TENNISON: A Distributed SDN Framework for Scalable Network Security

Lyndon Fawcett*, *Member, IEEE,* Sandra Scott-Hayward†, *Member, IEEE,* Matthew Broadbent*, *Member, IEEE,* Andrew Wright† and Nicholas Race*, *Member, IEEE*
*Lancaster University and †Queen's University Belfast
l.fawcett1@lancaster.ac.uk

***Abstract*—Despite the relative maturity of the Internet, the computer networks of today are still susceptible to attack. The necessary distributed nature of networks for wide area connectivity has traditionally led to high cost and complexity in designing and implementing secure networks. With the introduction of Software Defined Networks (SDN) and Network Functions Virtualisation (NFV), there are opportunities for efficient network threat detection and protection. SDN's global view provides a means of monitoring and defence across the entire network. However, current SDN-based security systems are limited by a centralised framework that introduces significant control plane overhead, leading to the saturation of vital control links. In this paper, we introduce TENNISON, a novel distributed SDN security framework that combines the efficiency of SDN control and monitoring with the resilience and scalability of a distributed system. TENNISON offers effective and proportionate monitoring and remediation, compatibility with widely-available networking hardware, support for legacy networks, and a modular and extensible distributed design. We demonstrate the effectiveness and capabilities of the TENNISON framework through the use of four attack scenarios. These highlight multiple levels of monitoring, rapid detection and remediation, and provide a unique insight into the impact of multiple controllers on network attack detection at scale.**

***Index Terms*—SDN, Monitoring, Network Security, Distributed Control, Scalable Security.**

I. INTRODUCTION

IN an increasingly networked world we rely on having reliable communications technologies for everyday business and social interaction. At the same time, the frequency of network disruptions caused by cyber-attacks is increasing. Distributed Denial of Service (DDoS) attacks, for example, increased by 71% between 2015 and 2016, with a 138% increase in attacks greater than 100 Gbps in the same period [1]. In October 2016, multiple DDoS attacks targeted systems operated by the Domain Name Server (DNS) provider, Dyn. The attack is believed to have been orchestrated by a botnet of IoT devices infected with the Mirai malware with more than 60 services affected [2]. These examples highlight several features of today's network attacks; they are distributed, can involve high traffic volume, and execute remotely through network intrusion. These features indicate the importance of monitoring network events, including network traffic, flow, and device status, to enable effective attack detection and protection.

Network threat detection and protection in traditional networks has typically been achieved with dedicated hardware security appliances (middleboxes), which are costly and require careful placement in the network to ensure appropriate traffic capture. This fixed placement constrains the capability of the network security functions by limiting visibility to the traffic on a particular network segment and thereby limiting the potential for contextual analysis of the captured data.

Software Defined Networking (SDN) has emerged as a concept for the dynamic control of configuration of computer networks. Fundamentally, SDN separates the control and data planes within a network device. This control is then ceded to a software-based controller, which defines the behaviour and operation of the network. The characteristics of SDN include the global network view and programmability of the data plane. In addition, the OpenFlow (OF) Switch specification [3], which describes a protocol for communication between the SDN data and control planes, defines counters for each flow entry/rule in the switch flow table. The flow rule definition also supports a large number of packet header fields (> 40 fields). The granularity of these network statistics captured at the data plane supports traffic monitoring for security functions.

These characteristics enable a powerful feedback loop as follows: network attacks can be detected by capturing traffic flow information and analysing the flow statistics with respect to known signatures/patterns (or through the application of machine learning techniques). Having detected an attack by a volume, type or pattern of traffic, the intrusion protection system can be deployed. The benefit of SDN is that it can be used to program the flow rules to block or filter traffic, or apply another remediation mechanism. However, a performance/accuracy trade-off arises when deploying a traffic monitoring service at scale. The volume of information to be collected can lead to overall performance degradation, whilst introducing a longer collection interval can lead to inaccuracy or delayed remediation. The impact of the volume of monitoring data is particularly significant in a centralized SDN security system with the potential to overwhelm the controller processing functions. It is possible for the monitoring of a DoS attack to generate sufficient monitoring traffic towards the controller that the controller itself becomes subject to DoS [4]. A solution is therefore required to offer a flexible and proportionate monitoring capability with distributed functionality to disperse the control and monitoring load for scalability and resilience.

To address this challenge, we present TENNISON, a distributed SDN security framework built on a multi-level remediation mechanism. A number of OF security frameworks

and applications have been proposed previously, each of which builds on a selection of these SDN characteristics. However, TENNISON's novelty is that it supports a multi-level capability for monitoring. It offers the ability to perform light-weight monitoring across a large number of flows, whilst offering the capability to perform Deep Packet Inspection (DPI) on a selection of flows.

The remainder of the paper is structured as follows. Section II reviews related work followed by a summary of the contributions of TENNISON in Section III. The system architecture is detailed in Section IV. In Section V, we conduct an evaluation of Distributed SDN Controller performance, analyse the performance of TENNISON in the context of four attack scenarios, and assess the scalability of TENNISON components. Finally, Section VI concludes the paper.

II. RELATED WORK

TENNISON is a distributed SDN security framework that supports multiple levels of monitoring. In this Section, we review prior work relating to SDN-based monitoring, SDN-based security systems and distributed SDN controller performance. There are also a number of IETF documents that are relevant to this work. For example, Interface to Network Security Functions (I2NSF) [5], DDoS Open Threat Signaling (DOTS) [6], and Network Security Header (NSH) [7]. We will address these specifically with respect to the TENNISON architecture in Section IV.

A. SDN Monitoring

The combination of the global network view and the granularity of the network statistics captured at the data plane has generated significant interest in network monitoring advances with SDN. Combinations of traditional monitoring protocols such as NetFlow/IPFIX and sFlow with the SDN protocol, OpenFlow, have been explored [8], [9].

Prior work has aimed to tackle the challenges of monitoring at scale. For example, FlowSense [10] uses a push-based approach to receive flow statistics from switches. Adaptive rate monitoring has also been introduced; OpenNetMon [11] and OpenTM [12] poll selected switches at an adaptive rate to reduce network and switch CPU overhead. PayLess [13] uses an adaptive sampling algorithm to vary polling frequency based on measured throughput. Similarly, FlowCover [14] reduces the monitoring communication cost by optimizing the polling function. OpenMeasure [15] uses online learning to adapt flow measurement. This enables scalable measurement with monitoring of the most informative flows and optimal placement of monitoring rules across multiple switches. Proxy-based Monitoring [16] introduces a monitoring table in the proxy to specify the measurement interval for traffic monitoring and associated flow rules are pushed to the OpenFlow switches. Flow-stats-requests/replies are then only exchanged for those specified monitored flows rather than all flows. This reduces the volume of statistics communication in a similar fashion to OpenTAM [17], which is an ONOS-specific adaptive monitoring tool. However, there are several identified limitations to the work; packet capture performance is limited to 60Mbps, the system is limited to 600 condition entries (i.e. rules for capture/monitoring) and it is based on OpenFlow 1.0.

In FlowRadar [18], the authors address the challenge of monitoring in data centers where the existing NetFlow implementation options are unsuitable either due to the prohibitive cost of high-end routers (hardware-based) or excessive switch CPU resource requirements (software-based). The FlowRadar solution is to balance the workload by encoding per-flow counters with low memory requirement and constant insertion time at switches. The decoding and analysis of the flow counters is then performed at the remote collector where there is available computation resource. FlowRadar provides a scalable solution for network-wide monitoring across the data centre independent of SDN or OpenFlow.

SDNMon [19] seeks to improve on monitoring application granularity with an SDN monitoring framework that separates the monitoring logic from the forwarding logic. SDNMon achieves monitoring in a similar way to TENNISON by using multiple tables to separate monitoring and forwarding. However, these tables are not OpenFlow tables, but instead are situated within an application that sits on a customised version of the Lagopus software switch. As such, SDNMon only works with this switch. UMON [20] also addresses the separation of forwarding and monitoring logic. This is achieved by introducing an additional monitoring table at the end of the forwarding pipeline. New monitoring actions are also introduced to support statistics collection on non-routing fields e.g. SYN, ACK etc. This enables, for example, port scan detection based on fine-grained monitoring. However, the implementation is specific to OpenvSwitch.

Most recently, Tsai et al. [21] present an overview of SDN monitoring solutions identifying the challenges and open issues. The research developments are classified according to the monitoring phase i.e. collection, preprocessing, transmission, analysis, and presentation, with the majority of research focused in the preprocessing phase e.g. OpenTM [12]. The benefit of leveraging sFlow is highlighted to integrate monitoring in hybrid environments with legacy network devices, which is the approach taken with TENNISON. Similarly, we tackle the open issues identified in [21]; leveraging monitoring and data collection to detect security threats, and multi-domain collaborative network monitoring.

