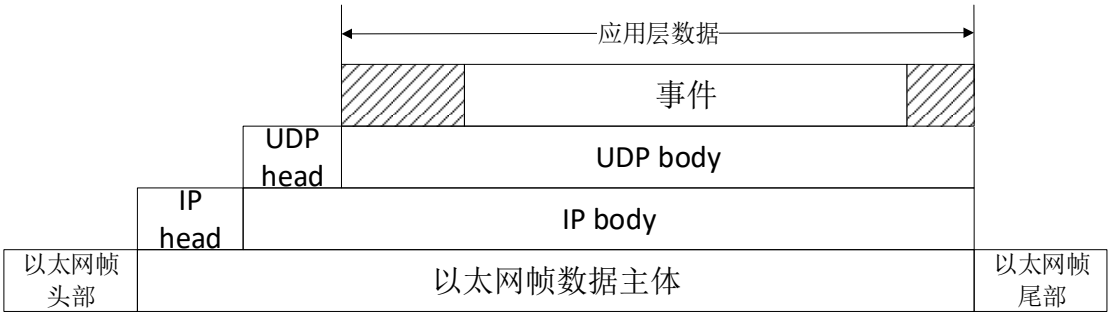

图 5-2 事件与网络协议关系

Netfilter 在确定有用户态客户端连接后，对 struct sk_buff 的指针 skb 以及由内核宏 eth_hdr()获取的以太网帧首部指针进行空指针判断，如果有其一为空，表示该数据包为空，直接通过即可。

linux 内核底层网络数据包报文的有效类型有 PACKET_HOST，PACKET_BROADCAST，PACKET_MULTICAST，PACKET_OTHERHOST，PACKET_OUTGOING5 种，分别表示发往本地主机、以太网广播报文、以太网多播报文、发往其他主机的报文和本地主机的环回报文。本文所述的物联网服务系统需要与外界设备进行通信，因此只需关注发往其他主机的报文。利用 skb 中字段 pkt_type，将值不是 PACKET_OTHERHOST 的数据包直接通过处理。

5.1.1.4 网络层的 IP 数据报处理

以上对网络数据包在数据链路层进行了相应的处理，然后使用内核宏 ip_hdr()获得 IP 数据报的首部指针，对网络层数据继续下一步处理，具体如下所示：

```
struct iphdr *iph = ip_hdr(skb);
```

IP 数据报首部中包含有源 IP 地址、目的 IP 地址、传输层协议等字段，其结构如图 5-3 所示[24]：

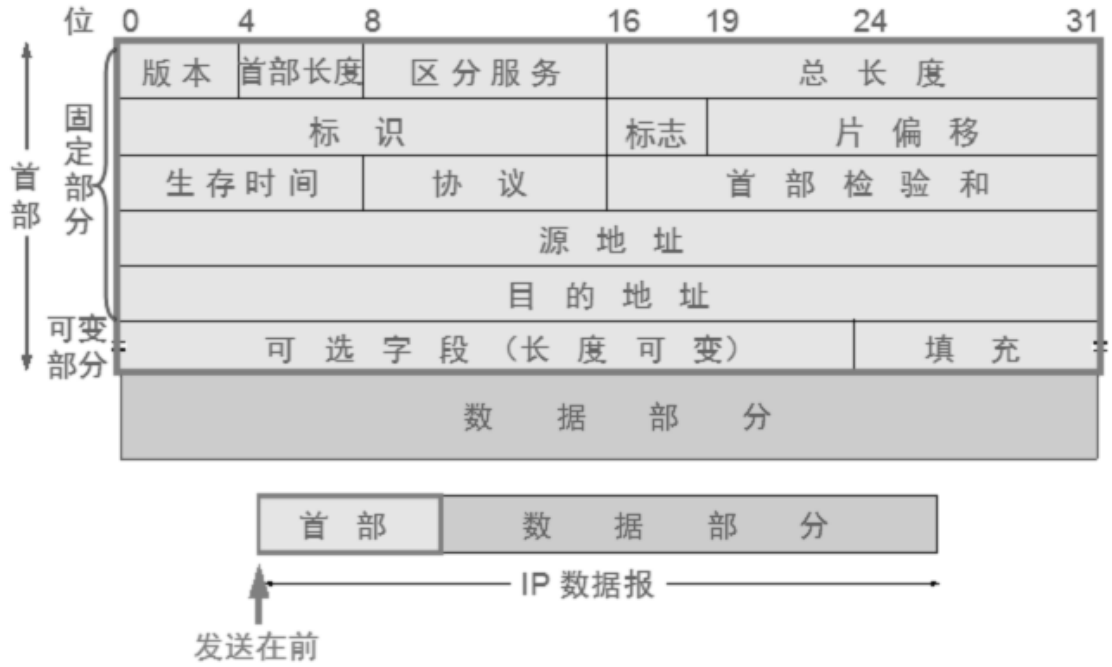

图 5-3 IP 数据报格式

网络层需要根据用户预定的源 IP 地址和目的 IP 地址过滤出满足要求的 IP 数据报。通过访问 iph 中字段 saddr 和 daddr 可以获取类型为__be32 的源 IP 与目的 IP 地址。__be32 是网络传输中使用的大端序 32 位无符号整数，而用户输入的 IP 地址类型往往为点分十进制的字符串类型，因此需要使用内核函数 in_aton() 将点分十进制的 IP 地址转化成__be32 类型才可以进行比较。

由于本文主要研究传输层协议为 UDP 的情况，通过访问 iph 中字段 protocol 可以获取此 IP 数据报中传输层协议，将值不是 IPPROTO_UDP 的数据包直接通过处理。

在对网络层的数据处理中，当 UDP 报文过长时 IP 数据报将发生分片。内核在存储单个分片时如数据长度较短则直接通过 skb->data 进行保存，称之为线性化存储；而当数据较长时，skb 将数据分片使用非线性化存储到各个分片上。当数据被分片存储时，由于其不连续性给数据的获取判断造成困难，因此可以使用内核网络数据线性化函数 skb_linearize()可以将各分片的数据聚集到同一个 skb 中集中进行处理。

5.1.1.5 传输层的 UDP 报文处理

以上对网络数据包在网络层进行了相应的处理，然后使用内核宏 udp_hdr() 获取 UDP 报文的首部指针，对传输层数据继续下一步处理，具体如下所示：

```
struct udphdr *udp_head = udp_hdr(skb);
```

UDP 报文的 body 部分可由首部指针向后偏移 UDP 首部长度获得，其中 UDP 首部长度为固定长度，具体如下所示：

```
char *udp_body = (char *)udp_head + sizeof(struct udphdr);
```

此时 udp_body 指向的便是我们需要的应用层数据，在该字符串中搜索事件匹配头和事件匹配尾，确定事件在应用层数据中的位置、长度和是否为关键事件。通过 Netlink 将此事件发送给用户态之后，开始超时等待返回结果。如上文所述，Netlink 的接收消息函数是用软中断实现的，触发此函数时无法确定 Netfilter 的运行情况，因此本文使用内核完成量 struct completion 来实现 Netlink 与 Netfilter 的消息同步，其超时等待函数原型如下所示：

```
unsigned long wait_for_completion_timeout(struct completion *comp, unsigned long timeout);
```

通过将 comp 在 Netfilter 与 Netlink 间共享，当 Netlink 接收到消息时使用 complete()函数唤醒在该处等待的 Netfitler。

Netfilter 会在等待超时时将此事件直接通过，如果等待被唤醒，则读取共享的事件验证结果进行处理。事件验证结果的表示定义如下所示：

```
enum UserCmd {
    ACCEPT,
    DISCARD,
} userCmd;
```

整个网络数据包采集处理时序如图 5-4 所示：

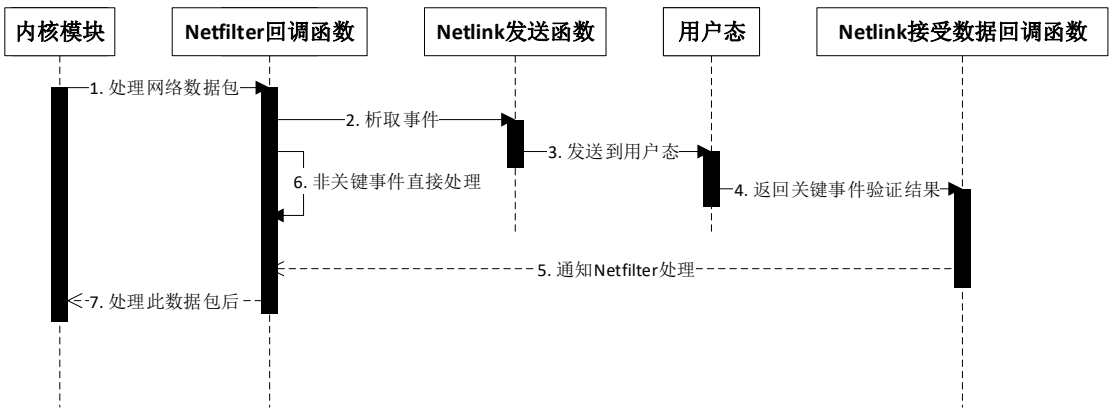

图 5-4 网络数据包采集处理时序图

5.1.1.6 用户界面客户端与内核的通信

用户界面客户端后台线程通过构造协议为 AF_NETLINK 的套接字与内核态的 Netlink 建立连接。为保障用户界面的后台逻辑不被阻塞，我们采用后台线程低时延轮询查询该套接字上是否有通信事件的方法。查询过程中应当使用非阻塞 I/O 进行超时等待，本文使用 I/O 复用中的 select 模型，将该套接字添加到 FD(File

Discriptor, 文件描述符)集合结构 fd_set 中，然后调用 select 函数在给定时间内等待该 FD 上的消息。select 函数原型如下所示[25]：

int select(int maxfdp, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval*timeout);

当客户端在调用 select 函数判断为有消息后，继续调用 recvfrom 函数一次性接收来自内核态的事件。

5.1.2 串口通信事件采集

串口通信事件采集模块集成在界面系统客户端中，该模块向界面系统的后台线程提供参数输入、模块初始化、事件查看与获取、通过与拦截事件等接口。

为了保障该模块本身的串口中继转发功能能高效率运行，我们同样使用上层的界面系统后台线程低时延轮询此模块，来查询模块上是否存在通信的事件。在该事件查询函数内依次访问其连接的两端 FD，即映射到 KVM 内物理串口的伪终端和宿主机自身的物理串口，查询 FD 上是否有字节流消息，如果有则进行事件析取。

串口中继器任意一端在调用 select 函数判断为有消息后，通过循环读取固定长度的字节流消息。然后在该字节流消息中进行事件头和事件尾的匹配，匹配成功的事件将通过自身向上层后台线程的接口提供给事件验证模块，匹配未成功的字节流将直接转发到串口中继器的另一端。

在上述字节流析取事件的设计中，由于字节流本身可能是连续的，但囿于缓冲区的大小和保证上层逻辑不阻塞，一次只能获取固定长度的字节流。由此可能出现的问题是单个事件在字节流传输时被分成两次获取，从而在匹配事件时匹配失败。为解决此问题，本文采用连续字节流滑动窗口的方式，仍然设置缓冲区为固定长度，但在部分字节流转发后将未转发的字节流前移，下次获取字节流时填充到缓冲区的末尾继续匹配。为了保证事件能够不被遗漏地全部被匹配到，要求缓冲区的长度要大于实际事件的最大长度。整个事件匹配中的过程如图 5-5 所示：

