

SYSTEM AND METHOD FOR DETECTING ROUTING LOOPS IN A SOFTWARE DEFINED NETWORK (SDN)

Field of the Invention

This invention relates to a system and method for detecting abnormal routing loops in computer networks or more particularly in Software Defined Networks (SDN).

Summary of the Prior Art

A communications protocol that is commonly used by network controllers to determine the path of network packets across a network of switches is the OpenFlow communications protocol. This communications protocol enables network administrators to execute various types of protocols in network switches such as routers, hubs, wireless access points and/or conventional switches. OpenFlow is defined in the OpenFlow Switch Specification, published by the Open Networking Foundation (ONF) where ONF is a consortium of software providers, content delivery networks, and networking equipment vendors whose purpose is to promote software-defined networking.

A key feature of the OpenFlow communications protocol is that it allows network switches residing in one plane to be managed by a controller residing in a different plane. It is this feature that makes this communications protocol ideal for use in Software Defined Networks (SDN).

It is noted that a SDN typically comprises of two separate planes, a control plane and a data plane. This makes the routing mechanism of the network differ greatly from those of conventional networks. Although the SDN provides users with more flexible network controls, the SDN has its own inherent disadvantages. A common challenge faced by such networks is that information security issues may exist between the SDN's control plane and switches in its data plane when the switches receive inconsistent routing information from the controller.

The OpenFlow protocol defines three types of tables in the data plane. The first table comprises a "Flow Table" that matches incoming packets to a particular flow and specifies the functions that are to be performed on the packets. It should be noted that there may be multiple flow tables that may be configured to operate in a pipeline fashion. A flow table may direct a flow to a second type of the table which is a "Group Table". The Group Table contains entries

which may each trigger a variety of actions that affect one or more flows. The third type of table is a "Meter Table" and this table can trigger a variety of performance-related actions on a flow.

Generally, a flow is defined as a sequence of packets traversing a network that share a set of packet header field values. For example, a flow may comprise all packets having the same source and destination IP addresses, or all packets with the same VLAN identifier.

It is useful to note that the function of the controller in the SDN is to calculate flow entries according to received forwarding policies and once calculated, the controller then delivers the flow entries to switches located in the data plane. Upon receiving the flow entries, the switches then create forwarding entries for the Flow Tables and perform forwarding of packets according to the created Flow Tables. It should be noted that different forwarding policies are generated by different applications. For example, a packet filter application will generate a different forwarding policy from that generated by a network isolation application. When policies generated by these applications create loops, routing conflicts then occur as the packets will not be properly forwarded to its intended destination.

Routing conflicts that commonly occur in a SDN comprise loopbacks, black-holes and various other similar types of conflicts that cause unreachability issues. In existing SDN networks, it has been proposed that before network policies arrive at a data forwarding layer, the network policies should be detected and assessed to ascertain whether any vulnerabilities are associated with each network policy.

One solution that has been proposed by those skilled in the art is that prior to the implementation of new network software or configurations, these entries are thoroughly inspected and tested for errors or faults. By running through all the possible code paths of the newly introduced policy entry, one may be able to detect errors and faults in the entry however; such an approach may not be applied for larger policy entries. In addition to the above, the system administrator who has been tasked to carry out checks on the network must have access to the software and configuration of the SDN's controller. This is impractical because in most setups, third parties are usually not granted administrator access to the SDN's controller and it is a big task to expect the SDN's administrator to thoroughly inspect and check all new entries individually. Further, these approaches operate offline and as such, are only able to detect errors or faults after they have occurred.

Another solution that has been proposed involves the addition of an intermediate layer between a SDN's control plane and its data plane. The solution of adding this intermediate layer between the control plane and the data plane refers to the addition of a middle layer that is configured to intercept interaction information (e.g. packet-in or flow-mod instructions) between the controller and the switches and for extracting network status information. Newly introduced rules will then be detected, intercepted and verified at this middle layer before the rules are delivered to the data forwarding layer. By doing so, this ensures that rules that are eventually delivered to the data forwarding layer do not conflict with existing rules of the SDN network.

However, the solutions proposed thus far are unable to react sufficiently fast. As a result, the proposed solutions are unable to detect and verify newly introduced policies or forwarding rules in real-time. Even though if they are able to detect and verify these new policies, the existing solutions are unable to propose an appropriate solution to address the conflicts detected. Further, the solutions proposed thus far are also unable to detect policy violations involving packet header modifications. In particular, packet headers that are rewritten or are encapsulated by encapsulation operations are unable to be detected by existing solutions.

For the above reasons, those skilled in the art are constantly striving to come up with a device and method to detect and prevent routing loops involving header modification in software defined networks in real-time.

Summary of the Invention

Systems and methods for detecting abnormal routing loops in a computer network when a new packet forwarding rule modification is introduced to the computer network are proposed, and these systems and methods are set out below in accordance with embodiments of the invention.

A first improvement proposed by embodiments of systems and methods in accordance with the invention is that forwarding loops arising due to the introduction of a new packet forwarding rule modification to the computer system may be detected even when the new packet forwarding rule modification comprises a packet header modification instruction or when the new packet forwarding rule modification affects an existing packet forwarding rule that comprises a packet header modification instruction.

A second improvement proposed by embodiments of systems and methods in accordance with the invention is that through the use of a multidimensional prefix tree (Trie tree), affected equivalence classes and their associated connected equivalence classes may be quickly and efficiently located thereby allowing for the detection of abnormal routing loops in real time.