TENNISON extends existing monitoring solutions to offer the unique ability to perform light-weight monitoring across a large number of flows whilst offering the capability to perform DPI on select flows. The TENNISON multi-level monitoring design selects the appropriate monitoring tool for the required detection i.e. sFlow when sampled monitoring is sufficient, IPFIX for detailed monitoring, and mirroring/redirection to DPI for payload inspection. This reduces the volume of monitoring traffic, supporting deployment in a large-scale network. The design of TENNISON further supports scalable monitoring through the distribution (and coordination) of monitoring across multiple controller instances.

B. SDN Security Systems

The first protection architecture for SDN was proposed in [22], prior to the development of the OpenFlow protocol. Since

then, prior work has focused on taking advantage of SDN characteristics for intrusion detection/prevention services.

FRESCO [23] proposed a framework for the design and development of security specific modules as OpenFlow applications. TENNISON also provides a security framework. However, TENNISON combines SDN and NFV to leverage both existing security functions (e.g. Snort, Bro etc.) and to provide the appropriate network telemetry for advanced security applications.

CIPA [24] applies an artificial neural net (ANN) across OF-SDN switches for the detection of distributed, coordinated intrusion attacks such as scanning, worm outbreak and DDoS. The False Positive/Detection Rate and communication overhead are all shown as improvements over Gamer [25]. With [26] and [27], Ha et al. consider intrusion detection in SDNs. In [26], a flow grouping scheme is proposed to determine which flows to forward to which IDSs to achieve the best intrusion detection performance. Principal Component Analysis (PCA) is used for grouping suspicious flows and gravity-based clustering is used to assign these groups to IDSs. In the example results, each of the TAPs feed into an aggregation switch from which the assignment to IDSs is made. From a latency perspective, this would counteract the benefit of the distributed implementation.

In [27], the authors present results for optimising the sampling rate for each switch to improve inspection performance of malicious traffic in large networks. This is designed to fully utilise the inspection capability of the malicious traffic while keeping the total volume of sampled traffic below the maximum processing capacity of the IDS. The optimisation is a function of the data rates of the switches, the malicious traffic rate of the flows and the sampling rate at the switches. However, the malicious traffic rate must be estimated to begin with and can then converge to the actual value based on the adjusted sampling rates. The selection of this rate would strongly influence the convergence time. Simulation results showed that the algorithm converged in about 100 s for the smaller network, which is somewhat impractical.

SDN4S [28] is proposed as a system and solution to minimise the time between incident detection and resolution by using automated countermeasures based on SDN. The system creates incident-specific response work-flows that automatically implement actions and network countermeasures. The work is motivated by the issue with the increasing volume of network threats and hence security alerts and the limited resources to analyse and respond to the alerts. The solution is based on the concept of playbooks, which match/tailor the security incident response to the business and environment. This work has some similarities with TENNISON. For example, there is a similar component architecture, the ability to receive alerts from external security systems, and the OpenFlow-based network protection mechanism. However, although the motivation of SDN4S is to minimize response time, there is no evaluation of the response time or of the effectiveness of the detection/protection mechanisms.

PSI [29] is proposed as a new enterprise network security architecture to address the issue of existing enterprise security approaches lacking precise defences in *isolation, context,* and *agility*. The authors describe these as follows: for *isolation*, the defence system must ensure that security policies do not interfere with each other; for *context*, the defence system must be able to enforce customised policies for individual network devices, and for *agility*, the defence system must be able to change policy at fine-grained time-scales. PSI and TENNISON address a similar set of problems related to usable network security; that of appropriate, efficient network security. Both systems achieve this by leveraging SDN and NFV. PSI emphasizes the use of NFV with the tunnelling of all network traffic through a cluster of virtualised appliances within which the relevant services are applied to the traffic. In contrast, TENNISON emphasizes the use of SDN. Rather than tunnelling all traffic through a cluster (albeit virtual and hence flexibly deployed), TENNISON leverages the SDN switch design to effectively apply security policy in the data plane through the selection of traffic for monitoring at different granularities. This results in flows being conservatively mirrored (rather than redirected), reducing overall network load and latency for benign traffic. Additionally, unlike PSI, the use of IPFIX and sFlow provides TENNISON with visibility in legacy networks.

Finally, Athena [30] is an SDN anomaly detection framework. Like TENNISON, Athena addresses the issue of scalability across large, distributed SDN deployments. The framework supports the development of machine-learning based security applications with two scenarios illustrated; DDoS detection and Link Flooding Attack mitigation. However, Athena does not support interoperability while TENNISON can integrate with a hybrid SDN-legacy network. Furthermore, Athena does not offer adaptive measurement for resource optimisation.

C. Distributed SDN Controller Performance

As highlighted in Section I, the ability to scale up controller processing in response to network traffic variations is critical to the network security system. Both hierarchical and distributed control mechanisms have been proposed, as surveyed in [4]. A few of these works specifically consider load-balancing. In Kandoo [33], local decision-making is separated from network-wide decision-making. Certain applications can be supported by event processing at local controllers reducing the load on the root controller. ElastiCon [34] proposes an elastic distributed controller architecture to dynamically adjust the controller pool in response to changing traffic conditions.

Hydra [35] presents a solution to support latency-sensitive applications by partitioning the control function based on functional rather than topological slicing (as in [33], [34]). Functional slicing splits control plane functions (and, therefore, applications) across servers. The performance results presented in [35] show an improvement in controller throughput and response time under increasing load for latency-sensitive applications. However, the dependency on communication-awareness means that the placement algorithm would be required to run very frequently in larger networks.

The SDN controller, ONOS [36], publishes performance results for each new controller version [37]. These include distributed performance results. However, the results analyse particular components of the controller in isolation. In this

TABLE I: Scalability comparison of SDN security systems

System Name	Controller	Multi-level monitoring	OF response methodology	Attack detection (Extensibility)	Distribution	SIEM-Like Human Interface
TENNISON [31]	ONOS	Yes	Hybrid	4+ (Yes)	Yes (Control & System)	Yes
FRESCO [23]	NOX	No	Reactive	3+ (Yes)	No	No
CIPA [24]	POX	No	Reactive	3+ (Yes)	No	No
SDN4S [28]	HPE VAN	No	Reactive	1+ (Yes)	No	Yes
PSI [29]	ODL	No	Proactive	1+ (via NFV)	Yes (System)	No
Athena [30]	ONOS	No	Reactive	2+ (Yes)	Yes (Control & System)	No
OFX [32]	Ryu	No	Hybrid	2+ (Yes)	Yes (Switch)	No

paper, we provide distributed control performance results for a full network implementation scenario considering varying traffic, cluster size, and network scale (volume of network devices). By simultaneously considering these multiple factors, real conclusions can be drawn regarding the performance of a distributed ONOS cluster in a production network.

III. TENNISON CONTRIBUTIONS

Section II-B highlighted security frameworks and applications that build on the SDN characteristics of programmability and logically centralised control. The performance impact introduced by the control communication channel (and controller processing capacity) is a challenge for low-latency attack detection. Similarly, many of the proposed security frameworks/applications rely on modifications to OpenFlow, SDN devices, or interference with fundamental network forwarding behaviours, which restrict real deployment of the solutions.

In Table I, we highlight the features of a scalable, distributed monitoring and security system, and compare existing solutions with the TENNISON framework.

The TENNISON framework is motivated by the desire to present an adaptive and extensible security platform that is technology-independent and capable of supporting a wide range of security functions. TENNISON does not remove the requirement for controller interaction, as demonstrated in Avant-Guard and OFX [32], [38]. However, the level of controller interaction is rendered flexible and proportionate to the threat detection requirements. This is assisted by the use of other monitoring and inspection tools that are deployed to the network, relieving pressure from the SDN control channel.