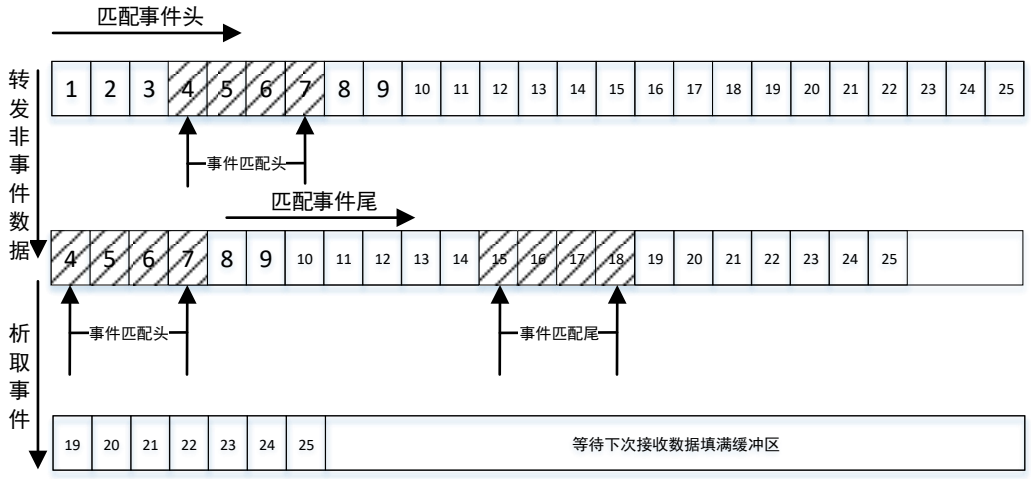

图 5-5 串口数据事件匹配原理图

事件匹配后被上层后台线程取走，此时串口中继器此次的中继转发工作暂时停止，直到下次上层轮询时才会进行下次中继转发工作。

5.1.3 内存事件采集

内存事件采集模块将从内存中物联网服务系统线程栈上采集到的数据按用户既定的事件格式封装成事件发送到界面系统客户端。由于该模块是独立进程运行，因此本文采用套接字实现事件的进程间传递。

内存事件采集模块开启一个 TCP server 进行监听，当界面系统客户端初始化时，建立一个 client 连接到此 server 上。与其他两个事件采集模块的连接相似，界面系统客户端通过 select 模型在固定时长内查看该事件来源上是否存在事件，如果存在事件则进行事件的接收并验证。

内存事件在验证完成后，如果是关键事件则将该关键事件的验证结果输出到用户界面显示。

5.2 事件验证

5.2.1 模型初始化

通过可视化状态机绘制界面，用户可以导出表示该状态机的序列化文件。当界面系统客户端启动后，在初始化模型时首先读取该序列化文件，解析提取其中的变量类型定义、状态机节点、转移关系和节点间时序约束，对相应的数据结构进行初始化。

本文使用 Z3 求解器对后续节点转移时表达式集合进行求解验证，而模型在初始化时提供给各状态机节点、转移关系以及节点间约束的均为字符串形式的表达式，因此在模型初始化时需要对字符串形式的表达式进行表达式解析并转化成 Z3 表达式保存在相应数据结构中[26]。该部分结构如图 5-6 所示：

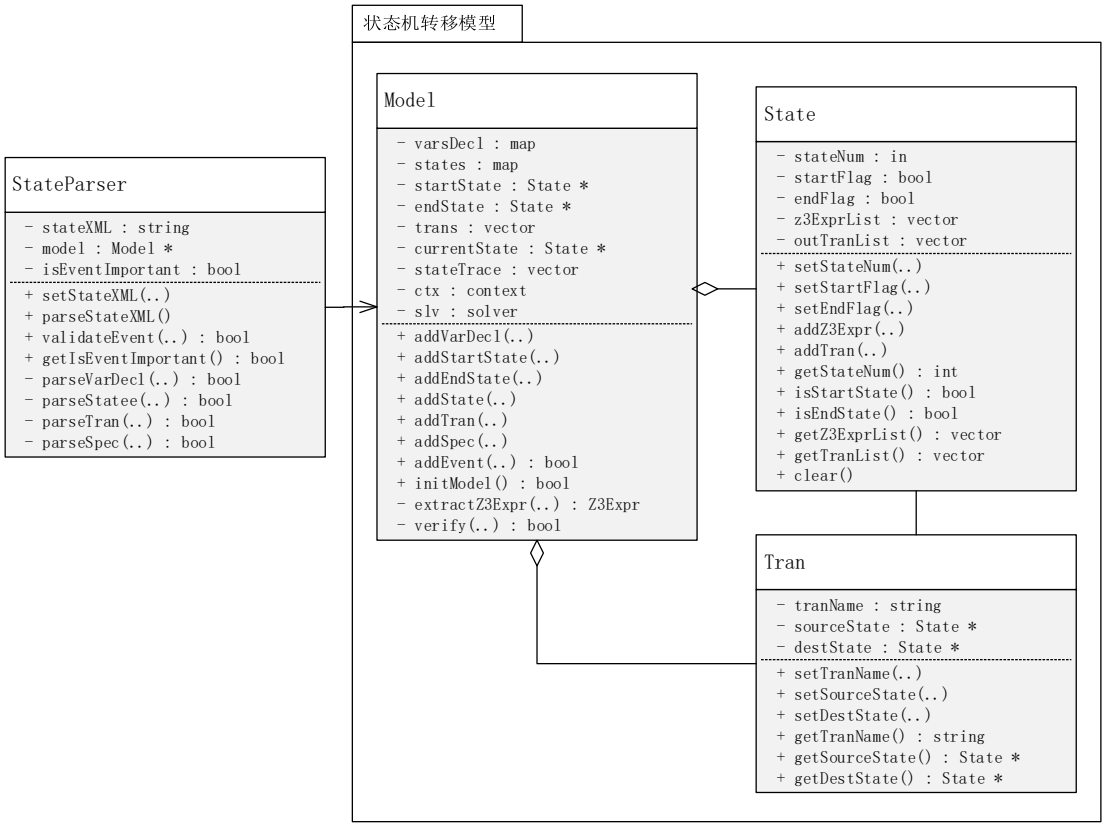

图 5-6 模型初始化 UML 图

StateParser 是对模型导出序列化文件的解析器，其向下调用状态机包的接口构建状态机，并向外提供序列化文件的输入接口。此外，当事件到达时，该类还负责进行事件的解析并输入到状态机中。

状态机模型由 Model 类和与之聚合其中的 State 类、Tran 类共同表示。Model 类管理着各状态机节点 State 类实例和各节点间的转移 Tran 类实例，为整个模型提供数据结构支持和内部算法实现。

5.2.2 事件解析转移

事件在采集中是由字符串形式表示的，而事件包含的信息到字符串这一过程可以有多种序列化方式。本文采用 xml 格式表示事件，事件名称、是否为关键事件等由 xml 根标签的相应属性表示，事件中的变量名和变量值对子标签表示。

事件的解析是事件反序列化的体现，将采集到的字符串形式的事件反序列化成内存中相应的数据结构，然后输入到事先建立好的模型中进行事件验证。

事件在添加到模型后根据模型中对变量类型的初始定义和事件中变量的具体值构成相应的 Z3 表达式，然后根据算法 4-3 进行事件转移。

事件在转移至下一状态时需要对该事件上的 Z3 表达式、以及下一状态上的 Z3 表达式进行联合求解。

Z3 提供求解器 solver 对多个表达式进行求解，可通过调用 solver::add()函数添加相应的 Z3 表达式，最后调用 solver::check()函数对添加的多个表达式进行联

合求解。同时在事件尝试转移中，需要调用 solver::push()和 solver::pop()对该次尝试转移过程中添加的状态上表达式进行弹出以保证整体最高效率验证运行。

整个事件获取和验证过程如图 5-7 所示：

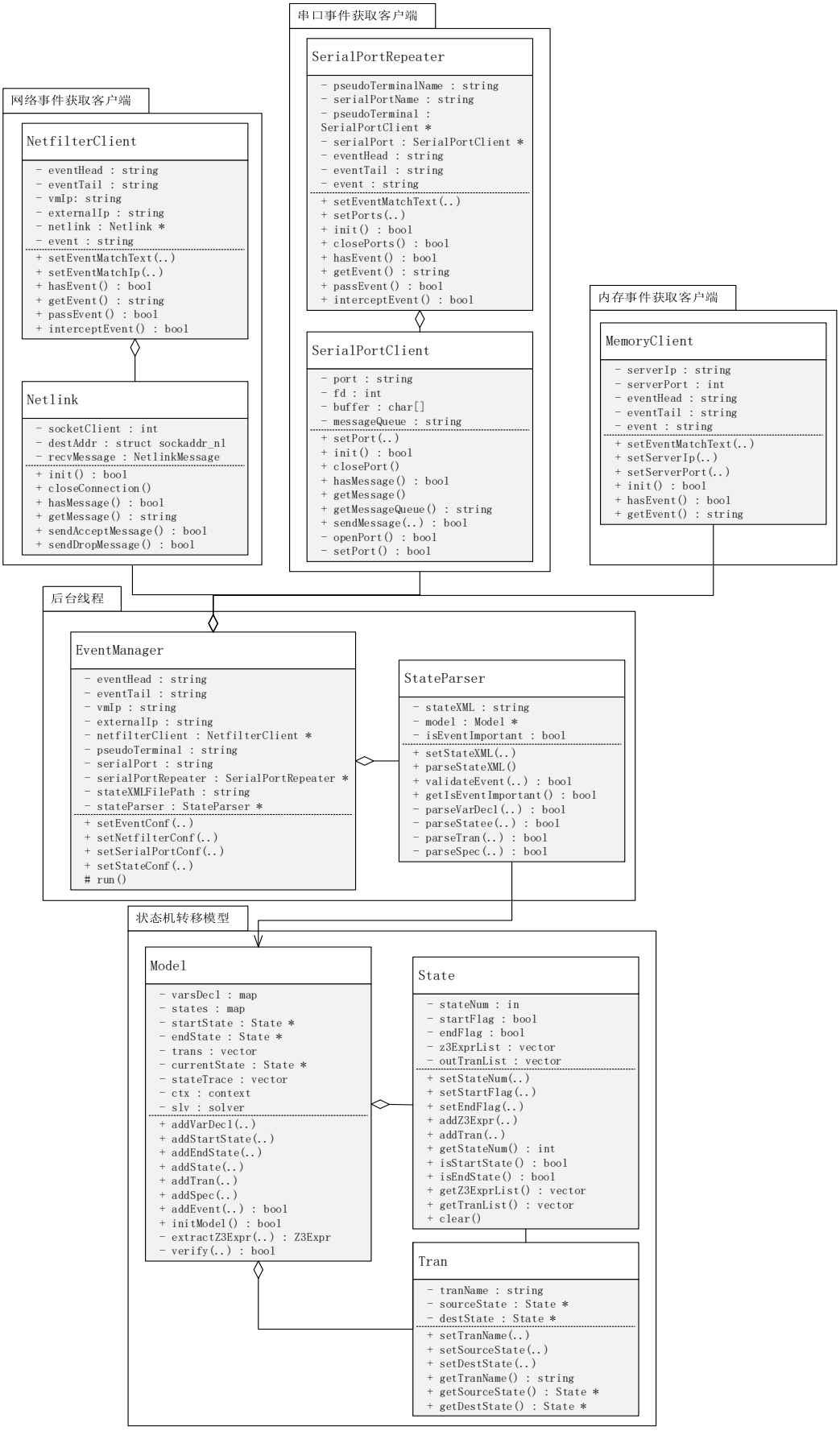

图 5-7 事件获取和转移验证 UML 图

EventManager 是界面客户端的后台线程，其控制着整个后台逻辑的运行。该类一方面与三个事件采集来源的客户端相连接，轮询调用各自接口获取事件，另一方面控制 StateParser 类，进行模型的构建与事件的移交状态机验证。