The above improvements are provided by embodiments in accordance with the invention operating in the following manner.

According to a first aspect of the invention a verification system for detecting abnormal routing loops in a computer network is disclosed, the system comprising: a loop detecting module positioned between network devices of the computer network and a controller of the computer network, the loop detecting module configured to: intercept a packet rule modification initiated by the controller before the packet rule modification is received by the network devices; identify equivalence classes affected by the intercepted packet rule modification; determine connected equivalence classes associated with each affected equivalence class; and verify routing of packets affected by the intercepted packet rule modification using the affected equivalence classes and the connected equivalence classes associated with each of the affected equivalence classes.

According to an embodiment of the first aspect of the disclosure, the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises the loop detecting module being configured to: group packets sent through the computer network into a plurality equivalence classes whereby each equivalence class comprises a set of packets that are affected by a set of common packet rules; and determine an equivalence class affected by the intercepted packet rule modification.

According to an embodiment of the first aspect of the disclosure, the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises the loop detecting module being configured to: apply the intercepted packet rule modification to a multidimensional prefix tree (Trie tree) generated using existing packet rules, whereby the Trie tree comprises a plurality of sub-trees, whereby each sub-tree represents a packet header domain and each leaf of the Trie tree stores an associated set of packet rules; identify existing packet rules affected by the intercepted packet rule modification by traversing the Trie tree to determine leaves having existing packet rules that overlap with the intercepted packet rule

modification; and generate the affected equivalence classes based on the overlapping existing packet rules.

Optionally, the step of determining the connected equivalence classes associated with each affected equivalence class comprises the loop detecting module being configured to: identify affected equivalence classes having a packet rule or a packet that involves a header modification instruction; and generate connected equivalence classes for each of the identified equivalence classes.

Optionally, the step of generating the connected equivalence classes for each of the identified equivalence classes comprises the loop detecting module being configured to: determine existing packet rules that overlap with packets of the identified equivalence classes before or after header modification of the packets of the identified equivalence classes utilizing the multidimensional prefix tree (Trie tree); construct connected equivalence classes from overlapping rules.

Optionally, the step of constructing connected equivalence classes from overlapping rules for each of the identified equivalence classes comprises the loop detecting module being configured to: for each of the identified equivalence classes, generate a forward filter whereby the forward filter comprises packets after the header modification; determine existing packet rules that overlap with the forward filter by applying the forward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the forward filter; and group packets affected by the existing packet rules that overlap with the forward filter to form connected equivalence classes.

According to an embodiment of the first aspect of the disclosure, the step of constructing connected equivalence classes from overlapping rules for each of the identified equivalence classes further comprises the loop detecting module being configured to: for each of the identified equivalence classes, generate a first backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determine existing packet rules that overlap with the first backward filter by applying the first backward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first backward filter; and group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes.

According to an embodiment of the first aspect of the disclosure, the step of constructing connected equivalence classes from overlapping rules comprises the loop detecting module being configured to: for each of the identified equivalence classes, generate a first backward filter and a second backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction and the second backward filter comprises a second set of original packets before the second set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determine existing packet rules that overlap with the first and second backward filters by applying the first and second backward filters to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first and second backward filters; and group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes associated with the first backward filter and group packets affected by the existing packet rules that overlap with the second backward filter to form connected equivalence classes associated with the second backward filter.

According to an embodiment of the first aspect of the disclosure, the step of verifying the routing of packets affected by the intercepted packet rule modification comprises the loop detecting module being configured to: construct forwarding graphs for each of the affected equivalence classes using connected equivalence classes associated with each of the affected equivalence classes whereby each of the forwarding graphs illustrate the routing of packets within each of the affected equivalence classes and the associated connected equivalence classes; and determine occurrence of routing loops based on the forwarding graphs.

According to an embodiment of the first aspect of the disclosure, the loop detecting module is further configured to: apply the packet rule modification into a leaf of the Trie tree associated with the packet rule modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the first aspect of the disclosure, the loop detecting module is further configured to: apply the packet rule modification into a leaf of the Trie tree associated with the forward filter of the rule if the packet rule modification has an action of packet header modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the first aspect of the disclosure, the intercepted packet rule modification comprises instructions to add a new packet rule or to amend or delete an existing packet rule.

According to an embodiment of the first aspect of the disclosure, the intercepted packet rule modification comprises instructions to forward packets, modify packet headers or both.

According to a second aspect of the invention, a loop detecting module provided within a verification system for detecting abnormal routing loops in a computer network is disclosed, the loop detecting module being positioned between network devices of the computer network and a controller of the computer network, the loop detecting module comprising: a processor; and a non-transitory media readable by the processor, the non-transitory media storing instructions that when executed by the processor, cause the processor to: intercept a packet rule modification initiated by the controller before the packet rule modification is received by the network devices; identify equivalence classes affected by the intercepted packet rule modification; determine connected equivalence classes associated with each affected equivalence class; and verify routing of packets affected by the intercepted packet rule modification using the affected equivalence classes and the connected equivalence classes associated with each of the affected equivalence classes.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor to identify the equivalence classes affected by the intercepted packet rule modification comprises: instructions for directing the processor to group packets sent through the computer network into a plurality equivalence classes whereby each equivalence class comprises a set of packets that are affected by a set of common packet rules; and determine an equivalence class affected by the intercepted packet rule modification.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor to identify the equivalence classes affected by the intercepted packet rule modification comprises: instructions for directing the processor to: apply the intercepted packet rule modification to a multidimensional prefix tree (Trie tree) generated using existing packet rules, whereby the Trie tree comprises a plurality of sub-trees, whereby each sub-tree represents a packet header domain and each leaf of the Trie tree stores an associated set of packet rules; identify existing packet rules affected by the intercepted packet rule modification by traversing the Trie tree to determine leaves having existing packet rules that overlap with the

intercepted packet rule modification; and generate the affected equivalence classes based on the overlapping existing packet rules.