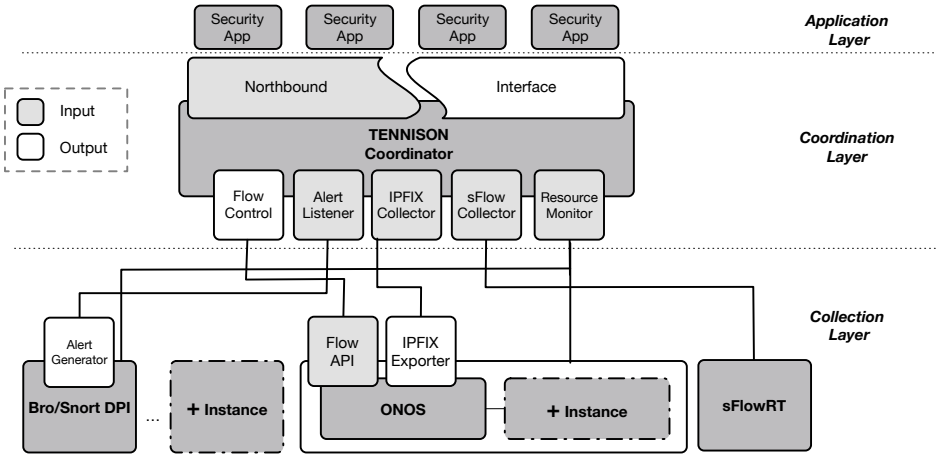
TENNISON offers a next-generation security framework with the following attributes:

- **Efficiency and Proportionality:** TENNISON provides an efficient monitoring and remediation framework where resource consumption is commensurate to current threat levels. TENNISON supports standard network monitoring protocols such as sFlow and IPFIX, incorporating them in a multi-level monitoring function, as described in Section II. As demonstrated in Section V, a volumetric attack can be efficiently detected using the TENNISON L1 monitoring with sFlow while more fine-grained attacks engage L2 IPFIX monitoring and/or L3 DPI capabilities.
- **Scalability and Visibility:** The network view provided by the distributed ONOS control function and leveraged by the TENNISON coordinator enables placement of the monitoring and remediation rules on the appropriate network devices for optimal security protection. Visibility across the network includes legacy equipment, through compatibility with ubiquitous protocols such as sFlow. Separation of the TENNISON coordinator from the SDN controller helps to scale out the system, and maintain openness in the selection of TENNISON's controller.
- **Programmability and Extensibility:** The TENNISON framework provides a rich API that allows operators to define the behaviour of the network and its resources in response to new and existing threats. To introduce a new security function, the template sFlow/IPFIX/DPI application can be modified and activated via the GUI. To modify a threshold in an existing function e.g. DDoS attack trigger threshold, the new threshold value is simply added via the GUI. This will automatically be mapped to the relevant TENNISON components.
- **Transparency:** TENNISON is transparent to other network functions within the network. This is realised using a custom *security pipeline* built using the OpenFlow multi-table feature to provide the network monitoring and remediation without interfering with the basic forwarding functionality and additional services of the network.
- **Availability and Resiliency:** One of the key requirements of a SDN security platform is the resilience of the control plane to support high availability and provide redundancy in the case of controller failure [39]. For this reason, TENNISON has been designed to integrate with a production-grade distributed controller, ONOS [36], which supports high availability and fault tolerance. Even in the case of a failure of a controller instance, the system can still monitor and protect the network.
- **Interoperability:** Finally, TENNISON works in conjunction with de-facto industry-standard security tools, such as Snort and Bro. The integration of Snort DPI with TENNISON is demonstrated in Section V. Available DPI instances are automatically detected by TENNISON.

IV. TENNISON SYSTEM ARCHITECTURE

In this section, we introduce the TENNISON system architecture highlighting the individual components and features of the design that realise the objectives of the framework described in the previous section.

Fig. 1: TENNISON System Architecture



In Figure 1, we illustrate the overall system architecture. The lower layers (Bro and Snort DPI, ONOS Controllers, and sFlowRT) are the appliances and instances deployed within the network, cumulatively referred to as the *Collection* layer. These instances are fundamental to the operation of TENNISON, in that they provide both control and monitoring functionality to the higher layers.

In some cases, minor extensions have been made to the Collection appliances to enable them to report and communicate with the TENNISON *Coordinator*, which forms the *Coordination* layer of the architecture, and is responsible for storing and aggregating all of the information generated by these appliances. The interfaces between these appliances and the Coordinator represent a flow of information, with the interfaces providing input or output (see legend in Figure 1). This illustrates that both the packet inspection tools and the flow monitoring applications produce input while the ONOS controller alone offers an output. This output enables the coordinator to modify the forwarding plane of the network, via the ONOS controller, and provide the functionality and programmability required for TENNISON to operate effectively.

The *Coordinator* is responsible for coordinating the overall operation of the architecture, and acts as an intermediary between the upper and lower layers; Application and Collection layers, respectively. The *Coordinator* is intentionally built independent of any other component, and is completely technology agnostic. This allows interoperability with alternative technologies and flexibility in terms of design and placement in the network. The system is easily distributable (i.e. multiple TENNISON instances), and, as such, is able to scale to the size of network. The central role of the coordinator grants an authoritative view of the network topology and its current state with the ability to simultaneously modify both the network *and* the monitoring services running within it. The final, and uppermost layer, is the *Application* layer, which hosts security applications. These applications interact with the coordinator leveraging its functionality to realise custom and dynamic security behaviours. This enables the creation of applications tailored to specific threats, or integration with existing tools deployed in a network.

The remainder of this section contains a detailed breakdown of each of the system components.

A. TENNISON *Coordinator*

In this section we detail the subsystems within the Coordinator that together provide the functionality to security applications in the Application layer (see Figure 2).

1) Southbound Interface (SBI) Modules: The first SBI module is the *Flow Control* interface, which is an *output* from the Coordinator. The Coordinator uses this to control the flow of traffic through a network, directing and shaping this towards the various appliances under its control. Importantly, this is an intent driven process; rather than defining a specific switch on which a flow should be modified, the Coordinator describes a high level intent, which is implemented by the network controller based on its knowledge of the topology. Further details of the network controller are provided in Section IV-C.

The remaining four SBI modules all provide *inputs* to the Coordinator. The *Alert Listener* is a REST interface used to collect messages generated from the various packet inspection appliances located in the network. An alert message is generated by a DPI tool when a suspicious flow or packet flows through the appliance. This alert is then passed from the appliance to the coordinator via this interface module. A similar process is followed for the *IPFIX Collector*, *sFlow Collector* and *Resource Monitor* interfaces. In the case of the *IPFIX Collector*, aggregate flow records are received from the network controller. Similarly, the *sFlow Collector* provides sampled flow record statistics when high volume monitoring is used in the network. Finally, the *Resource Monitor* collects resource usage information (e.g. cpu/memory/flow table occupancy) from the switches and controllers in the network. Together, these interfaces provide a holistic view of the entire network, including traffic levels, threat analysis and resource utilisation. We note that in networks managed by an NFV orchestrator, an additional SBI module would be required to share identifiers (e.g. MAC addresses) of new vDPIs added to the network.

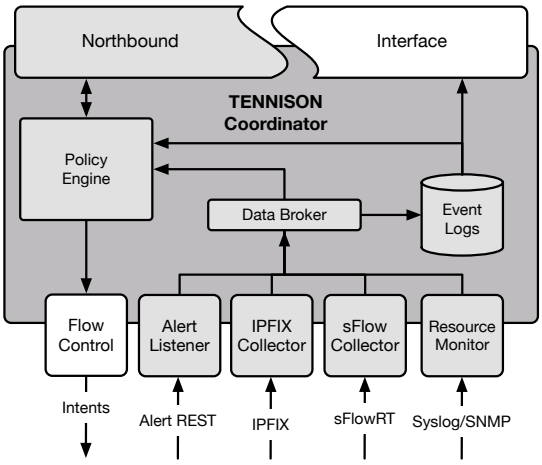


Fig. 2: TENNISON Coordinator Subsystems and SBI

2) Data Broker and Event Logger: Once a message is received from an SBI module, its content is passed to the *Data Broker*. This acts as an intermediary to determine the destination of the message within the system; toward the *Policy Engine* and/or the *Event Logger*. Importantly, the *Data Broker* enables extensibility of the SBI by providing a generic interface to which new collectors can connect. Furthermore, the *Data Broker* queues messages, acting as a virtual buffer between incoming messages and the policy engine.

The *Event Logger* is a long-term storage medium, realised with a key-value store, that keeps each message in its entirety. This database can then be searched by both the system and external and entities- to retrieve historical information. It provides persistent storage for the coordinator, enabling rapid recovery in cases of failure or migration. The event logger can also be configured to automatically expunge stored messages after a fixed period. This allows the storage footprint to remain consistent without the need for explicit maintenance.

3) Policy Engine: The *Policy Engine* contains the logic through which new messages are processed (originating from one of the input interfaces), historical trends are analysed (via Event Logger) and resulting actions are taken (via Flow Control). The policy engine has a permanent storage state to record security policies and monitoring decisions. It is also fully configurable by the security applications, to add, remove or modify logic, as required. Examples of the implementation of this policy engine are provided in Section IV-D.