NetfilterClient 是管理连接内核的 Netlink 的客户端类，负责与内核进行通信。SerialPortRepeater 是管理两个串口客户端类 SerialPortClient 类实例的串口中继器，负责从两个串口客户端中获取字节流数据并析取事件，同时实现两端数据的相互转发。MemoryClient 是连接内存事件获取服务的客户端，负责获取远端的内存事件。

事件在通过各个客户端获取后，由 StateParser 类解析其中的变量，输入给 Model 进行事件转移验证，并返回结果给 EventManager，最后交还给来源客户端进行相应的处理。

5.3 非法事件告警

当事件采集模块中采集到的事件经由事件验证模块验证之中，用户需要对其中一些关键信息有所感知，包括关键事件的采集情况、关键事件与非关键事件的验证结果、关键事件的拦截情况。界面系统客户端可根据事件的关键性在事件处理的整个流程中进行相应的输出，其中重要的需要用户及时能看到的日志应当通过界面系统客户端直接输出到可视化图形界面上，而常规的处理日志只需要输出到控制台或者日志文件中即可。

5.4 非法事件拦截

5.4.1 网络通信事件拦截

用户界面系统客户端在验证事件后得出事件验证结果后，如果该事件为关键事件，则客户端将通过已有的 Netlink 连接向内核发送相应的操作指令。Netlink 协议的协议头定义如下[27]：

```
struct nlmsghdr {
    __u32 nlmsg_len;
    __u16 nlmsg_type;
    __u16 nlmsg_flags;
    __u32 nlmsg_seq;
    __u32 nlmsg_pid;
};
```

由于我们只需要传输一条指令通知内核对数据包的操作，因此整个向内核发送的 Netlink 数据包只需要包含协议头即可。即协议身长度为 0，表示如下：

nlmsg_len = NLMSG_LENGTH(0)

与内核通过过程中交互的指令包括建立连接、断开连接、通过数据包、拦截数据包，这些指令通过事件商定的 nlmsg_type 来表示。

此外，内核与用户态通过 Netlink 是通过用户态进程的 PID 来建立连接的，因此每次用户态向内核态发送事件反馈指令时需要设置 nlmsg_pid 如下所示：

nlmsg_pid = getpid();

内核态的 Netlink 在接收到用户态发来的消息后，首先判断该消息是否完整，然后根据 Netlink 协议头的 nlmsg_type 来进行相应操作。如果指令为通过或拦截数据包，首先需要对内核中已等待超时的计数变量进行判断，只有该变量保持为 0 时，才能通过内核完成量通知 Netfilter 对数据包返回 NF_ACCEPT 或 NF_DROP 的处理结果，否则忽略此次指令，将计数变量减 1。

5.4.2 串口通信事件拦截

串口通信事件拦截是基于串口中继器的消息转发原理之上的。由于非关键事件在串口上被捕获后，虽然交给事件验证模块进行事件转移，但串口中继器本身会直接将该事件进行转发。而关键事件在采集后会等待事件验证模块的处理结果。如果事件验证通过，则调用串口中继器的接口将该事件转发来事件发送端的对端上，如果事件验证未通过，则不转发此事件，作忽略处理即可在串口通信的两端表现出该事件已被拦截。

5.5 系统控制运行展示界面

系统控制运行与展示界面是用 Qt 编写的一个 GUI 可视化程序，高度集成了事件采集、事件验证和事件反馈各模块。并且对运行在内核的网络通信事件采集模块实现了透明地自动化部署。事件定义界面如图 5-8 所示：

图 5-8 系统界面事件配置

其中事件是由完整的 XML 表示，根据事件匹配的需要将其拆分为事件匹配头、非匹配的事件主体、事件匹配尾，由三者合并共同表示该事件。由于事件匹配过程需要固定化匹配头与匹配尾，所以对于事件的前后共同特征将提取出置于“事件匹配头部”和“事件匹配尾部”中。事件名称等变化的属性和具体的变量表示方式置于“事件非匹配主体”中，作为用户提示使用，在实际事件匹配中不起作用，仅为规范事件的定义用。

事件的完整构成举例如图 5-9 所示：

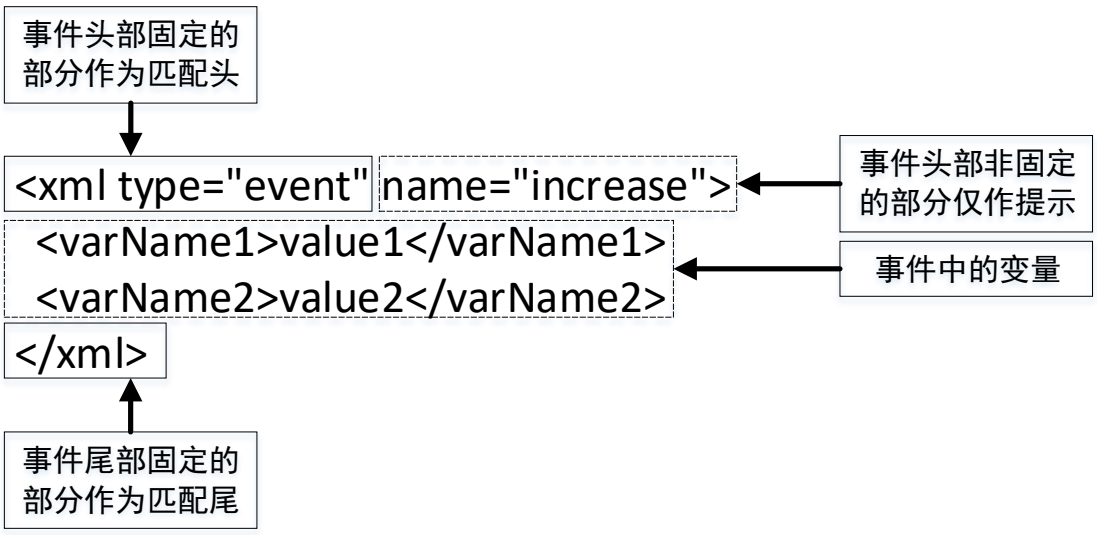

图 5-9 事件完整构成图

系统的运行展示界面如图 5-10 所示：

图 5-10 系统控制运行展示界面

用户只需要在界面输入事件的匹配头与匹配尾、网络通信过程中物联网服务系统与物联网受控设备的 IP 地址、串口通信过程中宿主机物理串口路径与映射成 KVM 内物理串口的宿主机伪终端路径。当用户点击启动按钮后，该界面将开启后台线程对各模块进行后台自动化部署。

5.6 本章总结

本章主要讲述了物联网服务系统运行时验证系统从理论到实际实现的详细过程，包括各模块内部使用的关键性技术以及接口的使用方法，还有模块间衔接的高效实现，模块的架构组织等。通过阅读本章，读者可以对本文所述系统理论方法如何落地施行在细节上和代码层面上有更进一步的了解。本章对物联网服务系统运行时验证系统的实现基本覆盖了功能需求中的各个方面，达到了设计目标，在下一章中，将对整个系统功能和性能进行全方位的测试，以使得整个系统能够真正地应用到实际生产环境中，为物联网服务系统作保障。

第六章 系统测试

本章对实现的物联网服务系统运行时验证系统进行全方位的测试。测试包括功能和性能两部分。首先，从功能测试的角度来说，整个系统各个模块要保证模块自身实现的功能达到设计目标，向外提供的接口要准备无误，各个模块间要合理衔接，能协调动作完成整体设计。其次，从性能测试的角度来说，整个系统需要满足高性能、高可靠性和健壮性等要求，这样才能满足实际应用环境的需要，所以对系统各模块和系统整体进行运行效率及对原有系统影响的测试是必不可少的环节。本章主要对功能和性能测试进行测试用例的设计与编写，通过测试程序对实现的物联网服务系统运行时验证系统进行相应的测试，最后分析测试数据结果得出对系统的综合评价。

6.1 测试环境

整个系统基于虚拟化环境，需要模拟实际运作的物联网服务系统。我们选择两台性能良好的物理机分别作为部署物联网服务系统的宿主机和物联网受控设备。

6.1.1 硬件环境

部署物联网服务系统的宿主机配置如下：

电脑型号　联想 启天 B4360-B015 台式电脑
处理器　英特尔 Pentium(奔腾) G2030 @ 3.00GHz 双核
内存　4 GB (三星 DDR3 1333MHz / 记忆科技 DDR3 1600MHz)
主硬盘　希捷 ST500DM002-1BD142 (500 GB / 7200 转/分)
网卡瑞昱　RTL8168/8111/8112 Gigabit Ethernet Controller / 联想

虚拟机配置如下：

处理器　虚拟的单核 CPU
内存　1GB
硬盘　50GB
网卡　虚拟网卡

模拟物联网受控设备的物理机配置如下：

电脑型号　联想 启天 B4360-B015 台式电脑
处理器　英特尔 Pentium(奔腾) G2030 @ 3.00GHz 双核

内存　　4 GB (记忆科技 DDR3 1600MHz)

主硬盘　　希捷 ST500DM002-1BD142 (500 GB / 7200 转/分)

网卡瑞昱　RTL8168/8111/8112 Gigabit Ethernet Controller / 联想

6.1.2 软件环境

由于本文所研究的物联网服务系统部署的虚拟化监控器为 KVM，同时对 KVM 的网络通信事件采集使用的是向 linux 插入内核模块的方式，因此部署物联网服务系统的宿主机操作系统应当使用主流 linux。本文所使用的内存事件采集模块是由 Java 和 Python 共同实现，因此宿主机中应当配置 JRE(Java Runtime Environment, Java 运行环境)和 Python 运行环境。用户界面客户端是由原生的 Qt 开发，因此宿主机中应当配置 Qt 运行环境。事件验证模块使用的是 Z3 求解器，因此宿主机中应当安装有 Z3 求解器。本文所使用的整个宿主机的软件环境如下：

1. Ubuntu16.04.01x64 桌面版
2. 内核版本 4.10.0-40-generic
3. QEMU-KVM 版本 1:2.5+dfsg-5ubuntu10.16
4. libvirt 版本 1.3.1-1ubuntu10.15
5. virt-manager 版本 1:1.3.2-3ubuntu1.16.04.4
6. JDK 版本 1.7.0_91
7. Python 版本 2.7.12
8. Qt 版本 5.9.2
9. Z3 求解器版本 4.4.0-5