Optionally, the instructions for directing the processor to determine the connected equivalence classes associated with each affected equivalence class comprises: instructions for directing the processor to: identify affected equivalence classes having a packet rule or a packet that involves a header modification instruction; and generate connected equivalence classes for each of the identified equivalence classes.

Optionally, the instructions for directing the processor to generate the connected equivalence classes for each of the identified equivalence classes comprises: instructions for directing the processor to: determine existing packet rules that overlap with packets of the identified equivalence classes before or after header modification of the packets of the identified equivalence classes utilizing the multidimensional prefix tree (Trie tree); construct connected equivalence classes from overlapping rules.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor to construct connected equivalence classes from overlapping rules comprises: instructions for directing the processor to: for each of the identified equivalence classes, generate a forward filter whereby the forward filter comprises packets after the header modification; determine existing packet rules that overlap with the forward filter by applying the forward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the forward filter; and group packets affected by the existing packet rules that overlap with the forward filter to form connected equivalence classes.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor construct connected equivalence classes from overlapping rules comprises: instructions for directing the processor to: for each of the identified equivalence classes, generate a first backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determine existing packet rules that overlap with the first backward filter by applying the first backward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first backward filter; and group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor to construct connected equivalence classes from overlapping rules comprises: instructions for directing the processor to: for each of the identified equivalence classes, generate a first backward filter and a second backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction and the second backward filter comprises a second set of original packets before the second set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determine existing packet rules that overlap with the first and second backward filters by applying the first and second backward filters to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first and second backward filters; and group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes associated with the first backward filter and group packets affected by the existing packet rules that overlap with the second backward filter to form connected equivalence classes associated with the second backward filter.

According to an embodiment of the second aspect of the disclosure, the instructions for directing the processor to verify the routing of packets affected by the intercepted packet rule modification comprises: instructions for directing the processor to: construct forwarding graphs for each of the affected equivalence classes using connected equivalence classes associated with each of the affected equivalence classes whereby each of the forwarding graphs illustrate the routing of packets within each of the affected equivalence classes and the associated connected equivalence classes; and determine occurrence of routing loops based on the forwarding graphs.

According to an embodiment of the second aspect of the disclosure, the loop detecting module further comprises: instructions for directing the processor to: apply the packet rule modification into a leaf of the Trie tree associated with the packet rule modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the second aspect of the disclosure, the loop detecting module further comprises instructions for directing the processor to: apply the packet rule modification into a leaf of the Trie tree associated with the forward filter of the rule if the packet

rule modification has an action of packet header modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the second aspect of the disclosure, the intercepted packet rule modification comprises instructions to add a new packet rule or to amend or delete an existing packet rule.

According to an embodiment of the second aspect of the disclosure, the intercepted packet rule modification comprises instructions to forward packets, modify packet headers or both.

According to a third aspect of the invention, a method for detecting abnormal routing loops in a computer network using a loop detecting module positioned between network devices of the computer network and a controller of the computer network comprises: intercepting a packet rule modification initiated by the controller before the packet rule modification is received by the network devices; identifying equivalence classes affected by the intercepted packet rule modification; determining connected equivalence classes associated with each affected equivalence class; and verifying routing of packets affected by the intercepted packet rule modification using the affected equivalence classes and the connected equivalence classes associated with each of the affected equivalence classes.

According to an embodiment of the third aspect of the disclosure, the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises: grouping packets sent through the computer network into a plurality equivalence classes whereby each equivalence class comprises a set of packets that are affected by a set of common packet rules; and determining an equivalence class affected by the intercepted packet rule modification.

According to an embodiment of the third aspect of the disclosure, the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises: applying the intercepted packet rule modification to a multidimensional prefix tree (Trie tree) generated using existing packet rules, whereby the Trie tree comprises a plurality of sub-trees, whereby each sub-tree represents a packet header domain and each leaf of the Trie tree stores an associated set of packet rules; identifying existing packet rules affected by the intercepted packet rule modification by traversing the Trie tree to determine leaves having existing packet

rules that overlap with the intercepted packet rule modification; and generating the affected equivalence classes based on the overlapping existing packet rules.

According to an embodiment of the third aspect of the disclosure, the step of determining the connected equivalence classes associated with each affected equivalence class comprises: identifying affected equivalence classes having a packet rule or a packet that involves a header modification instruction; and generating connected equivalence classes for each of the identified equivalence classes.

According to an embodiment of the third aspect of the disclosure, the step of generating the connected equivalence classes for each of the identified equivalence classes comprises: determining existing packet rules that overlap with packets of the identified equivalence classes before or after header modification of the packets of the identified equivalence classes utilizing the multidimensional prefix tree (Trie tree); and constructing connected equivalence classes from overlapping rules.