4) Northbound Interface (NBI): The *Northbound Interface* is key to enabling the programmability and extensibility inherent in TENNISON. As shown in Figure 3, the NBI enables the security applications to both read the current network state and to modify it according to custom internal logic, with the ability for multiple applications to be used in parallel.

A security application connecting to the NBI first registers with the controller by providing a unique ID and a set of credentials. This enables the coordinator to identify the application, and to authorise it with the appropriate permissions. This includes permission to read the network topology, and permission to interact with neighbouring applications,

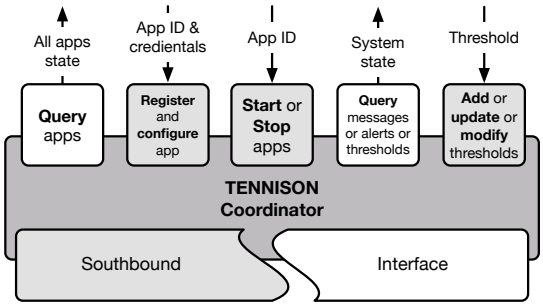


Fig. 3: TENNISON Coordinator Northbound Interface

as required. The benefit of this interaction with the other applications and access to the Coordinator policy information is to remove potential conflicts between applications in terms of network behaviour. From a system security perspective, this capability is closely controlled by the application authorisation in TENNISON to avoid information leakage that could expose the network to malicious attack.

We note the work by the IETF Interface to Network Security Functions (I2NSF) working group [5], which describes a framework and reference model for the integration of network security functions. RFC8329 [40] specifically highlights the importance of authentication, authorisation and auditing (AAA) of application functions and management of overlapping security policy. While TENNISON is currently a single-domain prototype implementation, the framework aligns with the recommendations of I2NSF e.g. AAA, IPFIX support etc. However, some extensions would be required to both the NBI and the policy engine to integrate externally controlled NSFs.

As will be described in Section V, one of the services provided by TENNISON is single-domain DDoS detection and mitigation. IETF DOTS (DDoS Open Threat Signaling) [6] proposes a protocol for a DOTS agent to alert a gateway to the threat of a DDoS attack by advertising black or whitelisted addresses. To provide an alert for neighbouring domains or to enable interoperation with multiple DDoS mitigation vendors, TENNISON could act as the DOTS client exporting an alert via the NBI.

B. TENNISON *Security Pipeline*

A key benefit of TENNISON is the ability to provide network monitoring and remediation without interfering with forwarding functionality and additional services of the network. This transparency is achieved with the TENNISON Security pipeline, as shown in Figure 4. The pipeline manifests itself as an ONOS driver, which prepends TENNISON's security tables to the other network application tables.

There were three primary reasons for designing a new pipeline for TENNISON. Firstly, the switches need to be able to store specific rules based on traffic that needs to be monitored, secondly to be able to support existing forwarding applications with TENNISON's security application, and thirdly, to mirror and tunnel duplicated traffic to DPIs using an overlay network. Where tunneling is required in an SDN environment, it is typically manually configured on the SDN switch using an

existing tunneling protocol, such as Generic Routing Encapsulation (GRE). There are a number of other solutions considered for SDN, including NSH [7] and Geneve [41]. However, these solutions focus on solving challenges such as Service Function Chaining or are designed to be used as a conventional tunnel without mirroring. For the purpose of a system such as TENNISON where dynamic scaling of functionality is required (i.e. allowing network functions to be automatically added/removed within the network), none of these solutions are suitable. As such, TENNISON implements a simple ONOS application to create point-to-point tunnels.

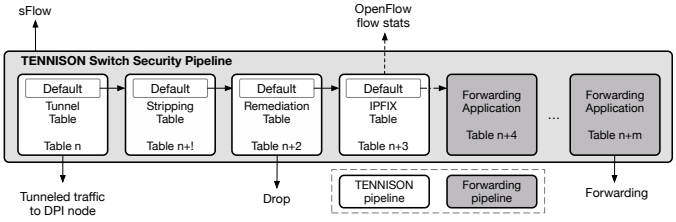


Fig. 4: TENNISON Security Pipeline

A number of designs were considered for the security pipeline during the development of TENNISON, alongside an analysis of the level of conformance of current market SDN switching equipment with OpenFlow 1.3 functionality. In order to provide a practical solution capable of deployment with available SDN devices, and because of limitations identified with current SDN hardware switches relating to the number of tables available, the match fields per table, and the actions available per table, the TENNISON security pipeline assumes only the base non-extended OpenFlow 1.3 requirements.

As shown in Figure 4, tables controlled by TENNISON precede the regular forwarding functionality. This enables transparent monitoring and actions to be implemented without the requirement to modify the forwarding functionality.

TENNISON employs a security pipeline with a total of 4 flow tables:

- The **Tunnel table** is the first table in the overall pipeline and acts as an overlay forwarding application that uses VLANs to classify tunneled traffic. The primary purpose of the table is to forward traffic to the nearest DPI service on the network.
- The **Stripping table** is used to strip tunneling forwarding logic from mirrored flows. Although not strictly required, this table is maintained to support compatibility across all OpenFlow 1.3 compliant hardware and software switches.
- The **Remediation table** contains the remediation intents. After tunneling management, this table appears next in the pipeline in order to optimise network performance by blocking/dropping traffic identified as malicious before it absorbs further processing resources.
- The **IPFIX table** contains the monitoring intents. This table is last in the security pipeline and may pass monitored traffic either to the DPI table for further analysis or directly on to the forwarding pipeline.

C. Network Controller

TENNISON relies on a network controller to operate effectively. Using SDN technology, this controller should be capable of viewing and modifying the underlying physical or virtual network paths, and support traffic steering or manipulation. TENNISON does not require a specific SDN technology. For the purpose of this work, we use ONOS [36].

1) Controller distribution: Designed by ON.Lab, Open Network Operating System (ONOS) was launched in 2014 as a SDN network operating system for service provider networks with a focus on high availability, scalability and performance. ONOS implements distributed control with multiple controller instances forming a cluster, which is a process by which one or more controllers are connected and data about the state of the network is shared between them. This ensures that in the event of one controller failing, the other remaining controllers in the cluster maintain the network. It also supports scale-out of the system; making it possible to manage networks with 100s of networking devices and 1000s of hosts.

The ONOS cluster instances synchronise to provide the global network graph using the Raft [42] consensus algorithm. The *StorageService* interface ensures a consistent state of the databases across all instances of an ONOS cluster. Each network element is assigned a master ONOS instance and the remaining instances will be standby for that network element. If the master instance fails, an election takes place between the remaining instances to elect a new master. It is possible to balance the masters to provide an even distribution of network elements to each member of the cluster.

For TENNISON, distributed ONOS provides a scalable and fault-tolerant substrate. TENNISON also leverages the ONOS default forwarding and routing applications.

2) Security Intents: As described in Section IV-A1, the Coordinator generates topology-agnostic intents, which are implemented by the network controller to alter the network behaviour. Several modifications have been made to ONOS to support integration with TENNISON, creating a more generic intent API. These modifications are present as an ONOS application and are represented as the Flow API in Figure 1. The ONOS application is registered with the highest priority in the ONOS event processing pipeline to support the flow illustrated in Figure 4. The provided intents are as follows:

***Monitoring intent**:* A monitoring intent instructs the controller to monitor a specific flow. Based on this intent, the ONOS controller will insert flow rules into switches that the flow traverses. OpenFlow statistics for this flow are then aggregated and converted into IPFIX data, which is passed to the TENNISON Coordinator via the *IPFIX Collector*.

***Redirection intent**:* A redirection intent instructs the controller to redirect a specific flow (defined by a tuple) towards a specific type of appliance. For example, it may instruct the controller to redirect all TCP traffic traversing port 80 towards the nearest Snort DPI instance. This redirect modifies the complete flow, tunneling it to a new destination, and stops any packets of the flow from traversing the normal forwarding path within the network. Based on the security pipeline logic, the redirection intent is written to the first table of the TENNISON pipeline.

Mirror intent: The mirror intent has similar functionality to that of redirection. However, there is a significant difference in that the original flow remains in the network. As such, the traffic is copied in the first instance, with the duplicate flow tunneled towards a new destination (e.g. DPI appliance), whilst the original packets are forwarded as normal.

Remediation intent: A remediation intent is the final step of a security application used by TENNISON, and enables the coordinator to make a definitive action with regard to an identified and detected threat. This includes completely blocking a flow in the network, or rate limiting a flow to control its behaviour. As previously noted, this intent is written to the third table in the TENNISON pipeline.

