ICS 35.030
CCS I6440

团 体 标 准

T/JSHLW 003—2025

多模态工业互联网络主动认知内生安全技术规范

Technical Specifications for Active Cognitive Endogenous Security of Multimodal Industrial Internet Networks

2025－05－06 发布　　2025－06－06 实施

江苏省互联网协会　　发 布

目录

目录 I
前言 II
多模态工业互联网络主动认知内生安全技术规范 1
1 范围 1
2 规范性引用文件 1
3 术语、定义和缩略词 1
3.1 术语和定义 1
3.2 缩略词 3
4 系统架构 4
4.1 概述 4
4.2 工业网络设备可信身份认证与访问控制平台架构 4
4.3 DoS/DDoS 攻击检测与防御平台架构 4
5 工业网络设备可信身份认证与访问控制平台 5
5.1 概述 5
5.2 设备管理模块 5
5.3 数据管理模块 6
5.4 用户管理模块 6
5.5 安全和认证模块 6
6 DoS/DDoS 攻击检测与防御平台 7
6.1 概述 7
6.2 高速流量检测以及防御模块 7
6.3 分布式异常数据学习模块 8
6.4 异常数据边缘检测模块 8
7 关键技术 9
7.1 多模态工业互联网络主动认知内生安防系统技术体系 9
7.2 工业网络设备可信身份认证与访问控制平台关键技术 9
7.3 DoS/DDoS 攻击检测与防御平台关键技术 10
8 接口开发规范 11
附 录 A （资料性） 工业网络设备可信身份认证与访问控制平台接口规范 12
附 录 B （资料性） 可编程交换机原语代码 21

前言

本文件按照 GB/T 1.1-2020《标准化工作导则第 1 部分：标准化文件的结构和起草规则》规则起草。

本文件规定了多模态工业互联网络主动认知内生安防系统的系统架构、平台模块、关键技术等的要求，以及对平台接入接口协议、交互数据格式以及适配系统的基础硬件需求提出了规范性要求。

本文件由江苏省互联网协会标准化技术委员会提出并归口。

本文件起草单位：东南大学、中国科学技术大学、清华大学、联想（北京）有限公司、北京泰豪智能工程有限公司。

本文件主要起草人: 熊润群、张兰、刘云浩、张华俊、孙鹏程、叶洪宇、王鑫铭、蒲佳杭、党凡、王需、单冯、丁玎、陈慈媛、徐祝庆、谢玮、过晓冰、马益荣。

多模态工业互联网络主动认知内生安全技术规范

1 范围

本文件规定了多模态工业互联网络主动认知内生安防系统的系统架构、平台模块、关键技术和接口开发规范等。

本文件适用于多模态工业网络设备可信身份认证与访问控制平台、DoS/DDoS 攻击检测与防御平台的应用开发。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 22080 信息技术 安全技术 信息安全管理体系 要求

GB/T 25062 信息安全技术 鉴别与授权 基于角色的访问控制模型与管理规范

GB/T 30279 信息安全技术 网络安全漏洞分类分级指南

GB/T 31491 无线网络访问控制技术规范

GB/T 37027 信息安全技术 网络攻击定义及描述规范

GB/T 42573 信息安全技术 网络身份服务安全技术要求

GB/T 43779 网络安全技术 基于密码令牌的主叫用户可信身份鉴别技术规范

YD/T 3844 工业互联网平台应用管理接口要求

AII/001 工业互联网平台 通用要求

AII/004 工业互联网平台 安全防护要求

3 术语、定义和缩略词

3.1 术语和定义

GB/T 22080、GB/T 25062、GB/T 30279、GB/T 31491、GB/T 37027、GB/T 43779、YD/T 3844、AII/001、AII/004 界定的以及以下术语和定义适用于本文件。

3.1.1

多模态工业互联网络 Multimodal Industrial Internet

通过融合图像、文本、语音、传感器信号等多模态数据，构建设备、系统和人之间的高效协同与智能交互的工业网络体系。

3.1.2

工业互联网平台 Industrial Internet Platform

集成工业数据采集、处理、分析与应用开发功能，提供智能化生产、运营优化和资源协同支持，连接设备、网络和应用的核心枢纽。

3.1.3

采集模块 Acquisition Module
从传感设备通信接口获取信息的计算机程序。

3.1.4

表述性状态转移 Representational State Transfer
降低开发复杂性，提高系统可伸缩性网络应用设计和开发方式的一组架构约束条件和原则。

3.1.5

远程过程调用 Remote Procedure Call
通过网络从远程计算机程序上请求服务，采用接口调用远程服务进程中函数的方式。

3.1.6

角色身份 Principals
采用用户名、邮箱等标识形式的主体标识属性。

3.1.7

证明/凭证 Credentials
主体持有的、只有其知晓的密码、数字证书等安全值。

3.1.8

工业串行通信协议 Modbus TCP/IP
运行在 TCP/IP 协议上的 Modbus 工业串行通信协议。

3.1.9

拒绝服务攻击 Denial of Service Attack
任何使当前服务可用性降低或失去可用性的干涉。

3.1.10

分布式拒绝服务攻击 Distributed Denial of Service Attack
处于不同位置的多个攻击者同时向一个或数个目标发动拒绝服务攻击，或一个攻击者控制位于不同位置的多台机器并对受害者同时实施拒绝服务攻击。

3.1.11

原语防御设计 Primitive Defense Design
通过将复杂的防御操作封装为功能模块、原语构建稳固、安全的基础结构，降低复杂性和潜在安全风险。

3.1.12

深度学习 Deep Learning
利用多层神经网络模型通过大量数据自动特征提取和模式识别，解决图像识别、语音处理等复杂任务，获得比传统方法更高的准确度和鲁棒性的机器学习。

3.1.13

整数线性规划 Integer Linear Programming

决策变量为整数，目标函数和约束条件为线性，用于生产调度、资源分配、运输问题和网络设计等实际问题中的一类优化问题。

3.1.14

联邦学习 Federated Learning

在多个设备或数据源间联合建模，无需集中收集或共享数据，数据保留在本地设备上，通过在本地训练模型并仅共享权重等模型参数或梯度学习，保护数据隐私和安全的分布式机器学习方法。