Optionally, the step of constructing connected equivalence classes from overlapping rules comprises: for each of the identified equivalence classes, generating a forward filter whereby the forward filter comprises packets after the header modification; determining existing packet rules that overlap with the forward filter by applying the forward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the forward filter; and grouping packets affected by the existing packet rules that overlap with the forward filter to form connected equivalence classes.

Optionally, the step of constructing connected equivalence classes from overlapping rules comprises: for each of the identified equivalence classes, generating a first backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determining existing packet rules that overlap with the first backward filter by applying the first backward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first backward filter; and grouping packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes.

Optionally, the step of constructing connected equivalence classes from overlapping rules comprises: for each of the identified equivalence classes, generating a first backward filter

and a second backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction and the second backward filter comprises a second set of original packets before the second set of original packets are modified into packets of the identified equivalence class by the header modification instruction; determining existing packet rules that overlap with the first and second backward filters by applying the first and second backward filters to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first and second backward filters; and grouping packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes associated with the first backward filter and group packets affected by the existing packet rules that overlap with the second backward filter to form connected equivalence classes associated with the second backward filter.

According to an embodiment of the third aspect of the disclosure, the step of verifying the routing of packets affected by the intercepted packet rule modification comprises: constructing forwarding graphs for each of the affected equivalence classes using connected equivalence classes associated with each of the affected equivalence classes whereby each of the forwarding graphs illustrate the routing of packets within each of the affected equivalence classes and the associated connected equivalence classes; and determining occurrence of routing loops based on the forwarding graphs.

According to an embodiment of the third aspect of the disclosure, the method further comprises: applying the packet rule modification into a leaf of the Trie tree associated with the packet rule modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the third aspect of the disclosure, the method further comprises: applying the packet rule modification into a leaf of the Trie tree associated with the forward filter of the rule if the packet rule modification has an action of packet header modification when the routing of packets affected by the intercepted packet rule modification are verified.

According to an embodiment of the third aspect of the disclosure, the intercepted packet rule modification comprises instructions to add a new packet rule or to amend or delete an existing packet rule.

According to an embodiment of the third aspect of the disclosure, the intercepted packet rule modification comprises instructions to forward packets, modify packet headers or both.

Brief Description of the Drawings

The above advantages and features in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating an exemplary block diagram of a Software Defined Network having a loop detection system module in accordance with embodiments of the invention;

Figure 2 illustrating a block diagram representative of components in an electronic device for implementing embodiments in accordance with embodiments of the invention;

Figure 3 illustrating a block diagram of a loop detection system in accordance with embodiments of the invention together with a flow diagram showing the interaction of external modules with internal modules contained within the virtual machine;

Figure 4 illustrating an extended Trie tree in accordance with embodiments of the invention;

Figure 5 illustrating a flow diagram showing the process flow of a newly introduced rule being detected and attended to in accordance with embodiments of the invention;

Figure 6 illustrating a flow diagram showing the process flow of a newly introduced rule being detected and attended to by an adaptive resolution module in accordance with embodiments of the invention; and

Figure 7 illustrating a flow diagram showing the process flow of a newly introduced rule having a packet header modification command being detected and attended to in accordance with embodiments of the invention.

Detailed Description

This invention relates to a system and method for detecting abnormal routing loops caused by packet forwarding conflicts in computer networks or more particularly in Software Defined Networks (SDN) when new packet forwarding rules are introduced to the SDN network.

The system and method may also be configured to detect flow entries or packet forwarding rules having modified packet header domains. In embodiments of the invention, once abnormal routing loops are detected, the system may then be configured to adaptively propose a solution to address or resolve the abnormal routing loop.

In software defined networks (SDNs), a SDN controller is configured to provide management control for network devices provided in the SDN network's data plane. An exemplary communications protocol that enables the SDN controller to remotely manage network devices in the data plane of the SDN network is the OpenFlow communications protocol. This protocol, which originated as an open standard, enables researchers to run experimental protocols in commercial Ethernet switches, routers and wireless access points. Although reference is made to the OpenFlow communications protocol throughout the description, one skilled in the art will recognize that other SDN compatible communications protocols may be also be utilized without departing from the invention.

One skilled in the art will also recognize that many functional units in this description have been labelled as modules throughout the specification. The person skilled in the art will also recognize that modules may be implemented as electronic circuits, logic chips or any combination of electrical and/or electronic discrete components within a device. Further, one skilled in the art will also recognize that a module may be implemented in software which may then be executed by a variety of processors. In embodiments of the invention, a module may also comprise computer instructions or executable code that may instruct a computer processor to carry out a sequence of events based on instructions received. The choice of the implementation of the modules is left as a design choice to a person skilled in the art and does not limit the scope of this invention in any way.

Figure 1 illustrates a block diagram of a SDN network in accordance with embodiments of the invention. In particular, Figure 1 illustrates SDN network 100 which comprises SDN controller 105, loop detection system (LDS) module 115 and network 140, which are provided with network devices 141. One skilled in the art will recognize that network devices 141 may include any type of device that may be configured to connect various types of electronic devices together so that these electronic devices may share resources or files by forwarding data packets between themselves in accordance with associated flow entries. Network devices 141 may include, but are not limited to switches, routers, access points or any virtualized versions of such physical devices.