D. TENNISON *Multi-level Monitoring*

TENNISON operates a tiered system of monitoring to provide scalable and transparent network security. The multiple levels of monitoring are illustrated in Figure 5, with lightweight monitoring for a high volume of flows at L1 and L2 leading to detailed monitoring for a reduced flow count at L3.

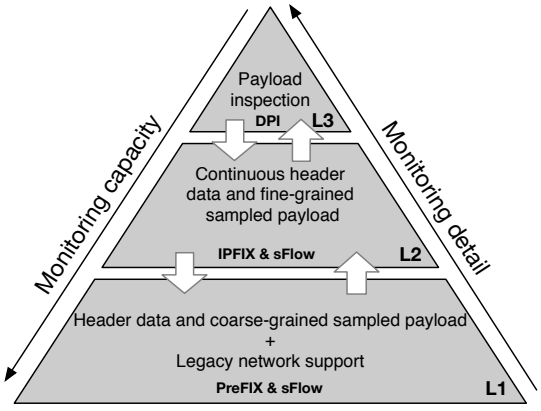


Fig. 5: TENNISON multi-level monitoring

"PreFIX" identified in Figure 5 is part of L1 and, by default, provides the header information for the *first* packet in the flow for *every* flow. At both L1 and L2, sFlow provides latent, always-on sampled monitoring. Furthermore, as sFlow can be configured on non-SDN switches, it provides TENNISON with visibility of legacy networks. sFlow captures flow information based on sampling. The sFlow agent in the network element is configured to export sFlow records to sFlow-RT, which then reports alerts to the TENNISON Coordinator for remediation and to support multi-stage attack detection. The sampling rate, polling rate, and packet header length are configurable and can be dynamically updated based on the network state and immediate monitoring requirements.

In addition, at L2, IPFIX data input to TENNISON based on defined OF monitoring intents provides a more fine-grained and continuous monitoring capability suitable for detection of attacks that can evade a sampling-based monitoring approach.

Finally, L3 represents TENNISON's capability to forward suspicious traffic towards DPIs for classification. This leverages TENNISON tunneling, which provides the means to forward and mirror specific traffic from any host on the network to any destination without modifying the packet. This is a further example of the system scalability. As the network increases in size and DPI processing throughput reduces, additional DPI instances and tunnels can be instantiated on-the-fly ensuring optimal network protection. This also has the advantage that it is invisible to reconnaissance from an attacker; with traffic being mirrored and not redirected, no latency is added to the forwarding plane.

1) Implementing Multi-level Monitoring: There are two key factors in the implementation of multi-level monitoring in TENNISON. The first is the TENNISON policy engine, which enables the network operator to define and specify monitoring and security detection and protection mechanisms in accordance with the network deployment environment. Figure 6 provides a visualisation of the policy engine in the form of a match action table. This table is "hit" when a new *event* enters the coordinator; an event could be generated from a variety of sources including DPI, sFlow-RT, IPFIX, PreFIX, or a custom event from a northbound application. The three columns in Figure 6 represent the following: *Matches*, which consist of packet headers or alert types; *Conditions*, which verify if an event violates a threshold; and *Treatments*, which manipulate or upgrade the level of monitoring for specified traffic. For example, policy #2 can be read as follows: given a DPI alert on a specified MAC src address, that flow will be blocked. Similarly, for policy #4, given an sFlow_RT alert for an identified IP src address, and with the threshold specified by policy #1 exceeded, that flow will be rate limited.

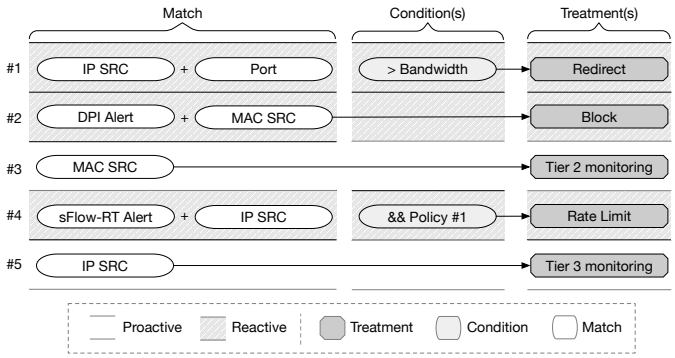


Fig. 6: TENNISON Policy Engine illustration

The second key element is the resource monitor, which provides three specific functions based on the resource usage information; (1) The optimal placement of the monitoring rule (i.e. on which switch(es) along the traffic path to place the rule) is determined with the objective of maximising network protection while maintaining network performance (i.e. avoiding potential switch flow table overflow and controller processing overload). (2) In the case that a switch/controller pair is approaching a resource limit, the management of the switch will be transferred to another controller. This enables more efficient network operation and accelerates detection/protection times in the case of attacks due to reduced latency on the data-control communication path. (3) In a high-traffic volume scenario, the resource usage information is used to dynamically adjust the monitoring level (simultaneously reducing IPFIX monitoring

and increasing sFlow sample-based monitoring to actively manage the network resources). While introducing a marginal increase in attack detection time, this multi-level monitoring design enables continued network operation under high load.

V. TENNISON EVALUATION

This Section provides a comprehensive evaluation of TENNISON. We outline our test environment, evaluate the scalability properties of a distributed SDN Controller, analyse the performance of TENNISON in the context of four attack scenarios, and assess the scalability of TENNISON components.

A. Evaluation Environment

To evaluate TENNISON, we use a topology with 350 nodes connected to 19 partially connected switches, representative of a large size business network [43] with access, distribution and core networking layers. This topology was realised using Mininet [44]. Mininet is a network experimentation tool that enables evaluation, validation and measurement of SDN applications. In order to create the network topology, Mininet instantiates Linux namespaces, which logically creates multiple hosts and switches (such as Open vSwitch (OvS)). The testbed runs on a general-purpose server with 256GB RAM, and two Intel Xeon E5-2697v4s totalling 32 cores. The server runs Ubuntu 16.04.3 and resources were shared between Mininet (Hosts, OvS), the SDN Controllers (ONOS v.1.11) and TENNISON. In order to test performance at scale, we use the controller benchmarking tool cbench [45], which can emulate the control plane of thousands of SDN switches.

B. Distributed SDN Controller Performance

Prior to testing the performance of TENNISON, two experiments were carried out using cbench to understand the scalability properties and potential overheads associated with running a distributed SDN controller; ONOS.

1. Controller maximum packet-in processing throughput. This was achieved by sending packet-ins originating from 16 emulated switches, and counting the number of response packet-outs from the controller. This experiment was repeated 70 times. The responses per second for an increasing number of controller instances are shown in Figure 7. From this, it can be determined that the performance of ONOS increases logarithmically to the cluster size.

2. Impact of increasing number of switches on ONOS cluster size. For this test, 1000 packet-ins per second are generated per switch, and the delay in their response is measured and averaged. The results in Figure 8 illustrate the latency with both an increasing number of switches and an increasing number of controller instances. These results highlight the trade-off between cluster size and volume of switches controlled by the cluster. For a network of up to 32 switches, a single controller instance provides the optimal response time. The reduction in latency is only achieved with additional controller instances for larger network sizes.

The results presented in Figures 7 and 8 provide a benchmark of the performance of the distributed controller and the relationship between network size and distributed control. We further explore the impact of multiple controller instances on detection and remediation time in an enterprise environment in Section V-E.

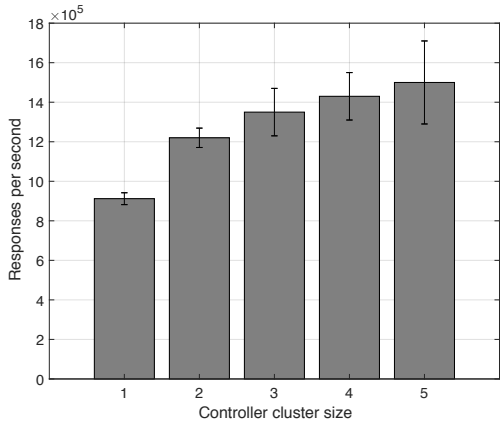


Fig. 7: ONOS Controller Performance (Responses/s)

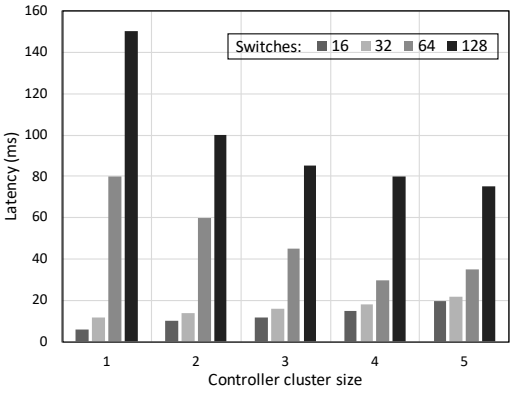


Fig. 8: ONOS Controller Performance (Latency)

C. Attack Detection and Remediation

TENNISON is designed to support the detection and remediation of a variety of attack types. In order to demonstrate TENNISON's multi-level monitoring capability, four attack scenarios are used to exercise different components of the system. TENNISON's detection capability goes beyond these four attack scenarios (summarised in Table II), but they serve to illustrate the system operation.