3.1.15

设备权限注册 Device Permission Registration

注册设备在安全环境中运行需要的时间、空间、网络环境等条件。

3.1.16

角色访问控制 Role-based Access Control

通过将访问权限分配给角色，将角色分配给用户，实现对资源访问控制的权限管理方法。

3.1.17

时空频信息 Spatial-Temporal-Frequential Information

同时涉及时间维度、空间维度和频率维度的信息特征，描述数据或事件在时间、空间和频域上的分布与变化规律的信息。

3.1.18

机用户 Machine User

由系统或应用程序代表服务自动执行任务的非人类用户实体，用于系统间通信或自动化操作的用户。

3.1.19

卷积神经网络 Convolutional Neural Network

通过卷积操作自动提取输入数据的局部特征，应用于图像识别、语音处理等领域的深度学习模型。

3.2 缩略词

下列缩略语适用于本文件。

CNN，卷积神经网络（Convolutional Neural Network）

CPU，中央处理器（Central Processing Unit）

DDoS，分布式拒绝服务（Distributed Denial of Service）

DoS，拒绝服务（Denial of Service）

ILP，整数线性规划（Integer Linear Programming）

P4，专为网络设备设计的编程语言（Programming Protocol-Independent Packet Processors）

SSL，安全套接层 (Secure Sockets Layer)

TLS，传输层安全 (Transport Layer Security)

TSN，时间敏感网络（Time-Sensitive Networking）
RBAC，角色访问控制（Role-Based Access Control）
SHA-2，安全散列算法 2（Secure Hash Algorithm 2）

4 系统架构

4.1 概述

多模态工业互联网络主动认知内生安防系统应从设备层、数据层与服务层三个层次构建协同防护体系。在设备层和数据层，应依托工业网络设备可信身份认证与访问控制平台，构建基于多模态感知、指纹提取和一致性验证的可信身份认证与访问控制机制；在服务层，应依托DoS/DDoS攻击检测与防御平台，实现面向工业服务的异常检测与主动防御。系统整体防护策略应实现由点防御向面防御的转变，形成多层次、多维度的主动防御和智能协同安全体系。

4.2 工业网络设备可信身份认证与访问控制平台架构

该架构如图1所示，用于实现工业网络设备的可信身份认证与访问控制，提供统一的认证机制和策略支撑：

a) 该架构通过多模态数据感知技术对不同模态的设备数据分析；
b) 通过多维度指纹提取技术提取难以伪造的可靠终端设备标识；
c) 通过时空频一致性验证技术基于多维度、多层次获取的模态数据的时间、空间和频域语义信息提供可靠的设备可信身份认证；
d) 该架构应支持 Modbus、Ethernet、Ethercat、Profinet 和 TSN 等模态设备的可信认证。

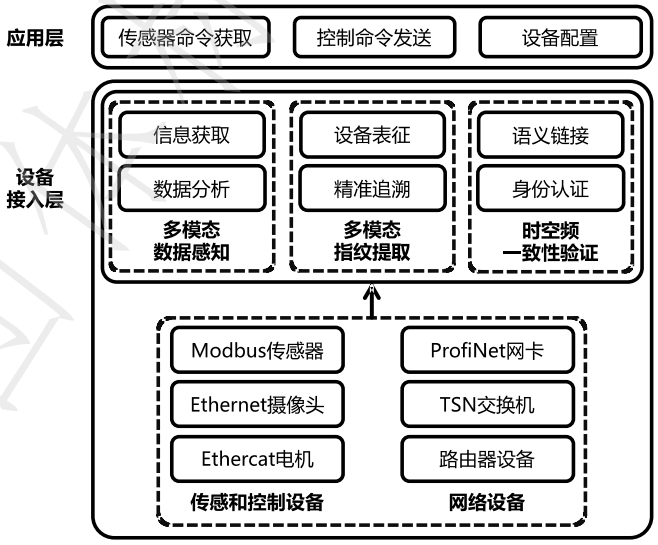

图1 工业网络设备可信身份认证与访问控制平台架构

4.3 DoS/DDoS 攻击检测与防御平台架构

该架构如图2所示，用于实现工业网络设备的DoS/DDoS攻击分层次安全防御，提供多维度流量监控、异常检测与智能防御策略部署能力：

a) 通过中心服务层、边缘服务层和可编程交换机协同，实现对网络设备的多维监控与流量检测，支持模型更新、策略管控与数据统计等功能，用于构建多层次、多模块协同的终端设备安全防护体系；

b) 中心服务器处于防御平台的最上层，上部署全局检测模型，承担下层边缘服务器的模型聚合、更新以及测试，同时直接监控整个网络，对应做出防御策略的调整；

c) 边缘服务器部署本地检测模型，利用硬件接口接收下层可编程交换机转发的可疑流量进行检测和防御，同时存储部分下层的防御程序；

d) 可编程交换机位于防御平台的最下层，直接接入被监控网络，利用硬件接口接收网络流量，并根据部署的防御程序进行流量检测和防御，借助与控制平面的通信进行防御策略的调整，存储部分策略到上层的边缘服务器当中。