SDN controller 105 is provided in SDN network 100's control plane and is configured to provide routing commands or packet-forwarding rules to network devices 141 in network 140. SDN controller 105 is the core of SDN network 100 and controls all traffic forwarding logic within SDN network 100. Located within SDN controller 105 are SDN applications that utilize controller 105 to forward routing instructions to network devices within network 140. These routing instructions are typically transmitted via northbound interface 110.

As illustrated in Figure 1, LDS module 115 has been provided between controller 105 and network 140 such that LDS module 115 may intercept any new rules conveyed from SDN controller 105 through southbound interface 110. In embodiments of the invention, LDS module 115 may instead be provided within SDN controller 105 and be directly connected to southbound interface 110 of controller 105.

Any new rule modifications intercepted by LDS module 115 will then be assessed and verified by extended multidimensional prefix tree (Trie tree) module 120 and forwarding graph module 125. As illustrated in Figure 1, these two modules are provided within LDS module 115. The detailed workings of these two modules will be described in the later sections below.

A person skilled in the art will understand that references in this description to a rule refers to a packet forwarding rule that typically comprises of three main parts, i.e. the rule's filter, an action list associated with the rule and the rule's priority order. Each of these rules are then to be installed within a switch whereby in certain scenarios, more than one rule may be installed within a single switch. Packets having headers that match a rule's filter will then be processed by an associated switch in accordance with the rule's action list. As a single switch may be installed with more than one rule, there is the possibility that a packet may be affected by more than one rule, i.e. the packet's header may be contained within the filters of two different rules due to an overlap of the rules' filters. When such a situation occurs, the action list associated with the rule having the higher priority order will be executed in place of the action list associated with the rule having the lower priority order.

Once the new rule modifications have been verified, the new rule modifications will then be forwarded onto network 140 via southbound interface 135 for execution by network devices 141 that are impacted by the rule modifications. It should be noted that new rule modifications refer to, but are not limited to, the introduction of a completely new rule, the deletion of an existing rule, or a revision or update to an existing rule.

If any of the rule modifications cause conflicts to occur with existing packet forwarding rules in SDN network 100, the rule modification that caused the conflict will then be forwarded to adaptive resolution module 130 to be handled or in embodiments of the invention; LDS module 116 may notify network 100's administrator of the occurrence of the conflict as indicated by arrow 117.

The detailed workings of SDN controller 105 and network 140 have been omitted for brevity as the functions performed by these two components are known to those skilled in the art.

Figure 2 illustrates a block diagram of a computer system architecture having components that may be provided within modules 115, 120, 125 and/or 130 and within interfaces 110 and 135 for implementing embodiments in accordance with embodiments of the invention. One skilled in the art will recognize that the exact configuration of components provided within each module may be different and may vary and Figure 2 is provided by way of example only.

In embodiments of the invention, computer system architecture 200 comprises controller 201 and user interface 202. User interface 202 is arranged to enable manual interactions between a user and module 200 and for this purpose includes the input/output components required for the user to enter instructions to control module 200. A person skilled in the art will recognize that components of user interface 202 may vary from embodiment to embodiment but will typically include one or more of display 240, keyboard 235 and track-pad 236.

Controller 201 is in data communication with user interface 202 via bus 215 and includes memory 220, processor 205 mounted on a circuit board that processes instructions and data for performing the method of this embodiment, an operating system 206, an input/output (I/O) interface 230 for communicating with user interface 202 and a communications interface, in this embodiment in the form of a network card 250. Network card 250 may, for example, be utilized to send data from electronic device 200 via a wired or wireless network to other processing devices or to receive data via the wired or wireless network. Wireless networks that may be utilized by network card 250 include, but are not limited to, Wireless-Fidelity (Wi-Fi), Bluetooth, Near Field Communication (NFC), cellular networks, satellite networks, telecommunication networks, Wide Area Networks (WAN), etc.

Memory 220 and operating system 206 are in data communication with CPU 205 via bus 210. The memory components include both volatile and non-volatile memory and more than one of each type of memory, including Random Access Memory (RAM) 220, Read Only Memory (ROM) 225 and a mass storage device 245, the last comprising one or more solid-state drives (SSDs) or any other types of drives. Memory 220 also includes secure storage 246 for securely storing secret keys, or private keys. It should be noted that the contents within secure storage 246 might only be accessible by a super-user or administrator of module 200 and might not be accessed by any user of module 200. One skilled in the art will recognize that the memory components described above comprise non-transitory computer-readable media and shall be taken to comprise all computer-readable media except for a transitory, propagating signal. Typically, the instructions are stored as program code in the memory components but can also be hardwired. Memory 220 may include a kernel and/or programming modules such as a software application that may be stored in either volatile or non-volatile memory.

Herein the term “processor” is used to refer generically to any device or component that can process such instructions and may include: a microprocessor, microcontroller, programmable logic device or other computational device. That is, processor 205 may be provided by any suitable logic circuitry for receiving inputs, processing them in accordance with instructions stored in memory and generating outputs (for example to the memory components or on display 240). In this embodiment, processor 205 may be a single core or multi-core processor with memory addressable space. In one example, processor 205 may be multi-core, comprising—for example—an 8 core CPU.