由于物联网服务系统需要使用 Java 运行，因此在虚拟机内安装相应的 Java 运行环境。同时，内存事件采集模块分析的 JVM 虚拟机是特定于 JDK1.7 版本的，因此本文所使用的虚拟机软件环境如下：

1. Ubuntu16.04x64 桌面版
2. JRE 版本 1.7

另一台物理机用于模拟物联网受控设备，其操作系统系统与软件环境均不受限制，只需要能够使用网络和串口与物联网服务系统进行通信即可。

两台物理机之间处于同一局域网网段，最大传输网速为 100Mbps。同时两台物理机通过串口线相连接，串口标准为 RS-232-C，使用常用的波特率 9600。

6.2 测试环境部署

在进行功能和性能测试之前，需要先在准备好的物理机上部署好测试环境。本文对物联网服务系统运行时验证系统实现了一体化自动部署，整体根据用户界面分为两个部分，分别是可视化状态机绘制界面和系统控制运行展示界面。

6.2.1 可视化状态机绘制界面

用户对物联网服务系统的描述是通过有限状态自动机来表示的，因此首先的是可视化的图元绘制界面。在本文所实现的系统中，该界面是由开源软件 mxGraph 修改后实现而成。mxGraph 的 Web 前端部分无需安装，使用浏览器打开相应位置的 html 页面即可显示，此时用户便可在 Web 页面上实现图元绘制。因为用户绘制的状态机图形需要通过 mxGraph 序列化成文本表示，因此需要 mxGraph 提供的后端保存图形功能支持。本文修改后的 mxGraph 后端使用 Java 编写，提供基于 Ant 的自动化构建，用户只需要在 mxGraph/java 目录下输入 ant 构建指令即可开启 mxGraph 后端服务功能。具体指令如下：

$ ant grapheditor

此时再通过 Web 前端绘制状态机图形时即可进行相应的序列化保存到本地功能。

本文采用的测试用模型分为简单模型和复杂模型两例，分别如图 6-1 和图 6-2 所示：

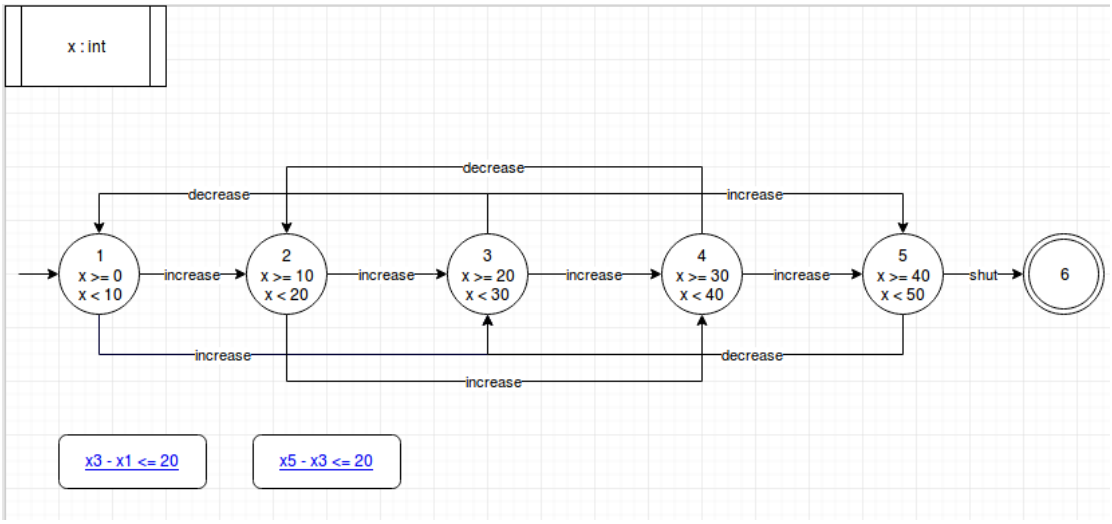

图 6-1 测试用简单模型

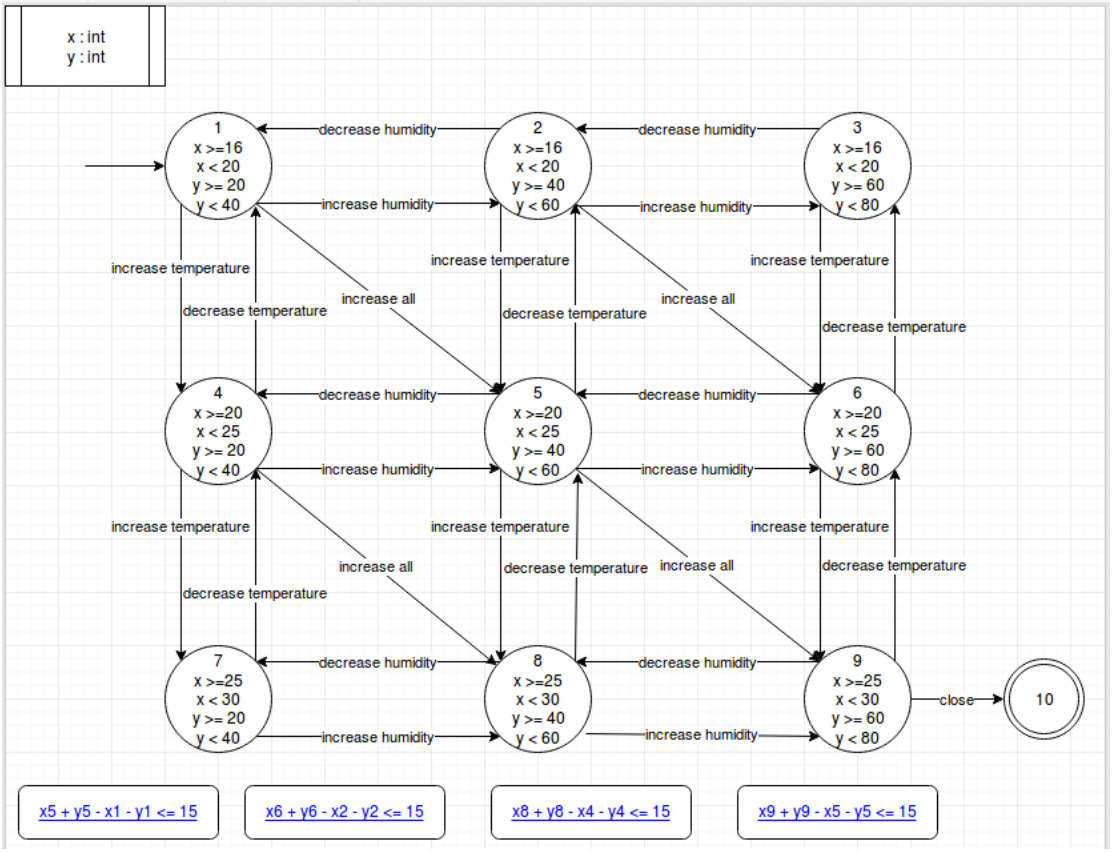

图 6-2 测试用复杂模型

其中均包括若干变量、状态节点，有一个起始节点和终止节点。此外还有多条时序约束的表达式，用于对系统运行中状态节点变化轨迹进行约束。

6.2.2 内存事件采集模块

该模块是单独的模块，未集成在本文所述的运行时验证系统之中，需要单独输入参数启动。该模块启动后将通过网络通讯提供内存事件。

6.2.3 物联网服务系统的部署

物联网服务系统需要安装在 KVM 中，同时设置 KVM 的网络模式为桥接模式，添加一个伪终端映射而成的物理串口。

在模拟物联网受控设备的物理机上部署相应的模拟程序，检测其与物联网服务系统的网络与串口通信是否异常。

6.3 功能测试

首先，我们需要编写相应的测试用例以保障物联网服务系统运行时验证系统的各个模块是否能够正常实现其自身的需求。

6.3.1 测试用例

表 6-1 系统部署测试

测试编号：功能测试 1
测试项目：系统部署测试
测试目的：测试系统通过简单的部署后能否正常启动运行
测试用例： 1. 启动物理机 A，按上文所述安装配置宿主机测试环境； 2. 在宿主机内启动虚拟机，安装配置虚拟机软件环境； 3. 在宿主机内启动修改后的 mxGraph，打开浏览器绘制状态机； 4. 启动物理机 B，配置模拟物联网受控设备软件环境； 5. 在宿主机 A 的虚拟机内启动与物理机 B 的网络与串口通信程序，本例使用的是运行在 Web 容器 tomcat 下的 Web 应用； 6. 配置运行内存事件获取模块； 7. 启动物联网服务系统运行时验证系统，输入配置参数，启动验证系统；
预期结果： 1. 物理机 B 能够与宿主机 A 正常进行网络与串口通信； 2. 物联网服务系统运行时验证能正常部署网络通信事件捕获模块，并能正常启动集成在自身的各个模块。

表 6-2 正常不包含事件数据通信测试

测试编号：功能测试 2
测试项目：正常不包含事件数据通信测试
测试目的：测试在不包含运行时系统需要采集的事件数据通信情况下，通信过程和运行时验证系统能否正常运行
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其发送不包含事件的其他数据与物理机 B 进行网络与串口通信，本例使用的是运行在 Web 容器 tomcat 下的 Web 应用； 2. 启动物联网服务系统运行时验证系统，查看后台终端输出日志以及界面告警信息。
预期结果： 1. 物联网服务系统通信过程正常，运行时验证系统运行正常； 2. 运行时验证系统后台没有捕获任何事件的记录，界面上无告警信息。

表 6-3 合法事件通信测试

测试编号：功能测试 3
测试项目：合法事件通信测试
测试目的：测试当物联网服务系统使用合法事件作为网络与串口通信数据与受控设备进行通信时，运行时验证系统能否准确采集事件并对其中的关键事件做验证通过反馈处理。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络与串口发送符合事件规范的数据到受控设备，本例使用的是运行在 Web 容器 tomcat 下的 Web 应用。其中事件均能通过图 6-1 所示状态机正常转移，且其中部分事件为关键事件； 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，观察后台终端输出日志以及界面告警信息。
预期结果： 1. 物联网服务系统通信过程正常，运行时验证系统运行正常； 2. 运行时验证系统后台可以查看到采集到的所有由物联网服务系统主动发出的事件和内存事件采集模块构造的事件； 3. 界面客户端上无任何告警信息。

表 6-4 非法事件通信测试

测试编号：功能测试 4
测试项目：非法事件通信测试
测试目的：测试当物联网服务系统在与物联网受控设备进行通信过程中包含有非法事件时，运行时验证系统能否准确采集事件并对关键事件中的非法与合法事件进行反馈处理。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络与串口发送符合事件规范的数据到受控设备，本例使用的是运行在 Web 容器 tomcat 下的 Web 应用。其中一部分事件能通过图 6-1 所示状态机正常转移，另外一部分事件会在事件转移中出现无匹配的转移名称、无可转移的状态节点、未通过状态机时序上的约束的原因而无法转移。所有情况中均有一部分为关键事件，另一部分为非关键事件； 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，观察后台终端输出日志以及界面告警信息。
预期结果： 1. 物联网服务系统通信过程部分正常，另一部分通信数据丢失。其中丢失的数据正好与无法转移的关键事件完全对应； 2. 运行时验证系统运行正常，后台可以查看到采集到的所有由物联网服务系统主动发出的事件和内存事件采集模块构造的事件； 3. 界面客户端上显示有无法转移的非关键事件提醒信息和无法转移的关键事件告警信息。