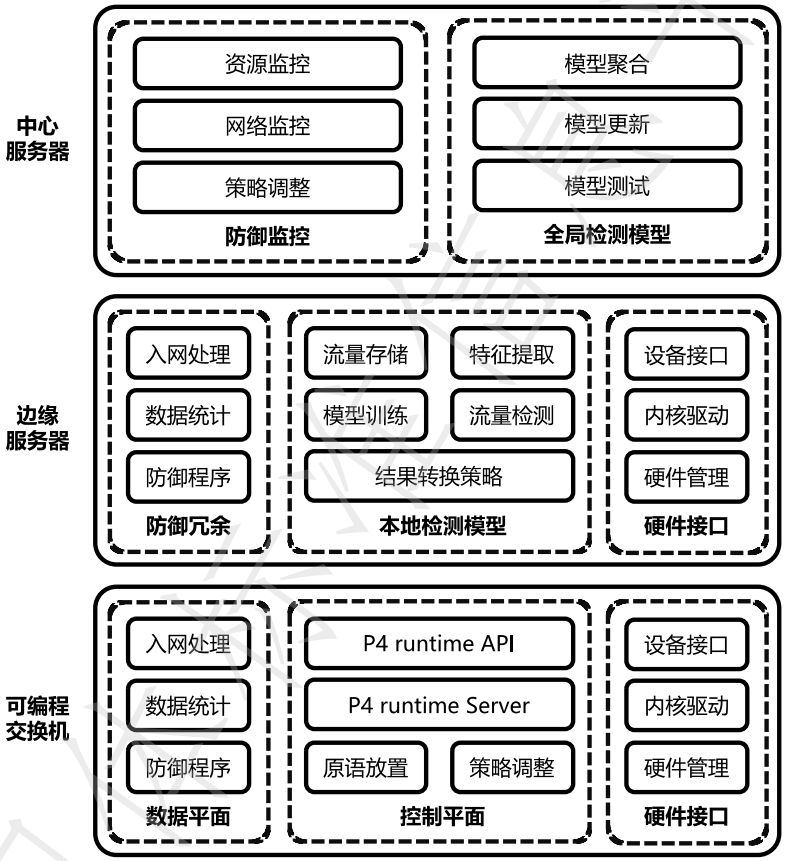

图2. DoS/DDoS攻击检测与防御平台架构

5 工业网络设备可信身份认证与访问控制平台

5.1 概述

工业网络设备可信身份认证与访问控制平台总体要求包括：

a) 平台模块应为可信身份认证与访问控制平台提供系统支撑和保障，支持工业设备和角色接入，涵盖设备管理、数据管理、服务管理和安全访问控制等基础功能；

b) 基础框架宜采用 Spring 架构，可扩展至 Spring Boot 框架，或基于 Spring Cloud 实现分布式服务；

c) 平台应包括设备管理模块、数据管理模块、用户管理模块以及安全与认证模块。

5.2 设备管理模块

本模块要求包括：

a) 应支持管理端自定义的设备注册认证、设备信息的增删改操作，并基于多层级用户角色、设备时间、空间以及频域一致性，实现设备权限管理功能；
b) 应具备设备权限注册与验证、权限分组与统计、设备操作审核、运行维护记录查询、可视化实时监控等功能；
c) 应支持基于国产密码方案的加密传输；
d) 应实现设备统一接入管理，并对可能遭受入侵的设备采用物理隔离。
e) 接口设计示例见附录 A.1。

5.3 数据管理模块

本模块要求包括：
a) 应通过建立统一的数据访问接口，屏蔽底层异构的数据格式和存储方式，应使得数据访问操作实现统一化；
b) 应使用数字签名、脱敏处理、加密等技术手段。
c) 接口设计示例见附录 A.2 用户管理服务接口、附录 A.3 认证管理服务接口、附录 A.4 安全防御服务接口。

5.4 用户管理模块

本模块要求包括：
a) 应支持用户创建、用户审核、角色创建、角色权限分配、用户角色分配、用户信息更改及用户信息展示等基础功能，并应 RBAC 技术实现用户对资源的限制访问；
b) 在用户管理服务中，用户审核注册、用户角色分配和角色权限分配应符合 RBAC 规范；
c) 用户管理系统对象应包括工业设备和系统等资源、权限、用户和角色；
d) 访问策略应包括角色访问策略和设备访问策略，只有同时符合这两种访问策略，用户才能访问相应的设备资源；
e) 角色权限分配应将权限与角色映射，用户角色分配应将用户与角色关联，实现用户对资源的控制访问。
f) 接口设计示例见附录 A.2 用户管理服务接口。

5.5 安全和认证模块

5.5.1 接口设计

本模块接口设计要求如下：
a) 应具备工业网络设备的可信身份认证与访问控制平台用户和设备数据保护、身份认证和访问控制等功能，应包括安全类接口和认证类接口。
b) 接口设计示例见附录 A.1 设备管理模块、A.3 认证管理服务接口、A.4 安全防御服务接口。

5.5.2 安全类接口

安全类接口应包括下列接口：
a) 身份认证类应包含获取认证信息、上传认证信息等接口；
b) 权限管理类应包含获取权限、授予权限、删除权限等接口；
c) 访问控制类应包含获取权限资源等接口；
d) 密钥管理类应包含密钥上传、密钥删除等接口；
e) 数据加解密类应包含数据加密、数据解密等接口；

f) 数据保护类应包含数据脱敏、数据审查、数据安全防御接口；
g) 业务操作合法性类应包含业务行为监测、分析、建模，异常行为预警接口；
h) 业务连续性类应包含数据备份、业务平台容灾接口；
i) 安全管理类应包含事件集中管理、安全态势、通报预警接口；
j) 安全处置类应包含安全演练、自动化智能处置接口；
k) 接口可选择支持 SSL、TLS、SHA2 等多种加密协议和算法。

5.5.3 设备认证接口

设备认证接口应包含获取认证信息、删除认证信息、验证认证信息等接口。

6 DoS/DDoS 攻击检测与防御平台

6.1 概述

DoS/DDoS 攻击检测与防御平台总体要求包括：
a) DoS/DDoS 攻击检测与防御平台应用于网络流量检测，支持低延时、高精度网络流量检测，且应保证面对新型的攻击方式快速检测并且做出防御措施；
b) 平台应保证具备可编程架构的一台交换机，以及多台服务器组成边缘服务器集群；
c) 基础服务模块应包括高速流量检测以及防御模块、分布式异常数据学习模块、异常数据检测模块；
d) 集群数量应满足 3 台及以上。

6.2 高速流量检测以及防御模块

本模块要求包括：
a) 应由可编程交换机提供支持；
b) 应在网络当中以线速度分析经过交换机等指定链路的流量数据，应支持根据预写入分析策略对网络数据包分析判断，并根据分析结果对数据包进行特定处理；
c) 对于当前策略无法判断的流量数据，应将该流量转发至边缘服务器做进一步判断；
d) 应至少支持下列流量转发操作：
①Drop：丢弃数据包，可用于异常流量；
②Forward：正常转发数据包，可用于正常流量；
③Confuse：针对无法判断的数据包，将该数据包转发至边缘服务器；
e) 应具备实时更新分析策略功能，提供更改策略的接口；
f) 可编程交换机应对数据包解析，并应符合表 1-表 3 的规定。

表 1 以太网标头 ethernet_t **单位：bit**

解析含义	解析长度
目标 MAC 地址	48
源 MAC 地址	48
以太网类型	12

表 2 IPv4 标头 ipv4_t **单位：bit**

解析含义	解析长度
源 IP 地址	32

目的 IP 地址	32
版本	4
IP 报文头部长度	4
优先级、延迟等服务质量控制	8
整个 IP 数据包的长度	16
标识分片数据包的 ID	16
标志位	3
当前数据包在数据流中的位置	13
数据包生存时间	8
IP 数据包的上层协议类型	8
IP 头部校验和	16

表 3 TCP 标头 tcp_t **单位：bit**

解析含义	**解析长度**
目的端口	16
源端口	16
seq 序列号	32
ack 序列号	32
TCP 头部长度的数据偏移量	4
保留位	4
标志位	1
接收窗口	16
校验和	16
紧急指针	16