Figure 3 illustrates a block diagram of various modules contained within LDS module 115 such as extended Trie tree module 120, forwarding graph module 125 and adaptive resolution module 130. When a packet forwarding rule event 301 occurs, rules 301 that are intercepted by LDS module 115 will be provided to extended Trie tree module 120 in LDS module 115. It should be noted that prior to receiving the new rule, a Trie tree based on the system's existing packet forwarding rules would have been pre-generated by a Trie tree module (not shown). In this description, a Trie tree is essentially an associative structural array that has been arranged as an ordered tree data structure. The function of the Trie tree is to associate packets affected by a packet-forwarding rule with the packet-forwarding rule itself using the packet's header whereby each level in the Trie tree corresponds to a specific bit of a packet

header (equivalently, a bit in the packet header) affected by a rule. An example of such a Trie tree is illustrated in Figure 4.

The Trie tree illustrated in Figure 4 comprises of a plurality of nodes or levels, e.g. nodes 415, 416, 417, 418, 419, and a few of these nodes may be grouped together to form a sub-tree or a Dimension. One skilled in the art will recognize that the Trie tree may comprise of any number of nodes as required and that nodes 415-419 are shown in Figure 4 for illustration purposes only. It should be noted that each sub-tree or Dimension may comprise of any number of nodes or levels as required. In general, the overall Trie tree may be described as a composition of several sub-trees or Dimensions, each corresponding to a packet header field of a packet structure.

For example, in an IPv4 packet structure, sub-tree D-1 may refer to the "version" field which comprises of 4 bits and as such will comprise of 4 levels in the sub-tree D-1 while sub-tree D-2 may refer to the "Destination IP address" field which comprises of 32 bits and as such will comprise of 32 levels in the sub-tree D-2, and etc. As a typical IPv4 packet structure comprises around 14 fields, including the "Options" field, and if all 14 fields are to be represented by the Trie tree, this means that integer "N" of D-N in Figure 4 will be 14, i.e. there will be 14 Dimensions (D-14) as each Dimension represents a packet header field of the IPv4 packet structure. Although this embodiment exemplifies the Trie tree of an IPv4 packet structure as having 14 Dimensions, one skilled in the art will recognize that any combinations of the Dimensions or any number of fields may be selected to form the Trie tree without departing from this invention.

In another embodiment of the invention, if the Trie tree in Figure 4 were to be applied to an OpenFlow packet structure, each sub-tree of the Trie tree would represent a packet header domain of the OpenFlow protocol. As the OpenFlow protocol defines 14 to 40 matching domains, this means that the Trie tree may have to be configured to maintain multiple-dimension subtrees with each node of each subtree indicating a specific bit of the matching domain. In embodiments of the invention, the Trie tree may be maintained for only four domains of the OpenFlow protocol, which are the "DL SRC" – Ethernet Source Address, "DL DST" – Ethernet Destination Address, "NW SRC" – IP Source Address and "NW DST" – IP Destination Address domains, as these domains are the ones that may support arbitrary wildcards and as such, may result in overlaps occurring between different rules.

Regardless of the type of packet structure represented by the Trie tree, these multiple-dimension subtrees may then be utilized by LDS module 115 to detect loop conflicts generated across multiple dimensions.

Further, as illustrated in circle 420a and expanded upon in circle 420b, each level or node in the Trie tree comprises three branches that each correspond to three possible values of a bit in the packet header: "x" (425), "1" (426) and "0" (427), where "x" refers to a wildcard. Depending on the specific bit in a packet header, the bit of the packet header may be assigned to either the wildcard "x" branch, the "1" branch or the "0" branch.

Each of the Trie tree's leaves, i.e. points at the bottom of the Trie tree (e.g. leaves A-F), are further configured to store rules corresponding to the leaf of the Trie tree and the path traced from the root of the Trie tree to the leaf represents the packet fields that are affected by the rule. In other words, for a path traced from the Trie tree's root to the leaf at the bottom of the Trie tree, this means that this path represents a rule having a filter that specifies all the bits traced by the path. For example, it is assumed that leaf A contains Rule A having a filter specifying a Source IP Address of "1010xxxx". This means that the path from the root of the Trie tree to leaf A would include the following bits: "1, 0, 1, 0, x, x, x, x". As such, any packet header having a Source IP Address of "1010xxxx" would be affected by Rule A.

In Figure 4, assuming that a path originating from the root of the Trie-tree is traced along the relevant branches of subtree D-1, along the relevant branches of subtree D-2 and ends at leaf A of D-N, this means that the packet header fields along this path are affected by the rules stored at leaf A (or covered by a filter of a rule at leaf A). Figure 4 also illustrates leaves B, C, D, E and F however one skilled in the art will recognize that the Trie tree may comprise of any number of leaves as required and that leaves A-F are shown for illustration purposes only. Once the Trie tree for the system has been created, the Trie tree may then be used to find equivalence classes that are affected by an existing and/or newly introduced rule quickly and efficiently.

An equivalence class essentially comprises a set of packets (defined by their packet headers) that are grouped together as these packets are affected by the same set of packet forwarding rules, i.e. covered by the rule's filter range. This means that all the packets contained within this single equivalence class will have packet headers that fall within a range covered by these rules' filters. In other words, packets having headers that do not fall within the

filter of the equivalence class (the filter defining the above range) will not be included into the equivalence class.