表 6-5 事件重复、遗漏、失序应对测试

测试编号：功能测试 5
测试项目：事件重复、遗漏、失序应对测试
测试目的：测试当物联网服务系统在与物联网受控设备进行通信过程中事件的通信存在重复、遗漏与失序情况时，运行时验证系统能否根据相应算法进行应对处理。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络与串口发送符合事件规范的数据到受控设备，本例使用的是运行在 Web 容器 tomcat 下的 Web 应用。其中事件依次存在重复、遗漏和失序的行为； 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，观察后台终端输出日志以及界面告警信息。
预期结果： 1. 运行时验证系统运行正常，事件重复、遗漏和失序行为都通过相应的算法进行了应对处理； 2. 本文所述的事件重复、遗漏与失序应对算法可以很好的处理物联网服务系统在运行中出现的此类情况。

6.3.2 测试说明及结果分析

在整个功能测试过程当中，我们设计了大量的测试用例来对物联网服务系统运行时验证系统整体和各个模块进行了比较全面的测试，包括非事件常规数据的正常通信、合法事件的正常通信、关键事件的事件反馈、非法事件的事件拦截等。

测试过程使用的是真实环境的第三方服务软件，具体是运行在 Web 容器 tomcat 下的 Web 应用，可以满足实验的要求。

通过比对各项测试结果与预期结果，最终可以证明整个物联网服务系统运行时验证系统整体各部分功能均达到预期的设计目标，能够正常运行。系统在对第三方服务软件的保障上能够完成设计的非法事件拦截、事件重复遗漏失序应对等功能。

如图 6-3 和图 6-4 所示为运行时验证系统后台终端输出与用户界面客户端输出的相应结果展示：

图 6-3 运行时验证系统后台终端输出展示

图 6-4 用户界面客户端提醒与告警信息输出展示

6.4 性能测试

除了基本的功能上的需求之外，性能也是本文所述运行时验证系统的核心要求。由于运行时验证系统的施行，在一定程度上可能干扰到物联网服务系统的自身运作，因此对于各模块的运行效率以及运行时验证系统的资源消耗是本节所要测试的目标。

6.4.1 测试用例

6.4.1.1 网络通信事件采集效率性能测试

表 6-6 非关键定长事件网络通信采集最大速率测试

测试编号：性能测试 1
测试项目：非关键定长事件网络通信采集最大速率测试
测试目的：测试物联网服务系统在与受控设备使用网络通信时，仅发送非关键事件可以保持正常通信且事件均能被运行时验证系统所采集到的最大通信速率，并分析网络通信事件采集对原有物联网服务系统通信过程的影响。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络发送符合事件规范的数据到受控物联网设备。其中事件均为非关键事件，事件长度为定长 100 字节； 2. 分别调整事件发送间隔为 50、20、10、5、4、2 和 1 微秒，并使事件在发送时每 100 包进行统一的停顿，即每发送 100 包停顿 5000、2000、1000、500、400、200 和 100 微秒； 3. 对于每一种情况将测试总时间设定为 30s； 4. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 5. 启动运行时验证系统，记录每一种情况在 30s 内发送的事件数、受控端接收到的事件数、运行时验证系统采集到的事件数； 6. 将各项数据进行汇总，计算出不同事件发送速率下的事件正常采集率和事件正常接收率，绘制相应的图表，进行结果分析。
预期结果： 1. 事件的正常采集率和事件正常接收率在事件发送间隔较大下均非常接近 100%，但在事件发送间隔较小时，事件的采集和接收都出现一定程度的丢包现象； 2. 网络通信事件采集模块在只采集的情况下对物联网服务系统正常网络通信影响非常小。

表 6-7 非关键不定长事件网络通信采集恒定速率下的最长事件测试

测试编号：性能测试 2
测试项目：非关键不定长事件网络通信采集恒定速率下的最长事件测试
测试目的：测试物联网服务系统在与受控设备使用网络通信时，仅发送非关键事件可以保持正常通信且事件均能被运行时验证系统所采集到的最长事件。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络发送符合事件规范的数据到受控物联网设备。其中事件均为非关键事件； 2. 分别调整事件长度为 100、200、400、800、1500 字节，其中 1500 字节接近保持 IP 数据包不分片的 MTU 长度； 3. 对于每一种情况发送相同长度的事件 100 条； 4. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 5. 启动运行时验证系统，记录每一种情况受控端接收到的事件数、运行时验证系统采集到的事件数； 6. 将各项数据进行汇总，计算出不同事件长度下的事件正常采集率和事件正常接收率，绘制相应的图表，进行结果分析。
预期结果： 1. 事件的正常采集率和事件正常接收率均保持 100%，内核虽然对较长数据进行了分页存储，但采用线性化函数仍然可以进行正常处理；

6.4.1.2 串口通信事件采集效率性能测试

表 6-8 非关键定长事件串口通信采集最大速率测试

测试编号：性能测试 3
测试项目：非关键定长事件串口通信采集最大速率测试
测试目的：测试物联网服务系统在与受控设备使用串口通信时，仅发送非关键事件可以保持正常通信且事件均能被运行时验证系统所采集到的最大通信速率，并分析串口中继器的正常工作情况。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过串口发送符合事件规范的数据到受控物联网设备。其中事件均为非关键事件，事件长度为定长 100 字节，各串口设备波特率均为 9600，通信中包含 1bit 停止位和 1bit 奇偶校验位，理论带宽为 960 字节/秒； 2. 分别调整事件发送间隔为 1024ms、512ms、256ms、128ms、110ms、100ms、90ms、80ms、70ms、64ms，直到接收带宽略大于理论带宽为止； 3. 对于每一种情况将测试总时间设定为 30s； 4. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 5. 启动运行时验证系统，记录每一种情况在 30s 内发送的事件数、受控端接收到的事件数、运行时验证系统采集到的事件数； 6. 将各项数据进行汇总，计算出不同事件发送速率下的事件正常采集率和事件正常接收率，绘制相应的图表，进行结果分析。
预期结果： 1. 事件的正常采集率和事件正常接收率在事件发送间隔较大时均非常接近 100%； 2. 在事件发送间隔较小时，虚拟机内伪终端不受串口物理特性影响，发送速率能够远超过串口波特率所计算出的最大通信速率。而外部串口设备的缓存使其仅能达到略大于计算出的最大通信速率。此时事件的正常采集率保持不变，正常接收率将急剧下降； 3. 串口通信事件采集模块在只采集的情况下能在最大带宽内很好的实现串口中继器的功能。

6.4.1.3 事件验证效率性能测试

表 6-9 事件验证效率随时间变化测试

测试编号：性能测试 4
测试项目：事件验证效率随时间变化测试
测试目的：测试物联网服务系统运行时验证系统的事件验证模块在持续接收事件进行验证并验证由事件产生的轨迹时，其验证效率的变化情况。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络发送符合事件规范的数据到受控物联网设备。其中事件均为非关键事件； 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，记录接收到 100 条事件中每一条事件的验证耗时，并计算每种情况下的平均验证耗时； 4. 将各项数据进行汇总，绘制出事件验证耗时时间变化的图表，进行结果分析。
预期结果： 1. 事件的验证耗时整体波动幅度较小，维持在同一水平。个别事件验证耗时较为突出，可验证其为从起始节点出发的事件； 2. 增量式验证方法使得对事件的验证能以极高的效率运行，几乎不随时间变化而显著增长。

表 6-10 事件验证效率随表达式数量变化测试

测试编号：性能测试 5
测试项目：事件验证效率与模型复杂度对比测试
测试目的：测试物联网服务系统运行时验证系统的事件验证模块在不同复杂度的模型下，其验证效率的对比情况。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络发送符合事件规范的数据到受控物联网设备。其中事件均为非关键事件； 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，记录接收到 100 条事件中每一条事件的验证耗时，并计算每种情况下的平均验证耗时； 4. 保存图 6-2 所示状态机到宿主机 A 中，重新配置运行时验证系统使其读取该状态机序列化； 5. 重新启动运行时验证系统，记录接收到 100 条事件中每一条事件的验证耗时，并计算每种情况下的平均验证耗时； 6. 将各项数据进行汇总，绘制出两个模型在事件验证耗时变化上的对比图表，进行结果分析。
预期结果： 1. 事件的平均验证耗时几乎不随状态机模型的复杂而显著变化；

6.4.1.4 非法事件拦截效率性能测试

表 6-11 网络通信非法关键事件拦截最大流量测试

测试编号：性能测试 6
测试项目：网络通信非法关键事件拦截最大速率测试
测试目的：测试物联网服务系统运行时验证系统在网络通信非法事件的拦截上可以接受多大的非法关键事件流量。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过网络发送符合事件规范的数据到受控物联网设备。其中 5%的事件均为关键事件，且关键事件在通过图 6-1 所示状态机转移中均会出现无匹配的转移名称、无可转移的状态节点、未通过状态机时序上的约束的原因而无法转移，事件长度为定长 100 字节。 2. 分别调整事件发送间隔为 50、20、10、5、4、2 和 1 微秒，并使事件在发送时每 100 包进行统一的停顿，即每发送 100 包停顿 5000、2000、1000、500、400、200 和 100 微秒； 3. 对于每一种情况将测试总时间设定为 30s； 4. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 5. 启动运行时验证系统，记录每一种情况在 30s 内发送的事件数、受控端接收到的事件数、运行时验证系统采集到的事件数； 7. 将各项数据进行汇总，计算出不同事件发送速率下的事件正常采集率、事件正常接收率和发送端的平均发送速率，绘制相应的图表，进行结果分析。
预期结果： 1. 事件的正常采集率始终能保持接近 100%； 2. 事件的正常接收率始终能保持接近 95%；

表 6-12 串口通信非法关键事件拦截最大流量测试

测试编号：性能测试 7
测试项目：串口通信非法关键事件拦截最大速率测试
测试目的：测试物联网服务系统运行时验证系统在串口通信非法事件的拦截上可以接受多大的非法关键事件流量。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其通过串口发送符合事件规范的数据到受控物联网设备。其中 5%的事件为关键事件，且事件在通过图 6-1 所示状态机转移中均会出现无匹配的转移名称、无可转移的状态节点、未通过状态机时序上的约束的原因而无法转移，事件长度为定长 100 字节。 2. 分别调整事件发送间隔为 1024ms、512ms、256ms、128ms、110ms、100ms、90ms、80ms、70ms、64ms，直到接收带宽略大于理论带宽为止； 3. 对于每一种情况将测试总时间设定为 30s； 4. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 5. 启动运行时验证系统，记录每一种情况在 30s 内发送的事件数、受控端接收到的事件数、运行时验证系统采集到的事件数； 7. 将各项数据进行汇总，计算出不同事件发送速率下的事件正常采集率、事件正常接收率和发送端的平均发送速率，绘制相应的图表，进行结果分析。
预期结果： 1. 事件的正常采集率始终能保持接近 100%； 2. 事件的正常接收率始终能保持为 95%； 3. 发送端的平均发送速率与通过波特率计算出的值几乎保持一致，说明串口通信事件采集模块在处理串口这种低速通信设备时几乎不会给其带来影响。