6.3 分布式异常数据学习模块

本模块要求包括：

a) 应由多台服务器构成的集群提供支持；

b) 应收集网络当中新出现的高速流量检测以及防御模块根据现有策略无法正确识别的数据包等异常数据，并作为训练样本进行模型训练迭代，通过将新的模型部署到异常数据边缘检测模型处保证平台始终使用最贴合当前网络环境的异常检测模型；

c) 应采用联邦学习架构，边缘服务器本地训练模型，只向中心服务器上传训练得到的参数，中心服务器经微调后下发新的模型参数。

6.4 异常数据边缘检测模块

本模块要求包括：

a) 应由多台服务器构成的边缘服务器集群构成；

b) 应对高速流量检测以及防御模块无法根据已有策略分析的流量数据进行判断，该功能应通过部署于服务器上的机器学习的随机森林模型、深度学习的 CNN 模型等异常检测模型实现，通过预训练的模型对流量分类，再根据分类结果制定对应的转发操作，并更新高速流量检测以及防御模块当中的分析策略；

c) 应支持 P4 Runtime API 以接收网络数据包以及更新高速流量检测和防御模块当中的策略，同

时在硬件层面上，应保证能够正常部署满足内存以及 CPU 计算能力等的异常检测模型。

7 关键技术

7.1 安防系统

安防系统构成应包含可信身份认证、精细访问控制和协同防御攻击关键技术，如图3所示。

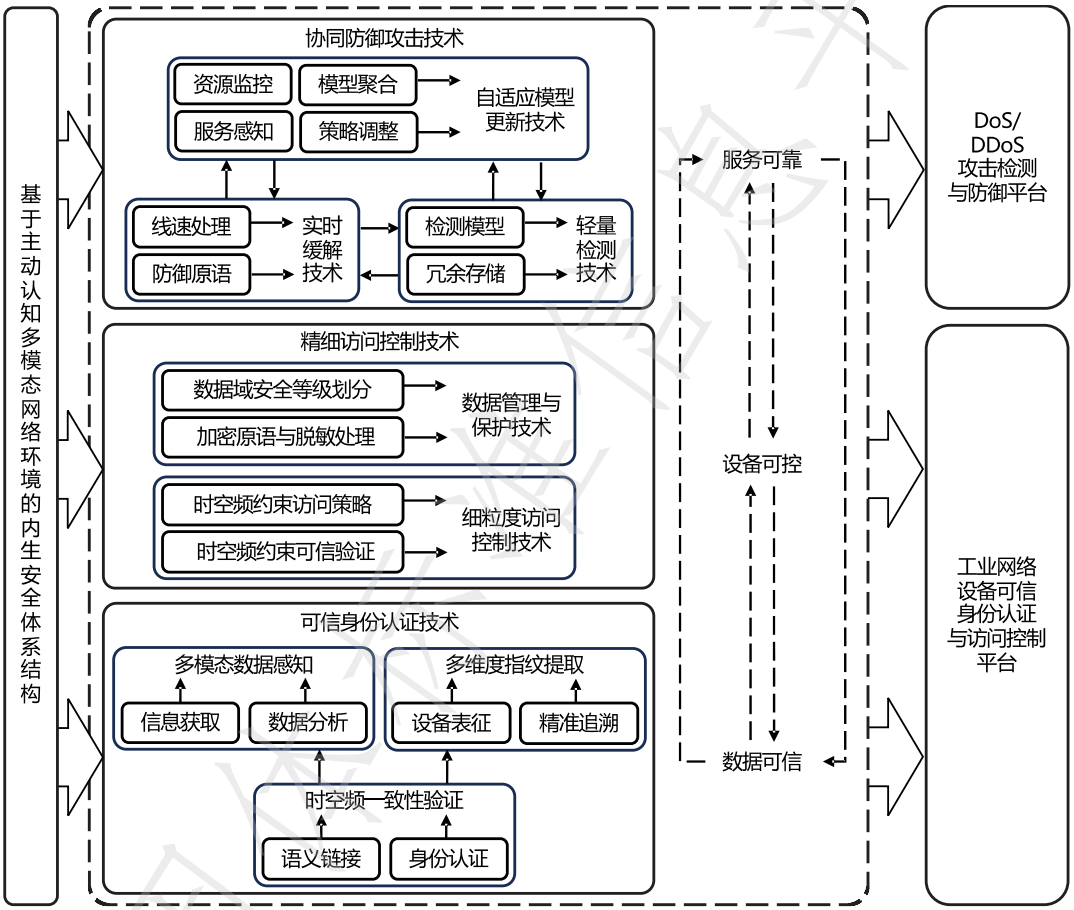

图3 安防系统构成

7.2 关键技术

7.2.1 一般要求

在分析工业网络设备可信身份认证与访问控制平台基础服务的基础上，应针对恶意设备接入和访问权限滥用的威胁，提出基于多模多维信息一致性的可信身份认证技术和基于细粒度时空频约束的精细访问控制技术。并应满足下列要求：

a) 终端设备验证应借助根据设备不可篡改、不可克隆的物理特性提取的与通信内容无关的唯一指纹信息，同时将多维度，多层次的数据与设备的身份行为相结合形成多维度的身份认证；

b) 用户访问控制应根据数据多维信息，按用户身份与行为特征动态划分数据安全等级，生成可自适应调整的访问策略。

7.2.2 身份认证

7.2.2.1 多维度指纹提取和验证技术

该技术要求包括：

a) 应基于终端设备不可篡改或克隆的物理特征，提取跨环境、跨设备的物理层信号指纹；
b) 提炼的信号指纹应与通信内容无关，应按终端系统特征，提供难以伪造的可靠终端标识，现快速、实时的特征匹配与验证。

7.2.2.2 多模态数据精准自适应感知技术

应根据工业互联网感知数据分析结果自适应调整感知模式，对不同模态数据和设备物理身份及网络行为的相关性建模分析，得到多模态感知数据在时空频多个维度上的语义连接。

7.2.2.3 时空频信息一致性验证技术

时空频信息一致性验证应多维度、多层次地获取模态数据的时间、空间和频域语义信息。

7.2.3 基于细粒度时空频约束的精细访问控制技术

7.2.3.1 时空频数据域安全等级划分技术

该技术要求包括：

a) 应基于数据生成时间、数据类型、语义等数据模态进行数据安全评估；
b) 应根据数据安全评估，结合系统架构和设备功能的安全性以及数据泄露风险，生成动态安全等级调整策略。