For example, it is assumed that there exists two packet forwarding rules that affect packets having a destination IP address field specifying IP address prefixes 55.1.0.0/16 and 56.1.0.0/16, i.e. Rule-1 having a filter: (55.1.0.0/16) and Rule-2 having a filter: (56.1.0.0/16). When a new rule Rule-New having a filter: (55.0.0.0/8), is introduced, this new rule will only affect packets having a destination field specifying IP address prefix 55.0.0.0/8. As such, this Rule-New will only affect the Rule-1 or only be affected by Rule-1 depending on their priorities. This means that packets having a destination field in the 55.1.0.0/16 range will only be affected and this newly introduced rule will not influence packets having a destination field outside the 55.0.0.0/8 range, i.e. packets having a destination field 56.1.0.0/16 will not be affected.

Therefore, the effect of overlapping Rule-New (55.0.0.0/8) with existing Rule-1 (55.1.0.0/16) on the forwarding of packets in the network may be collectively considered by dividing affected packets into equivalence classes using the packets' destination field. In this example, the first equivalence class represents packets having a destination IP address field specifying IP address prefixes at the lower bound range or having a first filter range which is between 55.0.0.0 to 55.0.255.255. The second equivalence class represents packets having a destination IP address field specifying IP address prefixes at the mid-bound range or having a second filter range which is between 55.1.0.0 to 55.1.255.255. Finally, the third equivalence class represents packets having a destination IP address field specifying IP address prefixes at the upper bound range or having a third filter range which is between 55.2.0.0 to 55.255.255.255. In this example, the first and third equivalence classes are considered as Rule-New's directly affected equivalences classes as these equivalence classes' filters overlap with Rule-New's filter and Rule-New take effects in these two filters' ranges. The second equivalence class's filter overlaps with both Rule-New's filter and Rule-1's filter. However, as Rule-1's priority order is assumed to be higher than Rule-New's priority order, Rule-New will not affect the packets in the second equivalence class.

As such, if the existing Rule-1 (55.1.0.0/16) causes an affected packet to be forwarded to Switch A and if the new Rule-New (55.0.0.0/8) causes an affected packet to be forwarded to Switch B, this means that affected packets grouped in the first equivalence class will be forwarded to Switch B, packets grouped in the second equivalence class will be forwarded to

Switch A (because Rule-1 (55.1.0.0/16) has higher priority order than the new Rule-New) while affected packets grouped in the third equivalence class will be forwarded to Switch B.

In summary, each time a new packet forwarding rule is added to the network, or when the network's existing packet forwarding rule is modified or deleted; this rule will cause the flow of packets specified by the rule's filter or subset(s) of the rule's filter to change in accordance with the rule's action list, or in the case of a deletion of a rule, the action list of the rule will no longer be applied to the affected packets. However, this change typically only affects a small sample of directly affected equivalence classes in the network. Hence, by utilizing the concept of directly affected equivalence classes, the network does not need to assess each packet individually to determine if the forwarding of the packet is affected by new/modified/deleted rules or new/modified/deleted rules that overlap with existing rules. This assessment instead may be done collectively as a group based on each directly affected equivalence class.

Returning to Figure 3, to recap, when a packet forwarding rule event 301 occurs, rule 301 that has been intercepted by LDS module 115 will be provided to extended Trie tree module 120. One skilled in the art will recognize that rule 301 may refer to the addition, modification or deletion of a rule. For brevity, although the subsequent parts of the description may refer to rule 301 as an addition of a new rule; one skilled in the art will recognize that the following parts in the description may be applied to a version of rule 301 that comprises a modification or deletion of a rule as well without departing from the invention.

Once Trie tree module 120 ascertains that rule 301 does not include an action that causes a packet's header to be modified, module 120 will then trigger module 305 to apply rule 301 to the network's existing Trie tree. This means that module 305 will determine which existing rules will overlap with rule 301. This can be done by traversing the Trie tree, subtree by subtree, to determine rules (located at the leaves) that overlap with new rule 301. For each subtree, module 305 will limit the area to be traversed by selectively traversing branches of the subtree that fall within the range indicated by rule 301's filter.

Existing rules that overlap with rule 301 and rule 301 itself are then provided to module 310. Module 310 then proceeds to generate directly affected equivalence classes that are affected by rule 301. Module 310 is then configured to determine if each directly affected equivalence class is associated with a connected equivalence classes. Module 310 achieves

this by determining if existing rules involve any header modification rules connecting a directly affected equivalence class to its connected classes.

It is useful at this stage to note that a connected equivalence class is defined as another equivalence class that contains some or all packets (having modified headers) that initially belonged to an originating equivalence class whereby some or all packets belonging to the originating equivalence class becomes the another equivalence class's packets after the packets' headers have been modified. The originating equivalence class is also a connected equivalence class of another equivalence class. We also say the former class is the latter class's upstream class and the latter is the former's downstream class. The downstream (resp. upstream) class may have its own downstream (resp. upstream) class as well. The downstream (resp. upstream) class's downstream (resp. upstream) class is also considered a connected class if some packets can change the equivalence classes they belong to twice. In summary, if some or all packets of an equivalence class become packets of another equivalence class after one or more times of header modifications, these two classes are connected. It should be noted that a class may have more than one downstream class or more than one upstream class. As such, it can be said that the originating equivalence class is indirectly linked with the connected equivalence classes through the modification of the packet headers.