The performance of TENNISON against these attack scenarios is measured by the length of time it takes to detect and subsequently protect against each attack. Figure 9 shows the breakdown of the attack remediation with each attack (DoS, DDoS, scanning and intrusion) stressing different parts of the system. In the figures, "Monitor rule" refers to the time to install the appropriate monitoring, "App detection" refers to the TENNISON northbound application attack detection time, "Mirror rule" refers to the time to install the mirroring rule, "Alert" refers to the time for Snort/sFlowRT to detect the attack and generate an alert, and "Block rule" refers to the time to install the block rule in the relevant network elements.

TABLE II: Summary of Attack Detection/Protection Mechanisms

Attack Type	Attack Identification	Fields of Interest	Detection Method	Protection Method
DoS	Volume of traffic flows targeting a single host exceeds a defined threshold	IP Source and Destination	Level 1 (sFlow)	Block/Drop
DDoS	A volume of traffic flows from multiple source IPs targeting a single host exceeds a defined threshold	IP Source and Destination	Level 2 (IPFIX) & Level 3 (DPI)	Block/Drop
Scanning	Increase in Attacker, Host A, ratio to target addresses	IP Source and Destination & Port	Level 2 (IPFIX)	Block/Drop
Intrusion	Attacker tries to log in to an FTP server with a username containing the predefined control sequence	Port & Username Field	Level 3 (DPI)	Block attacker and FTP Server

Note that the time to detect an attack can include the attack execution time e.g. x packets within y seconds includes $\leq y$ in the measurement. This will be discussed per scenario, as appropriate. During each experiment, the nodes performing attacks as well as the emulation environment are under stress, which can produce slightly varied results on each iteration. This variation is exacerbated on attacks that are more resource intensive. For this reason, the test is reproduced 10 times for each attack type (40 experiments in total) and the results averaged. The range of variability in remediation time is illustrated by the error bars in Figure 9.

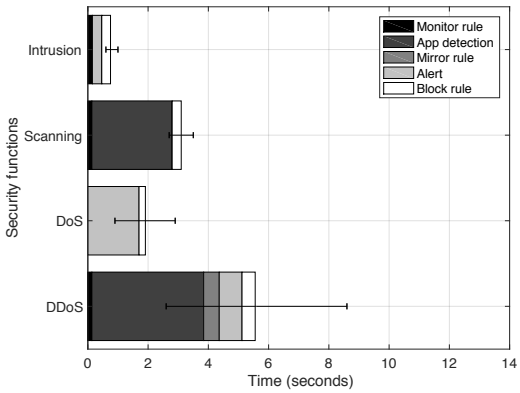


Fig. 9: Attack Remediation Latency - Single Controller

1) DDoS: For this attack, a single host is flooded with TCP SYN requests from multiple source IP addresses. The attack is executed using Hping3 with the following configuration: small packet size, SYN flag, random source, and fast sending rate. The method of detection for a DDoS attack is reactive as the number of host connections has to be tracked and the traffic then has to be forwarded to a DPI for confirmation. The attack is first detected when a volume of unique flows targeting a single host exceeds a configurable threshold within TENNISON's IPFIX DDoS application. For this experiment, the threshold was set to 70 connections from multiple sources to a single destination within 10 seconds. Note that it takes the attacker at least 2.5 s to send sufficient packets to exceed the set threshold defined in both Snort and TENNISON. The traffic is mirrored to Snort where the attack is confirmed via the Snort rule. An alert is sent to the coordinator, which in turn instructs ONOS to block traffic for that destination node.

As shown in Figure 9, the DDoS attack detection time is the longest of all those measured. This is a consequence of the application logic and the time to complete the attack. However,

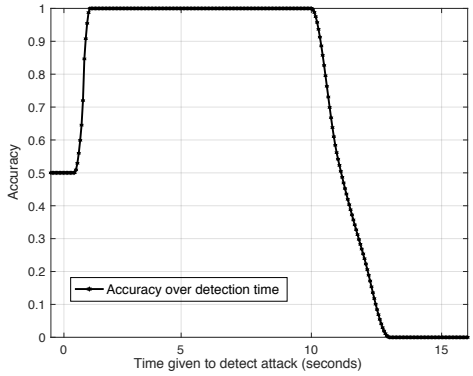


Fig. 10: Accuracy of DDoS Detection and Remediation

this combined detection approach ensures a high detection accuracy (low false positive rate), as illustrated in Figure 10.

The accuracy of the IPFIX DDoS application was measured with respect to the configured Snort DDoS SYN Flood detection threshold. It takes a number of parameters including packet count, which affect the time to detect an attack and the False Positive Rate (FPR). The accuracy measurement is based on Equation 1.

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN} \tag{1}$$

The results in Figure 10 show that for a detection time between 0 and 1.7 s, the accuracy is low due to a high FPR. However, increasing the Snort detection time, increases the accuracy. Of course, this also increases the overall attack remediation time. Once the detection time is increased beyond 10 s, the accuracy drops sharply. This is due to a high True Negative Rate (TNR), which occurs because the threshold is now too high to detect the DDoS SYN Flood attack. Due to this trade-off, we conclude that a Snort threshold detection time of 2 s will provide an optimum remediation time and high accuracy. Analysis of the error bars in Figure 9 highlights a significant variance of the DDoS attack when compared to the other attacks. This fluctuation in results between experiment iterations is due to the host-intensive nature of the attack, which causes the attacker to fluctuate the number of packets per second being sent, thus changing the time until thresholds are met within the system. TENNISON's detection accuracy is comparable to results shown in Athena, which under similar conditions show a +0.99 accuracy [30].

2) Scanning: For this attack, with Nmap, a single host scans 200 ports on another host on the network. The method of detection for a port scan is reactive as the number of ports has to be tracked. The TENNISON northbound port scan application tracks the number of ports accessed across all hosts on the network. Once the number of ports between two hosts exceeds the configured threshold within the defined period, the source of the attack is blocked. Similar to the TENNISON IPFIX DDoS application, the majority of detection time is absorbed by the gradually increasing suspicion within the application logic to determine whether or not the traffic is malicious. The results for the port scan are also in Figure 9.

3) Intrusion: For this exploit a backdoored version (2.3.4) of VSFTPD was used. The detection method for this exploit is proactive as the required thresholds to detect the attack are pre-installed within TENNISON; a threshold to mirror FTP traffic and a threshold to define the response to the Snort alert. The FTP server is exploited by connecting over a network and using ':)' as the username upon login. This opens a network interface on port 6500 which provides direct shell access to the servers' host. Then as the attacker logs in, traffic is redirected to Snort which analyses the payload, scanning for ':)' as the username. Once detected, an alert is sent to TENNISON, which in turn blocks the attacker.

Figure 9 shows that this attack is the quickest to detect. This is due to the proactive nature of the detection method with pre-installed thresholds. Furthermore, only one packet is required to detect this attack (the login packet) whereas the other attacks are detected over time following observation of multiple packets. The results from this attack are indicative of the general performance of the TENNISON framework as the attack exercises the complete security pipeline (i.e. monitoring, redirecting, and blocking) but without the variance included by the DDoS/Port Scan applications, which are dependent on the specific threshold configuration.

4) High-volume DoS: For this attack, a single host is flooded with TCP-SYN requests. The attack is executed using Hping3. An sFlowRT DoS application is configured with a threshold to detect network traffic towards any host exceeding 20,000 packets/s. The sFlow sample rate is set to 1:500. Once the threshold is exceeded, an alert is sent to the coordinator. The coordinator then sends a block intent to ONOS for the flow that exceeded the threshold. In this case, detection time is primarily dependent on the sFlow sampling rate configured in the network elements and the processing speed of sFlowRT. We note that sFlow monitoring is specifically selected for defence against the volumetric attack in order to protect the infrastructure from the increased load due to monitoring the volumetric attack. This is further highlighted in Section V-E.