6.4.1.5 运行时验证系统资源占用性能测试

表 6-13 运行时验证系统资源占用性能测试

测试编号：性能测试 8
测试项目：运行时验证系统资源占用性能测试
测试目的：测试物联网服务系统运行时验证系统在实际运行过程中对操作系统 CPU、内存的资源占用情况。
测试用例： 1. 修改宿主机 A 内虚拟机中的物联网服务系统，使其同时通过网络和串口发送符合事件规范的数据到受控物联网设备。 2. 保存图 6-1 所示状态机到宿主机 A 中，配置运行时验证系统使其读取该状态机序列化文件； 3. 启动运行时验证系统，记录运行时验证系统对宿主机 CPU 和内存占用随时间变化的情况，绘制相应的图表，进行结果分析。
预期结果： 1. 运行时验证系统对系统资源内存占用较小，对 CPU 在验证事件时有一定程度上的占用，对整个物联网服务系统整体影响不大。 2. 运行时验证系统在占用 CPU 资源上与事件相关。

6.4.2 测试说明及结果分析

6.4.2.1 网络通信事件采集效率性能测试

性能测试 1 的结果如表 6-14 和图 6-5 所示：

表 6-14 非关键定长事件网络通信采集最大速率测试结果表

事件平均发送间隔(微秒)	发送端发送的事件数	运行时验证系统采集到的事件数	接收端接收到的事件数	事件正常采集率	事件正常接收率
50	479800	479800	479800	100.00%	100.00%
20	965600	965600	965600	100.00%	100.00%
10	1004406	1004406	1004406	100.00%	100.00%
5	1853252	1853252	1853252	100.00%	100.00%
4	2249235	2249235	2248253	100.00%	99.96%
2	2273285	2273285	2259468	100.00%	99.39%
1	2530451	2525239	2437098	99.79%	96.31%

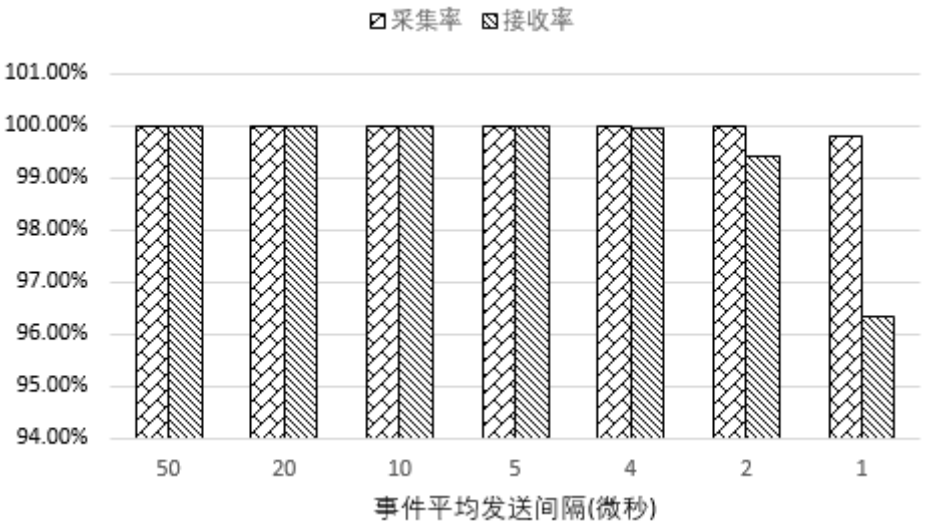

图 6-5 非关键定长事件网络通信采集最大速率测试柱形图

从图中可以看出，事件的正常采集率和事件正常接收率在事件平均发送间隔大于 2 微秒时事件的正常采集率能保持 100%的完整接收。此时可根据在 30s 内事件采集数量计算出采集的最大速率为 2273285 / 30 = 75776 事件/秒，即 75776 * 100 * 8 = 60.62Mbps，在测试环境最大带宽 100Mbps 的情况下，可以满足大部分应用需求。此外，当事件发送速率进一步增加时，网络事件的采集出现了一定的丢失，说明本文所使用的 netlink 内核与用户态通信的方式在复制网络数据包时存在一定的效率上的瓶颈。

性能测试 2 的结果如表 6-15 和图 6-6 所示：

表 6-15 非关键不定长事件网络通信采集恒定速率下的最长事件测试结果表

事件长度	发送端发送的事件数	运行时验证系统采集到的事件数	接收端接收到的事件数	事件正常采集率	事件正常接收率
100	100	100	100	100.00%	100.00%
200	100	100	100	100.00%	100.00%
400	100	100	100	100.00%	100.00%
800	100	100	100	100.00%	100.00%
1500	100	100	100	100.00%	100.00%

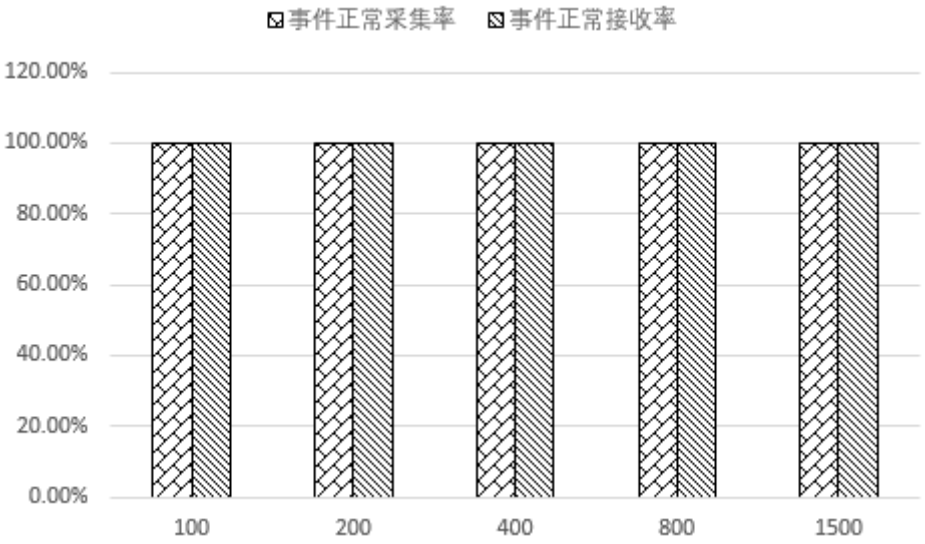

图 6-6 非关键不定长事件网络通信采集恒定速率下的最长事件测试柱形图

从图中可以看出，事件的正常采集率和事件正常接收率均能达到 100%。当事件较长时，此时通信过程中的 IP 数据报在内核中由 struct sk_buff 的 data 指针直接指向转变成分页非线性存储，但通过内核线性化处理函数可以收集分页进行连续处理。从结果中可以得出结论，网络通信事件采集模块能在应用层数据不分片时能正常采集所有数据，最长事件采集能力可以达到 1500 字节。

6.4.2.2 串口通信事件采集效率性能测试

性能测试 3 的结果如表 6-16 和图 6-7 所示：

表 6-16 非关键定长事件串口通信采集最大速率测试结果表

事件发送间隔(毫秒)	发送端发送的事件数	运行时验证系统采集到的事件数	接收端接收到的事件数	事件正常采集率	事件正常接收率
1024	30	30	30	100.00%	100.00%
512	60	60	60	100.00%	100.00%
256	120	120	120	100.00%	100.00%
128	240	240	240	100.00%	100.00%
110	271	271	271	100.00%	100.00%
100	295	295	295	100.00%	100.00%
90	330	330	288	100.00%	87.27%
80	371	371	288	100.00%	77.63%
70	424	424	298	100.00%	70.28%
64	463	463	300	100.00%	64.79%

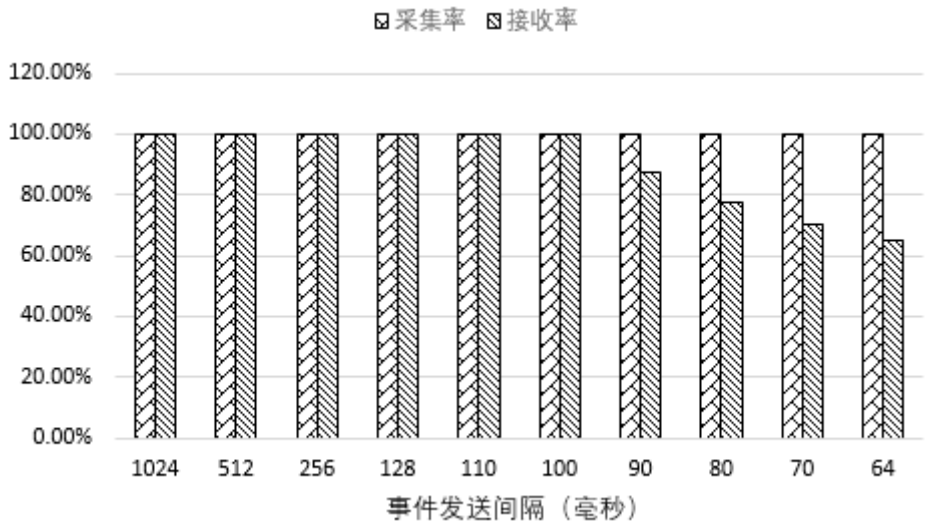

图 6-7 非关键定长事件串口通信采集最大速率测试柱形图

从图中可以看出，事件的正常采集率始终能保持在 100%，而事件的正常接收率在事件发送间隔小于 100 毫秒时开始下滑。根据 100ms 时事件的正常接收情况可以计算出此时的带宽为 295 /30 * 100 = 983.33 字节/秒，比理论上的最大带宽略大。其原因是物理串口设备存在输入输出缓存，能够适当提升通信效率。

因此可以从结果中得出结论串口通信事件采集模块在只采集的情况下能同时很好的实现串口中继器的功能。实际采集最大速率为 9.83 事件每秒。

6.4.2.3 事件验证效率性能测试

性能测试 4 的结果如表 6-17 和图 6-8 所示：

表 6-17 事件验证随时间变化耗时测试结果表

事件序号	事件验证耗时(ms)
1	0.98
2	0.444
3	0.064
4	0.467
5	0.492
6	0.073
7	0.59
8	0.49
9	1.229
10	0.075
…	…
100	0.319
平均值	0.31402