7.2.3.2 细粒度时空频约束访问策略技术

该技术要求包括：

a) 应支持使用机器学习或统计方法分析不同时空频条件下，不同用户身份和属性的访问频次和访问效率；
b) 应根据分析结果自动化分组用户、资源，并优化访问控制策略。

7.2.3.3 时空频约束实时可信验证技术

时空频约束实时可信验证应验证访问行为当中的时间、空间、频域信息、以及访问者的身份信息。

7.3 DoS/DDoS 攻击检测与防御平台关键技术

7.3.1 一般要求

在分析面向工业互联网络 DoS/DDoS 攻击检测与防御平台服务的基础上，应针对工业场景异构和动态的 DoS/DDoS 攻击，提出基于联邦学习架构的 DoS/DDoS 检测技术和基于可编程交换机的 DoS/DDoS 缓解技术。并应满足下列要求：

a) 低延迟的网络流量实时检测应依托于可编程交换机，可编程交换机通过专用集成电路设计，允许用户通过 P4 程序直接在网络层实现自定义操作；
b) 应包括对复杂 P4 程序进行原语级别的封装设计、可编程交换机内部的自适应调整防御机制，以及基于整数规划的最佳原语放置技术。

7.3.2 基于联邦学习架构的 DoS/DDoS 检测技术

7.3.2.1 资源服务感知和流特征分析技术

该技术要求包括：

a) 应监控工业网络资源和感知工业设备服务，确定工业网络内的潜在 DoS/DDoS 威胁；
b) 应对流入工业设备的网络流量进行完整特征提取，除网络数据包特征，还应针对工业数据包提取工业应用和工业协议作为特征。

7.3.2.2 轻量级本地模型检测技术

该技术要求包括：
a) 应引入 CNN 本地检测模型，训练时重复使用过滤器中的权重，减小学习参数的数量，减少边缘服务器部署模型的计算资源和内存资源；
b) 应以一段时间内的网络数据流特征矩阵为输入，利用 CNN 模型分析同一数据流的数据包之间的相关性，对流量多分类。

7.3.2.3 自适应联邦学习模型更新技术

该技术要求包括：
a) 应根据边缘设备本地模型在其私有验证集上的表现动态调整其参与全局模型聚合的频率和训练计算量；
b) 应支持每轮动态调整参与边缘服务器的本地训练步数，避免对全体边缘服务器施加统一计算负担，降低资源浪费和不必要的计算延迟。

7.3.3 基于可编程交换机的 DoS/DDoS 缓解技术

7.3.3.1 P4 程序模块化原语封装技术

该技术要求包括：
a) 应按功能模块的划分封装 P4 代码形成原语，原语替代 P4 代码作为最小的编程单元，参与构建上层的模块以及平台。
b) 示例原语封装设计见附录 B。

7.3.3.2 自适应防御策略调整技术

该技术要求包括：
a) 应支持在可编程交换机上同时部署多种防御策略；
b) 应支持实时统计当前数据包的特征数据，并根据当前呈现的不同特征自适应地启用不同的防御策略。

7.3.3.3 基于整数规划的最佳原语放置技术

该技术要求包括：
a) 应将原语部署问题建模为 0-1 ILP 问题，并通过求解器得到最佳原语放置方案；
b) 应根据最佳原语放置方案在可编程交换机上部署原语，并在可编程交换机资源饱和时，进行原语调度，将部分原语卸载到边缘服务器。

8 接口开发规范

平台之间、模块之间接口开发应遵循下列原则：
a) 单一职责原则：每个接口应专注于特定的验证任务，便于工业物联网设备状态的追踪和审计，同时对访问控制和权限管理。出现安全事故时，可快速定位到具体的验证环节；

b) 开放封闭原则：接口设计应分析未来的扩展性，新增安全要求、验证机制或设备类型的加入，不应影响现有的验证流程；
c) 里氏替换原则：在升级验证机制时，新验证方式应能够完全覆盖旧的验证场景，并保证系统的兼容性；
d) 接口隔离原则：不同类型设备和不同级别验证需求应使用不同的接口，应最小化每个接口的攻击面，同时便于实施细粒度的安全控制；
e) 依赖倒置原则：接口应依赖于抽象的安全策略和验证规则。当安全策略需要调整时，仅需修改策略定义，无需改变接口本身；
f) 最少知识原则：验证接口应仅暴露必要的信息，内部的验证细节和安全策略执行过程应对外部不可见。调用方只需了解如何正确使用接口验证。

附 录 A
（资料性）
工业网络设备可信身份认证与访问控制平台接口规范

A.1 设备管理服务接口

A.1.1 工业互联网设备加入

设备加入接口设计见表 A.1.1。

表 A.1.1 设备加入接口设计

接口路由地址	/industry/createDevice
接口功能	从管理端加入对应工业互联网设备
参数	realm(String) 域 name(String) 设备名字 remark(String) 设备备注 type(String) 网络类型 id(String) 设备 id fingerprint(String) 设备指纹
返回	无

A.1.2 工业互联网设备删除

设备删除接口设计见表 A.1.2。

表 A.1.2 设备删除接口设计

接口路由地址	/industry/delDevice
接口功能	从管理端删除对应工业互联网设备
参数	id(String) 设备 id
返回	无

A.1.3 工业互联网设备更新

设备更新接口设计见表 A.1.3。

表 A.1.3　设备更新接口设计

接口路由地址	/industry/updateDevice
接口功能	从管理端更新工业互联网设备状态
参数	id(String) 设备 id name(String) 设备名字 remark(String) 设备备注
返回	无

A.1.4 工业互联网设备列表展示

设备列表展示接口设计见表 A.1.4。

表 A.1.4　设备列表展示接口设计

接口路由地址	/industry/listDevice
接口功能	从管理端获取工业互联网设备列表
参数	offset(Integer) 页面偏移量 limit(Integer) 页面大小
返回	res(Result<List<IndNetworkDeviceRes>>) 物联网设备列表

A.1.5 时空频信息创建

时空频信息创建接口设计见表 A.1.5。

表 A.1.5　时空频信息创建接口设计

接口路由地址	/tsf/createTsf
接口功能	创建人员或设备对应的时空频信息
参数	type(Byte) 指名是设备还是人员 realm(String) 域 devId(String) 设备 id userId(String) 用户 id time(String) 时间 space(String) 空间
返回	无

A.1.6 时空频信息删除

时空频信息删除接口设计见表 A.1.6。

表 A.1.6　时空频信息删除接口设计

接口路由地址	/tsf/delTsf
接口功能	删除人员或设备对应的时空频信息
参数	id(String) 时空频 id
返回	无

A.1.7 时空频信息更新

时空频信息更新接口设计见表 A.1.7。

表 A.1.7　时空频信息更新接口设计

接口路由地址	/tsf/updateTsf
接口功能	更新时空频信息
参数	id(String) 时空频 id time(String) 时间 space(String) 空间
返回	无

A.1.8 时空频信息列表展示

时空频信息列表展示接口设计见表 A.1.8。

表 A.1.8　时空频信息列表展示接口设计

接口路由地址	/tsf/listTsf
接口功能	从管理端获取时空频列表
参数	offset(Integer) 页面偏移量 limit(Integer) 页面大小
返回	res(Result<List<TsfRes>>) 时空频列表