D. System Usability

TENNISON has various aspects that attribute to its ease of use. Table III shows the comparison of lines of code (LoC) for each attack detection application (including imports) between TENNISON and Athena [30]. It is not possible to provide the comparison with other similar security frameworks [23], [24], [29], [35] as either their source code is not openly available or they do not support user applications.

TABLE III: User Application LoC Comparison

System	DDoS	DoS	Intrusion	Scan
TENNISON	107	304	0	135
Athena	1946 [46]	-	-	-

As shown in similar work [23], [30], [47], LoC can be loosely attributed to the development learning curve of a system. The comparison in Table III shows that the application creation and integration with TENNISON is relatively easy. This is due to TENNISON's rich RESTful northbound API, which enables users to easily and succinctly create applications without directly tying to the complexity of the larger system.

E. System Scalability

This section describes TENNISON monitoring scalability. In order to show that TENNISON will operate in a variety of network sizes and topologies, this section also analyses multiple components of the system and compares the results with statistics from real network traces.

1) Multi-Level Monitoring: The limitations of reactive based SDN applications under extreme traffic volumes i.e. heavy controller communication/processing workload are highlighted in [48], [49]. To combat this, TENNISON implements multi-level monitoring, as described in Section IV-D.

A specific optimisation is also applied to protect the network controller against the effects of a traffic flooding scenarios, such as those caused by DDoS attacks. This was introduced in response to an issue identified with ONOS in which the controllers would experience high CPU and memory use, losing communication with the switches and eventually leading to the collapse of the network. The solution makes use of the TENNISON resource monitor and introduces a thresholding mechanism to scale back the volume of monitoring traffic, when appropriate, to prevent the controller and control plane from becoming overwhelmed by traffic. Although ONOS is designed to transfer control to a back-up controller in case of failure, in reality the transfer takes place too late. Our TENNISON solution pre-emptively maintains network control.

The polling rate is adjusted based on three thresholds: **Threshold A:** Once the network traffic reaches threshold A, the TENNISON resource monitor triggers TENNISON to increase the IPFIX polling interval to from 1 s to 5 s. This reduces processing on the controller which aids in performance. **Threshold B:** On breach of the second threshold, the polling interval is increased to 10 s. **Threshold C:** Once the third threshold is exceeded, TENNISON disables the IPFIX polling and increases the sFlow sampling rate. In this situation, TENNISON relies solely on sFlow for network monitoring.

For each threshold, the default timeout is 20 s i.e. after 20 s, TENNISON resets the IPFIX polling interval back to the default (in the case of thresholds A & B) or to re-enable IPFIX polling (in the case of threshold C). The impact of the increased polling interval on DDoS attack detection/protection latency is illustrated in Table IV. The results are averaged across a series of 5 tests with thresholds A, B, and C set to 1 Kpps, 5 Kpps, and 20 Kpps, respectively.

TABLE IV: Impact of polling rate adjustment on DDoS attack detection/protection latency

IPFIX Polling Rate	Protection Time	Protection Time Increment
1 s	7.865 s	-
5 s	8.585 s	+9.154%
10 s	8.500 s	-0.990%

As shown in Table IV, there is a small increment in DDoS protection time based on the increased polling interval. However, the benefit of the adaptive polling interval is to enable continuous attack detection even in a situation of high traffic processing load at the controller. It should be noted that these results have been produced with a multiple controller (3 ONOS instances) test environment.

2) Distributed Control Cost: As previously highlighted, in order to scale to larger networks, an increased number of SDN controllers will be necessary to manage the additional networking devices and requests from the network. As network state information is shared between all of the ONOS controller instances, this may ultimately lead to an increase in the TENNISON detection and remediation times. To determine the potential impact of multiple controller instances, additional experiments were carried out to measure the time to remediate a DDoS attack (as per the test set-up described in Section V-C1) in distributed cluster configurations of varying sizes.

Figure 11 shows the impact (additional delay) of adding ONOS controller instances to the cluster. The results show that after the second instance is added, the time to detect and remediate gradually increases with new instances. Note that this increase in the remediation time is not attributed to the design of TENNISON but is a result of the distributed ONOS implementation (based on state synchronisation). Current research in this area is exploring the optimum design for distributed controllers [50].

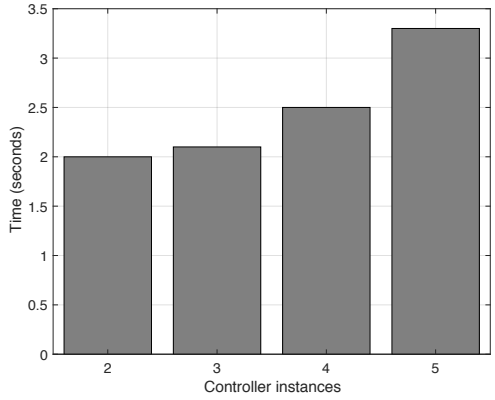


Fig. 11: DDoS Attack Remediation Latency

3) Monitoring Performance Analysis: As described in Section IV, the policy engine in TENNISON stores the thresholds against which the security applications detect an event/attack. The size of this threshold table and the matching algorithm for event detection is a potential source of delay in the system. Figure 12a shows the time that it takes to process an incoming flow against the threshold table within the policy engine for an increasing volume of policies (thresholds). The results indicate that the incurred delay is relatively minor e.g. 500 ms to test a threshold when the policy engine contains 100K thresholds, with the delay increasing approximately linearly.

Figure 12b identifies the time it takes to install a set of monitoring rules. This measurement indirectly shows the maximum capacity of new flows that can be monitored per second. Importantly, this does ***not*** describe the ability of the system to manage throughput, but merely shows that this is the maximum number of new flows that could be monitored per second. The results show that TENNISON can handle up to 10K ***new*** flows in burst and 8K continuously. For flows that have already been observed with monitoring setup, TENNISON will process packets at line rate as per the capabilities of the networking hardware with which TENNISON is deployed.

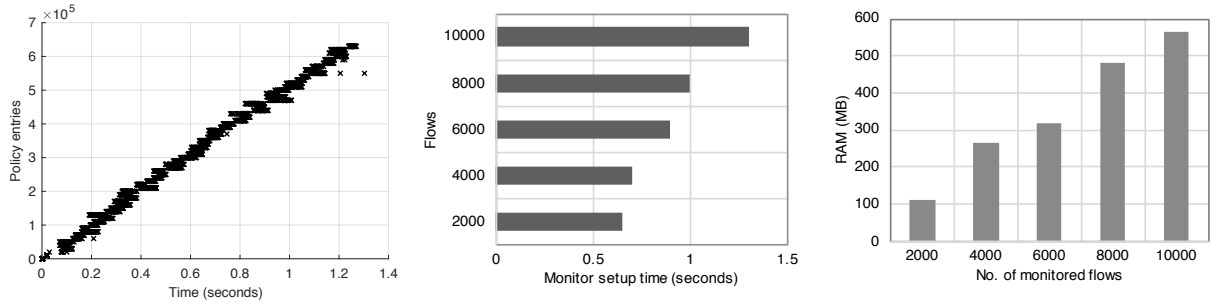
Figure 12c shows the cost of RAM per flow based on actual system readings during the DDoS attack. The memory usage per flow was calculated from the overall system's baseline memory usage and the measured memory usage at different traffic loads. On average, each flow monitored consumes around 64 KB of RAM. This means that with 6 GB of RAM, the system can keep track of 100,000 flows.

4) Summary and discussion of TENNISON*'s scalability:* TENNISON uses both distribution and adaptive escalation techniques to achieve scale whilst maintaining detailed monitoring capability. Based on the results in Figure 12, it is evident that TENNISON can scale to a range of network environments and use cases. For example, one of Facebook's datacenters manages 500 flows per second [51], and in [52], the authors discuss 10 different datacenters that individually support between 20 to 5000 active flows. In these examples, even in the worst case of all active flows starting simultaneously, TENNISON can still monitor each flow without added congestion or packet drops, and using less then 300 MB of RAM.

In cases where the network size exceeds ONOS's maximum switch capacity [53], multiple SDN controller clusters should be deployed into various islands, separating the load of any controller cluster to its respective domain(s). The results in Figures 7, 8 and 11, show the impact of multiple controllers on systems such as TENNISON. Whilst adding more controllers increases monitoring capacity, traffic capacity, and resiliency, it also adds a relatively small delay to remediation time due to the additional delays associated with a distributed system. In order to fulfil uptime requirements, the system should always be deployed with more than one controller. However, the trade-off in remediation time, as illustrated in Figure 11, should be taken into consideration.