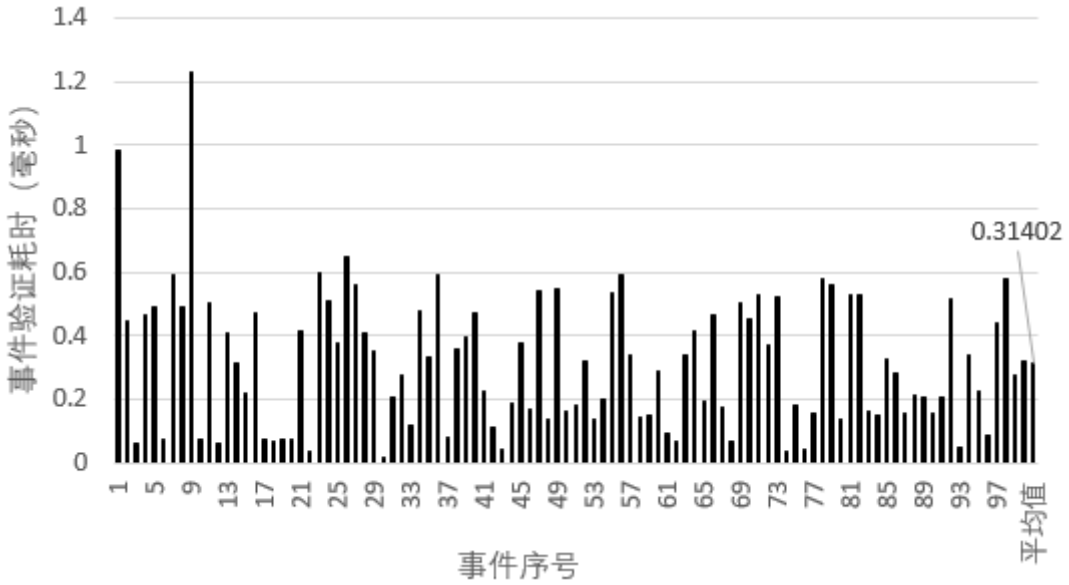

图 6-8 事件验证随时间变化耗时测试柱形图

从图中可以看出，事件的验证耗时整体表现较低且波动幅度较小，维持在 0.05~1.3 毫秒之间。图中个别事件表现为较高的验证耗时，可以验证均为从状态机起点出发的事件，因为状态机从起点开始转移时需要进行初始化工作添加时序

约束逻辑，因此需要更多的时间。由此可以得出结论，本文采用的事件验证方法与实现可以以非常高的效率执行，事件验证耗时表现较低且较为稳定。

性能测试 5 的结果如表 6-18 和图 6-9 所示：

表 6-18 事件验证效率与模型复杂度对比测试结果表

事件序号	简单模型事件验证耗时(ms)	复杂模型事件验证耗时(ms)
1	1.158	1.149
2	0.304	0.24
3	0.254	0.291
4	0.251	0.24
5	0.286	0.211
6	0.256	0.256
7	0.268	0.215
8	0.265	0.307
9	0.243	0.297
10	0.267	0.267
…	…	…
100	0.251	0.292
平均值	0.31068	0.29415

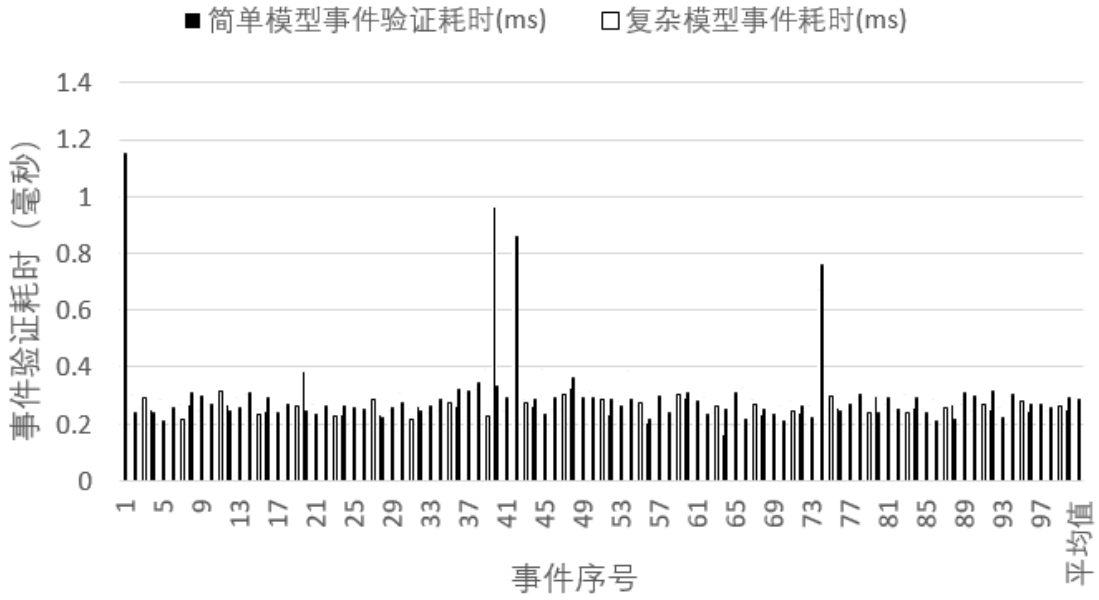

图 6-9 事件验证效率与模型复杂度对比测试柱形图

从图中可以看出，事件的平均验证耗时几乎不随状态机模型复杂度变化而有明显变化。由此可以得出结论，本文采用的节点上字符串表达式预处理成 Z3 表

达式并在运行中动态增量添加 Z3 表达式的方法可以最大限度的降低复杂模型带来的验证效率问题，提高系统执行效率。

6.4.2.4 非法事件拦截效率性能测试

性能测试 6 的结果如表 6-19 和图 6-10 所示：

表 6-19 网络通信非法关键事件拦截最大速率测试结果表

事件发送间隔(ms)	发送端发送的事件数	运行时验证系统采集到的事件数	接收端接收到的事件数	事件正常采集率	事件正常接收率
50	479824	479824	455833	100.00%	95.00%
20	965663	965663	917380	100.00%	95.00%
10	1004723	1004723	954487	100.00%	95.00%
5	1859527	1859527	1766551	100.00%	95.00%
4	2249452	2249452	2136979	100.00%	95.00%
2	2274256	2274256	2160543	100.00%	95.00%
1	2530257	2523425	2456373	99.73%	97.08%

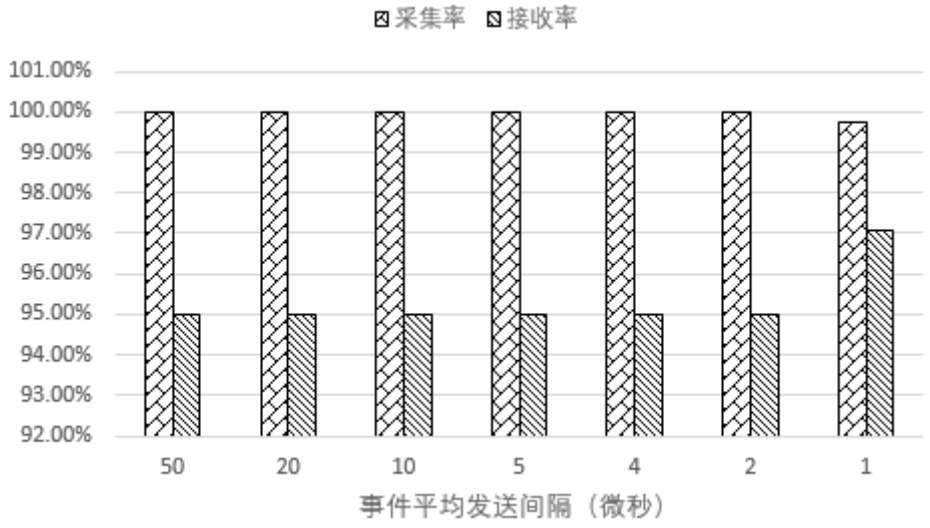

图 6-10 网络通信非法关键事件拦截最大速率测试柱形图

从图中可以看出，在事件平均发送间隔大于 2 微秒时，事件的正常采集率始终能保持 100%，事件的正常接收率始终能保持为 95%。说明事件采集过程没有遗漏，同时对非法事件的拦截也没有遗漏。但事件的平均发送间隔小于 2 微秒时，事件的正常采集率开始下滑，同时事件的正常接收率开始上升，说明事件拦截出现部分因为处理不及时而对事件放行的情况。

由此可得出结论，由于运行时验证系统对关键事件的验证效率有一定的瓶颈，实际运行中在关键事件比例为 5%时可接受的最大事件速率为 2274256 / 30 = 75808.53 事件每秒。

性能测试 7 的结果如表 6-20 和图 6-11 所示：

表 6-20 串口通信非法关键事件拦截最大速率测试结果表

事件发送间隔(ms)(共 5s)	发送端发送的事件数	运行时验证系统采集到的事件数	接收端接收到的事件数	事件正常采集率	事件正常接收率
1024	30	30	29	100.00%	96.67%
512	60	60	57	100.00%	95.00%
256	120	120	114	100.00%	95.00%
128	240	240	228	100.00%	95.00%
110	271	271	257	100.00%	94.83%
100	295	295	280	100.00%	94.92%
90	331	331	314	100.00%	94.86%
80	370	370	351	100.00%	94.86%
70	425	425	403	100.00%	94.82%
64	464	464	441	100.00%	95.04%

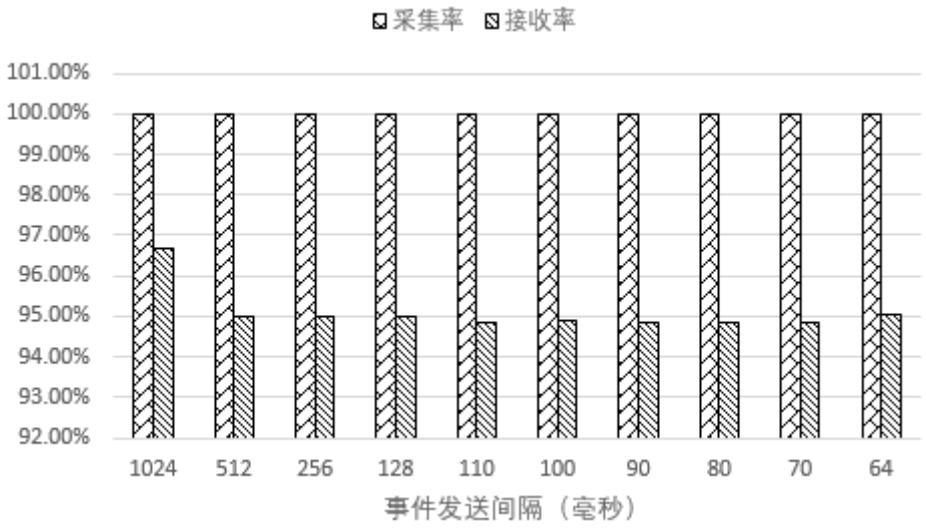

图 6-11 串口通信非法关键事件拦截最大速率测试柱形图

从图中可以看出，事件的发送间隔较小时，事件的正常采集率始终能保持 100%，事件的正常接收率始终能保持接近于为 95%。通过进一步的验证可以证明事件采集与对非法事件的拦截过程均没有遗漏。

由此得出结论，串口通信事件采集模块在处理串口这种低速通信设备时对非法关键事件的拦截与串口的实际最大通信速率相近，实际速率为 $464/30=15.47$ 事件每秒。