A.2 用户管理服务接口

A.2.1 部门创建

部门创建接口设计见表 A.2.1。

表 A.2.1　部门创建接口设计

接口路由地址	/org/createOrg
接口功能	创建部门
参数	name(String) 部门名字 chargeUserId(String) 创建用户名 remark(String) 部门说明
返回	无

A.2.2 部门删除

部门删除接口设计见表 A.2.2。

表 A.2.2　部门删除接口设计

接口路由地址	/org/delOrg
接口功能	删除部门
参数	id(String) 部门 id delSubTree(int) 子部门删除策略

返回	无

A.2.3 部门更新

部门更新接口设计见表 A.2.3。

表 A.2.3　部门更新接口设计

接口路由地址	/org/update
接口功能	更新部门信息
参数	name(String) 部门名字 chargeUserId(String) 修改用户名 remark(String) 部门说明
返回	无

A.2.4 部门中添加用户

部门添加用户接口设计见表 A.2.4。

表 A.2.4　部门添加用户接口设计

接口路由地址	/org/add/user
接口功能	部门中添加用户
参数	orgId 部门 id userIds(Set<String>) 添加用户 id
返回	无

A.2.5 部门中移除用户

部门中移除用户接口设计见表 A.2.5。

表 A.2.5 部门中移除用户接口设计

接口路由地址	/org/remove/user
接口功能	部门中移除用户
参数	orgId 部门 id userIds(Set<String>) 移除用户 id
返回	无

A.2.6 查询机用户列表

查询机用户列表接口设计见表 A.2.6。

表 A.2.6　查询机用户列表接口设计

接口路由地址	/serviceUser/list
接口功能	查询机用户列表
参数	q(String) 查询机用户名 roleId(String) 角色身份 offset(Integer) 页面偏移量

	limit(Integer) 页面大小
返回	res(Result<List<ServiceUserListResVo>>) 机用户列表

A.2.7 创建机用户

创建机用户接口设计见表 A.2.7。

表 A.2.7 创建机用户接口设计

接口路由地址	/serviceUser/create
接口功能	创建机用户
参数	serviceName(String) 机用户名称 remark(String) 机用户描述
返回	无

A.2.8 更新机用户

更新机用户接口设计见表 A.2.8。

表 A.2.8 更新机用户接口设计

接口路由地址	/serviceUser/update
接口功能	更新机用户
参数	serviceName(String) 机用户名称 remark(String) 机用户描述 secret(String) 密钥
返回	无

A.2.9 删除机用户

删除机用户接口设计见表 A.2.9。

表 A.2.9 删除机用户接口设计

接口路由地址	/serviceUser/del
接口功能	删除机用户
参数	id(String) 用户 id
返回	无

A.2.10 机用户添加角色

添加机用户角色接口设计见表 A.2.10。

表 A.2.10 添加机用户角色接口设计

接口路由地址	/serviceUser/add/role
接口功能	机用户添加角色
参数	serviceUserId(String) 机用户 id roleIds(Set<String>) 角色 id

返回	无

A.2.11 机用户删除角色

删除机用户角色接口设计见表 A.2.11。

表 A.2.11　删除机用户角色接口设计

接口路由地址	/serviceUser/remove/role
接口功能	机用户删除角色
参数	serviceUserId(String) 机用户 id roleIds(Set<String>) 角色 id
返回	无

A.2.12 每日用户数统计

每日用户数统计接口设计见表 A.2.12。

表 A.2.12　每日用户数统计接口设计

接口路由地址	/stat/daily/users
接口功能	每日用户数统计
参数	startDate(String) 开始日期 endDate(String) 结束日期
返回	res(Result<List<Integer>>) 用户数量

A.2.13 查询热门认证资源

热门认证资源查询接口设计见表 A.2.13。

表 A.2.13　热门认证资源查询接口设计

接口路由地址	/stat/top/resources
接口功能	热门认证资源查询
参数	TopResourcesId(List<String>) 资源 id
返回	res(Result<List<TopResourcesStatResVo>>) 热门认证资源

A.2.14 每日登录次数统计

每日登录次数统计接口设计见表 A.2.14。

表 A.2.14　每日登录次数统计接口设计

接口路由地址	/stat/daily/logins
接口功能	每日登录次数统计
参数	startDate(String) 开始日期 endDate(String) 结束日期 type(int) 资源类型
返回	res(Result<List<Integer>>) 登录次数统计

A.2.15 认证行为地理位置分布

认证行为地理位置分布查询接口设计见表 A.2.15。

表 A.2.15　认证行为地理位置分布查询接口设计

接口路由地址	/stat/geo/logins
接口功能	认证行为地理位置分布
参数	date(String) 日期
返回	res(Result<List<LoginsGeoStatResVo>>) 地理位置

A.3 认证管理服务接口

A.3.1 工业互联网设备认证

工业物联网设备认证接口设计见表 A. 3. 1。

表 A. 3. 1　工业物联网设备认证接口设计

接口路由地址	/devAuth
接口功能	设备认证
参数	type(Integer) 通信协议类型
返回	res(Result<Object>) 设备认证结果

A.3.2 人员认证

人员认证接口设计见表 A. 3. 2。

表 A. 3. 2　人员认证接口设计

接口路由地址	/userAuth
接口功能	人员认证
参数	userName(String) 用户名 Password(String) 密码
返回	res(Result<Boolean>) 人员认证结果

A.3.3 获取存在的设备类型

获取存在的设备类型接口设计见表 A. 3. 3。

表 A. 3. 3　获取存在的设备类型接口设计

接口路由地址	/getExistTypes
接口功能	获取存在的设备类型
参数	无
返回	res(Result<List<Integer>>) 存在的设备类型

A.3.4 远程启动电机

远程启动电机接口设计见表 A. 3. 4。

表 A. 3. 4　远程启动电机接口设计

接口路由地址	/startMotor
接口功能	启动电机
参数	id(String) 设备 id
返回	无

A.3.5 远程关闭电机

远程关闭电机接口设计见表 A. 3. 5。

表 A. 3. 5　远程关闭电机接口设计

接口路由地址	/stopMotor
接口功能	关闭电机
参数	无
返回	无

A.4 安全防御服务接口

A.4.1 工业互联网设备负载查询

工业互联网设备负载查询接口设计见表 A. 4. 1。

表 A. 4. 1　工业互联网设备负载查询接口设计

接口路由地址	/loadQuery
接口功能	检查设备的运作情况
参数	无
返回	res(Result<List<Integer>>) 设备的工作负载（例如：cpu 占用率）