VI. CONCLUSIONS AND FUTURE WORK

This paper has presented TENNISON, a multi-level distributed monitoring and remediation framework for SDNs. With a unique security pipeline, TENNISON offers lightweight visibility across a large number of flows. The distributed implementation is supported by multiple control instances, tunneling for efficient attack detection and remediation, and multi-level monitoring. The evaluation illustrates the low latency de-



(a) Policy Engine Performance (b) Monitor Setup Time (c) RAM Usage

Fig. 12: TENNISON Scalability Results

tection and protection capability and the potential for scalable monitoring in large networks. The current implementation supports optimisation of network monitoring and protection based on appropriate controller/switch assignment and optimal monitoring rule placement. Future work includes automation of the scaling process, including provisioning additional controllers to meet increased network load. In addition, the potential for scaling at different layers within the TENNISON architecture will be investigated. For example, introducing distribution at the coordination layer creates additional requirements in terms of consistency, which must be appropriately handled to provide continued coverage and visibility. Finally, as P4 and the concept of SDN matures, new attack definition methods could be introduced to TENNISON along with the potential to further optimise the TENNISON *security pipeline*.

ACKNOWLEDGEMENTS

The authors are grateful to the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the TOUCAN (EP/L020009/1) and NG-CDI (EP/R004935/1) projects, which supported the work presented in this paper, and are grateful to Lancaster University for funding Lyndon Fawcett's PhD studentship in association with TOUCAN.

REFERENCES

[1] "Dyn DDoS Mitigation," https://dyn.com/ddos/.

[2] "DDoS on Dyn Impacts Twitter, Spotify, Reddit and other social media services," https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/.

[3] "OpenFlow Switch Specification Version 1.5.1," Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/sdn-resources/technical-library

[4] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[5] IETF I2NSF (Interface to Network Security Functions). [Online]. Available: https://datatracker.ietf.org/wg/i2nsf/documents/

[6] IETF DOTS (DDoS Open Threat Signaling). [Online]. Available: https://datatracker.ietf.org/wg/dots/documents/

[7] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (nsh)," Tech. Rep., 2018.

[8] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.

[9] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[10] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*. Springer, 2013, pp. 31–41.

[11] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for OpenFlow networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

[13] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[14] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 1956–1961.

[15] C. Liu, A. Malboubi, and C.-N. Chuah, "Openmeasure: Adaptive flow measurement & inference with online learning in sdn," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE, 2016, pp. 47–52.

[16] Y. Taniguchi, H. Tsutsumi, N. Iguchi, and K. Watanabe, "Design and evaluation of a proxy-based monitoring system for openflow networks," *The Scientific World Journal*, vol. 2016, 2016.

[17] "OpenTAM Traffic Analysis and Monitoring." [Online]. Available: https://wiki.onosproject.org/display/ONOS/OPEN-TAM%3A+Traffic+Analysis+and+Monitoring

[18] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers," in *NSDI*, 2016, pp. 311–324.

[19] X. T. Phan and K. Fukuda, "Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processing*, vol. 25, pp. 182–190, 2017.

[20] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Umon: Flexible and fine grained traffic monitoring in open vswitch," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015, p. 15.

[21] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network Monitoring in Software-Defined Networking: A Review," *IEEE Systems Journal*, 2018.

[22] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks." in *USENIX Security Symposium*, vol. 49, 2006, p. 50.

[23] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013.

[24] X.-F. Chen and S.-Z. Yu, "CIPA: A collaborative intrusion prevention architecture for programmable network and SDN," *Computers & Security*, vol. 58, pp. 1–19, 2016.

[25] T. Gamer, "Collaborative anomaly-based detection of large-scale internet attacks," *Computer Networks*, vol. 56, no. 1, pp. 169–185, 2012.

[26] T. Ha, S. Yoon, A. C. Risdianto, J. Kim, and H. Lim, "Suspicious Flow Forwarding for Multiple Intrusion Detection Systems on Software-Defined Networks," *IEEE Network*, vol. 30, no. 6, pp. 22–27, 2016.

[27] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim,

"Suspicious traffic sampling for intrusion detection in software-defined networks," *Computer Networks*, vol. 109, pp. 172–182, 2016.
[28] T. Koulouris, M. Casassa Mont, and S. Arnell, "SDN4S: Software Defined Networking for Security," https://www.hpl.external.hp.com/techreports/2017/HPE-2017-07.pdf, Hewlett Packard Labs, 2017.
[29] T. Yu, S. K. Fayaz, M. Collins, V. Sekar, and S. Seshan, "Psi: Precise security instrumentation for enterprise networks," in *Proc. NDSS*, 2017.
[30] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 249–260.
[31] "TENNISON," https://github.com/SDN-Security/TENNISON/, 2018.
[32] J. Sonchack, J. M. Smith, A. J. Aviv, and E. Keller, "Enabling practical software-defined networking security applications with ofx." in *NDSS*, vol. 16, 2016, pp. 1–15.
[33] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.
[34] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.
[35] Y. Chang, A. Rezaei, B. Vamanan, J. Hasan, S. Rao, and T. Vijaykumar, "Hydra: Leveraging functional slicing for efficient distributed sdn controllers," *arXiv preprint arXiv:1609.07192*, 2016.
[36] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.
[37] "ONOS Topology scaling results," https://wiki.onosproject.org/display/ONOS/1.11-Performance+and+Scale-out.
[38] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 413–424.
[39] S. Scott-Hayward, "Design and deployment of secure, robust, and resilient sdn controllers," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–5.
[40] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for interface to network security functions," Tech. Rep., 2018.
[41] J. Gross, T. Sridhar, P. Garg, C. Wright, I. Ganga, P. Agarwal, K. Duda, D. Dutt, and J. Hudson, "Geneve: Generic network virtualization encapsulation," *IETF draft*, 2014.
[42] J. Halterman and J. Halterman, "Atomix: RAFT based Fault-tolerant distributed coordination framework." [Online]. Available: http://atomix.io/atomix/docs/
[43] C. N. Academy, *Connecting Networks Companion Guide*. Pearson Education, 2014, ch. 1.1.1 Enterprise Network Campus Design.
[44] "Mininet," http://mininet.org.
[45] R. Sherwood and Y. KOK-KIONG, "Cbench: an openflow controller benchmark tool," 2010.
[46] "Athena DDoS user application," https://github.com/shlee89/athena/blob/master/athena-tester/src/main/java/athena/user/application/Main.java.
[47] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags." in *NSDI*, vol. 14, 2014, pp. 543–546.
[48] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of onos reactive forwarding applications in isp networks," *Computer Communications*, vol. 102, pp. 130–138, 2017.
[49] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.
[50] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "Scl: Simplifying distributed sdn control planes." in *NSDI*, 2017, pp. 329–345.
[51] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.
[52] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.
[53] "ONOS Experiment E: Topology Scaling Operation," https://wiki.onosproject.org/display/ONOS/1.10%253A+Experiment+E+-+Topology+Scaling+Operation.

Lyndon Fawcett Lyndon Fawcett is a PhD Student within the School of Computing and Communications at Lancaster University and is associated with the EPSRC funded TOUCAN project. His primary research interests are in applying NFV and SDN to Fog computing infrastructures to enhance network security at the edge. This research encompasses multiple disciplines and entails designing innovative platforms for network and infrastructure orchestration.

Sandra Scott-Hayward Dr. Sandra Scott-Hayward, CEng, is a Lecturer (Assistant Professor) in Network Security at Queen's University Belfast. In the Centre for Secure Information Technologies (CSIT) at QUB, Sandra leads research and development of network security architectures and security functions for software-defined networking (SDN). She has presented her research globally and received Outstanding Technical Contributor and Outstanding Leadership awards from the Open Networking Foundation (ONF) in 2015 and 2016, respectively.

Matthew Broadbent Dr. Matthew Broadbent is a Lecturer in Computer Networks and Networked Systems at Lancaster University. His main research interests lie in the use of software-defined networking, and in particular, their ability to aid service delivery and deploy experimental environments. He has been involved in a number of national and international research projects, including OFELIA, GN3plus, Fed4FIRE, TOUCAN, and more recently, NG-CDI.

Andrew Wright Mr Andrew Wright is a Senior Engineer at Queen's University Belfast, in the Centre for Secure Information Technologies (CSIT). Andrew's interests are in the general area of network and software security but with a focus on using code efficiently to provide secure services especially on low power devices.

Nicholas Race Prof. Nicholas Race is a Professor of Networked Systems within the School of Computing & Communications at Lancaster University. His research is broadly around experimental networking and networked media, specialising in the use of Software Defined Networking (SDN) and Network Functions Virtualisation (NFV) for new network-level services, including in-network media caching, network-level fairness and network monitoring.