6.4.2.5 运行时验证系统资源占用性能测试

性能测试 8 的结果如表 6-21 和图 6-12 所示：

表 6-21 运行时验证系统资源占用随时间变化测试结果表

启动后时间(s)	CPU 占用率	内存占用率
1	2.00%	1.90%
2	9.40%	1.90%
3	1.70%	1.90%
4	9.50%	1.90%
5	1.70%	1.90%
6	9.20%	1.90%
7	1.80%	1.90%
8	8.70%	1.90%
9	1.70%	1.90%
…	…	…
100	9.70%	1.90%
平均值	5.57%	1.90%

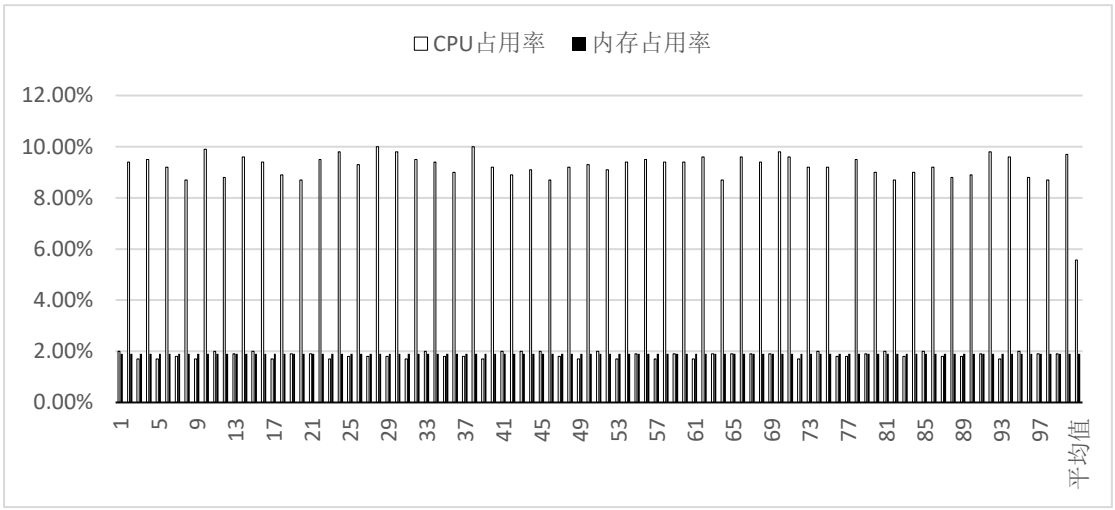

图 6-12 运行时验证系统资源占用随时间变化测试柱形图

从图中可以看出，随着时间的变化，运行时验证系统占用宿主机的内存资源均较小，但由于其需要处理各个来源的事件并计算验证，因此将保持占用一定的 CPU 资源。同时，运行时验证系统对资源的占用波动较小，整体表现稳定。

6.5 本章总结

本章对整个物联网服务系统运行时验证系统从功能和性能两方面进行了全方位的详细测试。为从系统部署到资源占用等均编写了相应的测试用例。根据测试结果可以说明，在功能性上，本文所实现的运行时验证系统可以满足第三章需求分析中提到的全面功能，同时在性能上该系统达到了高性能、高效率的要求。

第七章 总结与展望

7.1 工作总结

本文所研究的物联网服务系统运行时验证系统起源于作者所在实验室的长期探索。在对物联网服务系统的保障方面，已有的研究主要是将保障系统与物联网服务系统相融合，直接对其进行事件的获取与分析后反馈。但随着物联网服务系统对安全的要求日益提高，紧密融合的方式会造成保障系统难以达到较高程度的可信性。因此急需研究出一种更高级别的具有隔离性的保障方式。

本文依托实验室的研究方向，对物联网服务系统的保障提出了新的解决办法。通过将保障系统与物联网服务系统分离，将物联网服务系统置于封闭的虚拟机中，并对其与外界的通信渠道进行控制，从而以服务隔离的方式达到高可靠的保障。

对物联网服务系统与物理世界的交互渠道控制上，本文提出采用多种来源融合处理互相补偿的方法。一方面在网络通信与串口通信的控制上，本文均提出了独到的解决办法，以高效的方法对通信过程进行分析控制。其中对串口通信的处理，本文另辟蹊径，以串口中继器的方式解决了传统串口设备独占性问题。另一方面在多来源事件的融合分析中，本文考虑了隔离性带来的事件采集问题，提出了一系列创新性的事件融合与补偿算法。将三个观察源获取到的事件进行融合验证，对事件的重复、遗漏等情况进行高效应对。最后在事件的验证上，本文以模型状态机节点实例化的方式，结合增量式求解验证方法，将事件验证的整体效率大大提升，扩大了整个运行时验证系统的实际应用范围，增强了应用能力。

7.2 工作展望

本文所论述的物联网服务系统运行时验证系统集隔离性、高效性与一体，能为运行中的物联网服务系统提供高可靠的运行保障。针对本文所设计实现的系统，未来可以有如下几个方面的工作改进与延伸，例如：

1. 网络通信事件捕获模块在内核中运行时对事件的析取是通过应用层数据进行搜索的，将来可以为具体的应用层协议更改为精确的事件定向，从而进一步提高运行效率。

2. 本文所使用的包括 linux 内核模块、虚拟机内存自省技术等较为依赖具体的系统与软件版本。尽管实际实现已经尽可能地适配不同的版本，但 linux 与软件版本变化众多，难以全部兼容，因此将来可以考虑更换成更容易兼容的技术来代替此系统的技术选型。

参考文献

[1] 刘志硕, 魏凤, 柴跃廷, 等. 关于我国物联网发展的思考 [J]. 综合运输, 2010, (02): 37-40.

[2] Barringer, H., Goldberg, A., Havelund, K., et al. Rule-based runtime verification. In Verification, Model Checking, and Abstract Interpretation (VMCAI), number 2937 in LNCS. Springer-Verlag, 2004.

[3] Ligatti, J., Reddy, S.. A theory of runtime enforcement, with results. in Computer Security–ESORICS 2010. Springer, 2010, pp. 87–100.

[4] Gamage, T., McMillin, B., Roth, T.. Enforcing information flow security properties in cyber-physical systems: a generalized framework based on compensation. Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, 2010, pp, 158-163.

[5] Mitsch, S., Platzer, A.. ModelPlex: verified runtime validation of verified cyber-physical system models. Formal Methods in System Design, 49 (1-2) , 2016, pp. 33-74.

[6] 张硕, 贺飞. 运行时验证技术的研究进展 [J]. 计算机科学, 2014, 41(S2):359-363.

[7] 倪志敏, 赵凡. 基于虚拟化技术的网络安全管理的研究与实现 [J/OL]. 中国建材科技, http://kns.cnki.net/kcms/detail/11.2931.TU.20171101.1713.004.html, 2017.

[8] 赵常智. 基于运行时验证的软件监控关键技术研究[D]. 长沙: 国防科学技术大学, 2011.

[9] 潘进. 互联网安全中的安全事件验证[D]. 北京: 北京邮电大学, 2011.

[10] 文哲. 计算机虚拟化技术在企业中的应用[J/OL]. 电子技术与软件工程, (15):151(2017-08-03).http://kns.cnki.net/kcms/detail/10.1108.TP.20170803.1503.240.html, 2017.

[11] 崔泽永, 赵会群. 基于 KVM 的虚拟化研究及应用[J]. 计算机技术与发展, 2011, 21(06): 108-111+115.

[12] 刘风华, 丁贺龙, 张永平. 关于 NAT 技术的研究与应用[J]. 计算机工程与设计, 2006, (10):1814-1817.

[13] 杨金花. 虚拟机中网络连接问题的研究[J]. 电子设计工程, 2014, 22(03):77-79+82.

[14] 姚晓宇, 赵晨. Linux 内核防火墙 Netfilter 实现与应用研究[J]. 计算机工程, 2003, (08):112-113+163.

[15] 董昱, 马鑫. 基于 netlink 机制内核空间与用户空间通信的分析[J]. 测控技术, 2007, (09):57-58+60.
[16] 卢东, 陆以勤, 吕锦. 利用伪终端实现串行通信链路复用功能[J]. 微计算机信息, 2008, (02):74-75+73.
[17] 罗文华, 汤艳君. 基于 Volatility 的内存信息调查方法研究[J]. 中国司法鉴定, 2012, (04):90-93.
[18] 谭同超. 有限状态机及其应用[D]. 广州: 华南理工大学, 2013.
[19] 赵子都. 定理机器证明[J]. 自然辩证法研究, 1994, (05):46-50.
[20] 雷渊明. 基于 Netfilter 的包分类研究与设计[D]. 长沙: 湖南大学, 2009.
[21] 林龙增. KVM 虚拟机设备虚拟化及串口转发器的研究与应用[D]. 成都: 成都理工大学, 2012.
[22] 徐志强, 柴志雷, 须文波. JVM 实时内存管理模型的设计与实现[J]. 微计算机信息, 2009, 25(12):271-273.
[23] He Fang-jian. Network Modeling and Visualization Platform based on Moodle and Mxgraph[A]. International Research Association of Information and Computer Science.Proceedings of the 3rd International Conference on Applied Social Science Research(ICASSR 2015)[C]. International Research Association of Information and Computer Science:, 2015:3.
[24] 张变玲. IP 数据报在分组无线网中的传输[D]. 西安:西安电子科技大学, 2001.
[25] 池莹. 应用 Select 模型实现 TCP 并发服务器[J]. 科技广场, 2006, (04):30-32.
[26] 张恒若, 付明. 基于 Z3 的 Coq 自动证明策略的设计和实现[J]. 软件学报, 2017, 28(04):819-826.
[27] 熊伟, 丁涵, 罗云锋. 支持多线程并发与消息异步处理的 Linux Netlink 通信机制研究[J]. 软件导刊, 2017, 16(10):99-103.

致谢

两年半的研究生生涯即将结束了，从当初跨专业考研什么都不懂不会到现在能独当一面独立完成各个项目，整个研究生期间进步巨大。这首先要感谢我的导师章洋老师。章老师是我见到的最为认真负责的老师，对于学习工作的事均能非常细心耐心地对我们进行指导。没有章老师的指导，就没有我现在取得的成果，非常非常感谢老师！

其次要感谢已经毕业的华强师兄和戴弘扬师兄，他们给我的研究方向探索出了一条道路，让我能够沿着他们的道路继续走下去。

还要感谢同组的同学，尤其是和我研究内容相关的贺路路同学，和你们在技术上的互相探讨让我解决了很多疑问。

此外，感谢开源软件的贡献者们，是你们无私奉献地精神让计算机软件这个领域有了长足的发展。本文也用到了许多开源的技术，非常感谢原作者们的奉献。

感谢微软公司提供的 Windows10 操作系统和 office2016 办公软件，让我能方便且高效率地写完这篇论文。

感谢“坚果云”提供的文件版本控制与同步功能，让我在写论文时不用担心文件的异常等丢失情况。

感谢陪伴我五年半的联想笔记本电脑，在这关键的时候没有出任何差错，非常感谢！

最后感谢我的家人，我的女朋友，给我精神上的支持和充足的自由时间去完成这篇论文，谢谢你们！

攻读学位期间取得的研究成果

[1] 陈宽, 陈俊亮. 物联网服务系统运行时验证系统的研究与实现 [EB/OL]. 北京：中国科技论文在线 [2017-11-14].http://www2.paper.edu.cn/releasepaper/content/201711-46, 2017.