A.4.2 远程开启防御策略

远程开启防御策略接口设计见表 A. 4. 2。

表 A. 4. 2　远程开启防御策略接口设计

接口路由地址	/startDefense
接口功能	开启防御策略
参数	无
返回	无

A.4.3 远程变更防御策略

远程变更防御策略接口设计见表 A. 4. 3。

表 A. 4. 3　远程变更防御策略接口设计

接口路由地址	/updatePolicy
接口功能	变更可编程交换机上的防御原语

参数	attackClass(String) 攻击类型
返回	无

A.4.4 查询检测模型实时检测结果

查询检测模型实时检测结果接口设计见表 A. 4. 4。

表 A. 4. 4　查询检测模型实时检测结果接口设计

接口路由地址	/modelQuery
接口功能	查询模型对当前流量的分析结果
参数	无
返回	flowClass(String) 流量种类

A.4.5 查询防御策略执行

查询防御策略执行接口设计见表 A. 4. 5。

表 A. 4. 5　查询防御策略执行接口的设计

接口路由地址	/workQuery
接口功能	查询当前防御策略的执行具体情况
参数	无
返回	res(Result<List<WorkStatusTable>>)　开启的防御原语以及对应的命中次数列表

附 录 B
（资料性）
可编程交换机原语代码

B.1 udp-flood defense

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

/* CONSTANTS */

const bit<16> TYPE_IPV4 = 0x800;
const bit<8>    TYPE_TCP    = 6;
const bit<8>    TYPE_UDP    = 17;

#define BLOOM_FILTER_ENTRIES 4096
#define BLOOM_FILTER_BIT_WIDTH 1


typedef bit<9>    egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;


header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>     etherType;
}

header ipv4_t {
    bit<4>      version;
    bit<4>      ihl;
    bit<8>      diffserv;
    bit<16>     totalLen;
    bit<16>     identification;
    bit<3>      flags;
    bit<13>     fragOffset;
    bit<8>      ttl;
    bit<8>      protocol;
```

```
    bit<16>     hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

header tcp_t{
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>   dataOffset;
    bit<4>   res;
    bit<1>   cwr;
    bit<1>   ece;
    bit<1>   urg;
    bit<1>   ack;
    bit<1>   psh;
    bit<1>   rst;
    bit<1>   syn;
    bit<1>   fin;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}

header udp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<16> length_;
    bit<16> checksum;
}

struct metadata {
    /* empty */
}

struct headers {
    ethernet_t     ethernet;
    ipv4_t          ipv4;
    tcp_t           tcp;
    udp_t            udp;
}
```

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol){
            TYPE_TCP: tcp;
            TYPE_UDP: udp;
            default: accept;
        }
    }

    state tcp {
        packet.extract(hdr.tcp);
        transition accept;
    }

    state udp {
        packet.extract(hdr.udp);
        transition accept;
    }
}

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}

control MyIngress(inout headers hdr,
                  inout metadata meta,
```

```
                    inout standard_metadata_t standard_metadata) {

    register<bit<BLOOM_FILTER_BIT_WIDTH>>(BLOOM_FILTER_ENTRIES) bloom_filter_1;
    register<bit<BLOOM_FILTER_BIT_WIDTH>>(BLOOM_FILTER_ENTRIES) bloom_filter_2;
    bit<32> reg_pos_one; bit<32> reg_pos_two;
    bit<1> reg_val_one; bit<1> reg_val_two;
    bit<1> direction;

    register<bit<32>>(4096) syn_register;
    register<bit<32>>(4096) ack_register;
    counter(4096, CounterType.packets_and_bytes) synCounter;
    counter(4096, CounterType.packets_and_bytes) ackCounter;
    bit<32> reg_val_syn;
    bit<32> reg_val_ack;

    register<bit<32>>(4096) udp_register;
    register<bit<48>>(4096) udp_first_time_register;
    bit<32> reg_val_udp;
    bit<48> udp_rate;
    bit<48> time_first_udp;
    bit<48> udp_duration;
    bit<48> udp_cnt;

    action drop() {
        mark_to_drop(standard_metadata);
    }

    action compute_hashes(ip4Addr_t ipAddr1, ip4Addr_t ipAddr2){
        hash(reg_pos_one, HashAlgorithm.crc16, (bit<32>)0, {ipAddr1,
                                                            ipAddr2,
                                                            hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);

        hash(reg_pos_two, HashAlgorithm.crc32, (bit<32>)0, {ipAddr1,
                                                            ipAddr2,
                                                            hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
```

```
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    action set_direction(bit<1> dir) {
        direction = dir;
    }

    table check_ports {
        key = {
            standard_metadata.ingress_port: exact;
            standard_metadata.egress_spec: exact;
        }
        actions = {
            set_direction;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }

    apply {
        if (hdr.ipv4.isValid()){
            ipv4_lpm.apply();
            direction = 0;
            reg_val_syn = 0;
            reg_val_ack = 0;
            reg_val_udp = 0;
            if (check_ports.apply().hit) {
```

```
if (direction == 0) {
    compute_hashes(hdr.ipv4.srcAddr, hdr.ipv4.dstAddr);
}
else {
    compute_hashes(hdr.ipv4.dstAddr, hdr.ipv4.srcAddr);
}
if (direction == 1){
    if(hdr.udp.isValid()){
        bloom_filter_1.read(reg_val_one, reg_pos_one);
        bloom_filter_2.read(reg_val_two, reg_pos_two);
        if(reg_val_one == 1 && reg_val_two == 1){
            udp_register.read(reg_val_udp,reg_pos_one);
            reg_val_udp = reg_val_udp + 1;
            udp_register.write(reg_pos_one,reg_val_udp);
        }
        else{
            bloom_filter_1.write(reg_pos_one, 1);
            bloom_filter_2.write(reg_pos_two,1);
            udp_first_time_register.write(reg_pos_one,standard_metadata.ingress_
            global_timestamp);
        }
        udp_first_time_register.read(time_first_udp,reg_pos_one);
        udp_duration=standard_metadata.ingress_global_timestamp-
        time_first_udp;
        udp_cnt = (bit<48>)reg_val_udp;
        udp_rate = 1;
        if(udp_rate * udp_duration < udp_cnt * 100 * 1000){
            drop();
        }
    }
    if (hdr.tcp.isValid()){
        if (hdr.tcp.syn == 1){
            bloom_filter_1.read(reg_val_one, reg_pos_one);
            bloom_filter_2.read(reg_val_two, reg_pos_two);
            if(reg_val_one == 1 && reg_val_two == 1){
                syn_register.read(reg_val_syn,reg_pos_one);
                reg_val_syn = reg_val_syn + 1;
                syn_register.write(reg_pos_one,reg_val_syn);
                synCounter.count(1);
            }
            else{
                bloom_filter_1.write(reg_pos_one, 1);
                bloom_filter_2.write(reg_pos_two, 1);
```

```
                    }
                }

                if(hdr.tcp.ack == 1 && hdr.tcp.rst != 1){
                    bloom_filter_1.read(reg_val_one, reg_pos_one);
                    bloom_filter_2.read(reg_val_two, reg_pos_two);
                    if(reg_val_one == 1 && reg_val_two == 1){
                        ack_register.read(reg_val_ack,reg_pos_one);
                        reg_val_ack = reg_val_ack + 1;
                        ack_register.write(reg_pos_one,reg_val_ack);
                        ackCounter.count(1);
                    }
                }

                if(reg_val_syn - reg_val_ack >= 50){
                    drop();
                }
            }


        }
    }




    }
}

control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
        update_checksum(
            hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
```

```
                hdr.ipv4.totalLen,
                hdr.ipv4.identification,
                hdr.ipv4.flags,
                hdr.ipv4.fragOffset,
                hdr.ipv4.ttl,
                hdr.ipv4.protocol,
                hdr.ipv4.srcAddr,
                hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
    }
}

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

B.2 syn-flood defense

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

/* CONSTANTS */

const bit<16> TYPE_IPV4 = 0x800;
const bit<8>    TYPE_TCP    = 6;

#define BLOOM_FILTER_ENTRIES 4096
```

```
#define BLOOM_FILTER_BIT_WIDTH 1

typedef bit<9>    egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;


header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>     etherType;
}

header ipv4_t {
    bit<4>      version;
    bit<4>      ihl;
    bit<8>      diffserv;
    bit<16>     totalLen;
    bit<16>     identification;
    bit<3>      flags;
    bit<13>     fragOffset;
    bit<8>      ttl;
    bit<8>      protocol;
    bit<16>     hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

header tcp_t{
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>   dataOffset;
    bit<4>   res;
    bit<1>   cwr;
    bit<1>   ece;
    bit<1>   urg;
    bit<1>   ack;
    bit<1>   psh;
    bit<1>   rst;
    bit<1>   syn;
    bit<1>   fin;
```

```
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}

struct metadata {
    /* empty */
}

struct headers {
    ethernet_t     ethernet;
    ipv4_t         ipv4;
    tcp_t          tcp;
}

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol){
            TYPE_TCP: tcp;
            default: accept;
        }
    }

    state tcp {
        packet.extract(hdr.tcp);
        transition accept;
```

```
    }
}

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}


control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

    register<bit<BLOOM_FILTER_BIT_WIDTH>>(BLOOM_FILTER_ENTRIES) bloom_filter_1;
    register<bit<BLOOM_FILTER_BIT_WIDTH>>(BLOOM_FILTER_ENTRIES) bloom_filter_2;
    bit<32> reg_pos_one; bit<32> reg_pos_two;
    bit<1> reg_val_one; bit<1> reg_val_two;
    bit<1> direction;

    register<bit<32>>(4096) syn_register;
    register<bit<32>>(4096) ack_register;
    register<bit<48>>(4096) first_time_register;
    counter(4096, CounterType.packets_and_bytes) synCounter;
    counter(4096, CounterType.packets_and_bytes) ackCounter;

    bit<32> reg_val_syn;
    bit<32> reg_val_ack;
    bit<48> time_first_syn;
    bit<48> syn_rate;
    bit<48> syn_duration;
    bit<48> syn_cnt;

    action drop() {
        mark_to_drop(standard_metadata);
    }

    action compute_hashes(ip4Addr_t ipAddr1, ip4Addr_t ipAddr2){
        hash(reg_pos_one, HashAlgorithm.crc16, (bit<32>)0, {ipAddr1,
                                                            ipAddr2,
                                                            hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);

        hash(reg_pos_two, HashAlgorithm.crc32, (bit<32>)0, {ipAddr1,
```

```
                                                    ipAddr2,
                                                    hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    action set_direction(bit<1> dir) {
        direction = dir;
    }

    table check_ports {
        key = {
            standard_metadata.ingress_port: exact;
            standard_metadata.egress_spec: exact;
        }
        actions = {
            set_direction;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
```

```
    apply {
        if (hdr.ipv4.isValid()){
            ipv4_lpm.apply();
            if (hdr.tcp.isValid()){
                direction = 0;
                reg_val_syn = 0;
                reg_val_ack = 0;
                if (check_ports.apply().hit) {
                    if (direction == 0) {
                        compute_hashes(hdr.ipv4.srcAddr, hdr.ipv4.dstAddr);
                    }
                    else {
                        compute_hashes(hdr.ipv4.dstAddr, hdr.ipv4.srcAddr);
                    }
                    if (direction == 1){
                        if (hdr.tcp.syn == 1){
                            bloom_filter_1.read(reg_val_one, reg_pos_one);
                            bloom_filter_2.read(reg_val_two, reg_pos_two);
                            if(reg_val_one == 1 && reg_val_two == 1){
                                syn_register.read(reg_val_syn,reg_pos_one);
                                reg_val_syn = reg_val_syn + 1;
                                syn_register.write(reg_pos_one,reg_val_syn);
                                synCounter.count(1);
                            }
                            else{
                                bloom_filter_1.write(reg_pos_one, 1);
                                bloom_filter_2.write(reg_pos_two, 1);
                                first_time_register.write(reg_pos_one,
standard_metadata.ingress_global_timestamp);
                            }
                        }

                        if(hdr.tcp.ack == 1 && hdr.tcp.rst != 1){
                            bloom_filter_1.read(reg_val_one, reg_pos_one);
                            bloom_filter_2.read(reg_val_two, reg_pos_two);
                            if(reg_val_one == 1 && reg_val_two == 1){
                                ack_register.read(reg_val_ack,reg_pos_one);
                                reg_val_ack = reg_val_ack + 1;
                                ack_register.write(reg_pos_one,reg_val_ack);
                                ackCounter.count(1);
                            }
                        }
                        first_time_register.read(time_first_syn, reg_pos_one);
```

```
                        syn_duration = standard_metadata.ingress_global_timestamp -
time_first_syn;
                        syn_cnt = (bit<48>)reg_val_syn - (bit<48>)reg_val_ack;
                        syn_rate = 1;
                        if (syn_rate * syn_duration < syn_cnt * 100 * 1000){
                            drop();
                        }
                    }
                }
            }
        }
    }
}

control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
        update_checksum(
            hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.diffserv,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
```

```
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
    }
}

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```