For example, with reference to Figure 4, it is assumed that an existing rule A1 modifies a destination field in the headers of packets contained within a first directly affected equivalence class affected by rule A1 and that rule A1 is the only header modification rule to be processed. In this example, it is assumed that rule A1 modifies the headers of packets in the equivalence class such that for a packet in header range X, this packet's destination field is changed to fall in header range Y. Module 310 will therefore generate a forward filter for each header modification rule taking effect in a directly affected equivalence class. In this example, a forward filter will be generated for the equivalence class's packets modified by rule A1. The forward filter will then be used to find downstream equivalence classes. Similarly, a backward filter will be used to find upstream equivalence classes in embodiments with a header modification rule modifying packet headers and turning packets into the packets of a directly affected equivalence class. In embodiments of the invention, it should be noted that multiple header modification rules may coexist simultaneously. In such an embodiment, all the header modification rules connecting directly affected equivalence classes and their connected classes have to be processed as well.

If it is determined that existing rules involve header modification rules, module 310 will then generate a forward filter for each header modification rule taking effect in the directly affected equivalence class. The function of the forward filter is to define the equivalence class's packets with headers that have been modified by the action of the corresponding header modification rule. Module 310 then utilizes the Trie tree to find leaf nodes (rules) having filters that overlap with the forward filters. These overlapping leaf nodes (rules) are then gathered together to form one or more connected equivalence classes that are indirectly linked with the original directly affected equivalence class. These equivalence classes are downstream classes. Similarly, module 310 finds upstream classes through generating a backward filter for each header modification rule connecting a directly affected equivalence class and upstream class(es). The function of the backward filter is to define the original packets that will become the packets of the directly affected equivalence class after header modification of the original packets by the action of the corresponding header modification rule. It should be noted that an equivalence class may comprise one or more backward filters.

For example, it is assumed that an equivalence class's packet header is 111xxxx. In this example, Rule 1 caused an original packet header of 010xxxx to change to 111xxxx while Rule 2 caused an original packet header of 101xxxx to change to 111xxxx. This means that a backward filter associated with Rule 1 would contain 010xxxx while another backward filter associated with Rule 2 would contain 101xxxx.

Module 310 then utilizes the Trie tree to find leaf nodes (rules) having filters that overlap with the backward filter. These overlapping leaf nodes (rules) are then gathered together to form one or more connected equivalence classes that are indirectly linked with the original directly affected equivalence class. Any downstream (resp. upstream) class's downstream (resp. upstream) classes could be connected equivalence classes. Therefore, module 310 also finds downstream (resp. upstream) classes for each downstream (resp. upstream) class with above method until all connected classes are found. The directly affected equivalence classes are then provided to forwarding graph module 125.

Conversely, if it is determined that existing rules do not involve header modification rules, module 310 will then provide the directly affected equivalence classes to forwarding graph module 125.

Forwarding graph module 125 then utilizes module 315 to generate a forwarding graph for each directly affected equivalence class. A forwarding graph is essentially a graph showing the path of how data packets within the directly affected equivalence class flows from a source network device to a destination network device. Each node in the forwarding graph is used to represent a particular network device on the path of the data packet as the data packet moves from a source node to the destination node. In other words, the forwarding graph shows a data structure of a network device's flow table, which includes a next-hop address that corresponds to the rule. For example, a forwarding graph that shows "S1→S2→S3→S4" indicates that according to the flow table, at node S1, packets falling within the range of this directly affected equivalence class are forwarded to node S2. Similarly, packets within node S2 are then forwarded to node S3 and subsequently forwarded to node S4.

The forwarding graph for each directly affected equivalence class is then forwarded onto module 320. Module 320 is configured to determine if a loop occurs in an directly affected equivalence class by deeply traversing a forwarding graph associated with the directly affected equivalence class. In particular, during traversal of a forwarding graph, the switches or nodes passed by packets in a directly affected equivalence class are recorded using a cache. If a node or a switch is passed twice by packets of the directly affected equivalence class, i.e. if the node or switch has been recorded in the cache, this means that a loop has occurred. Module 320 then marks the directly affected equivalence class accordingly and then proceeds to inform SDN controller 105 of the occurrence of the loop for the affected equivalence class or module 320 will then forward the equivalence class having the detected loop to adaptive resolution module 130.

Once forwarding graph module 125 has detected a directly affected equivalence class having a loop, module 125 may either collect all such directly affected equivalence classes with loops together before processing them or module 125 may process each directly affected equivalence class with loops individually. One skilled in the art will recognize that processing the equivalence classes with loops individually or as a group will not depart from the invention.

Module 125 will then forward an equivalence class having a loop onto adaptive resolution module 130 or onto SDN controller 105.

In an embodiment of the invention, once the equivalence class having the loop has been received by SDN controller 105, controller 105 will then notify a system administrator of the occurrence of the loop in the affected equivalence class. Although such a notification is

effective and efficient, this embodiment does not provide for ways for the loop conflict to be resolved in real time and as such, manual intervention is still required to be carried out by the system administrator.

In embodiments of the invention, the equivalence class having the loop conflict may be forward to adaptive resolution module 130 instead. Module 130 provides a method for loop conflict resolution. In particular, module 130 addresses loop conflicts by combining and delivering parts of the equivalence class that are classified as normal or by marking abnormal equivalence classes (equivalence classes having loop conflicts) as deny rules. In other words, when a conflicting equivalence class is located at the head or tail of a new rule, the normal equivalence classes are directly combined and delivered with the conflicting equivalence classes being omitted. When a conflicting equivalence class is in the middle of a new rule, the original rule (allow rule) is delivered first and the abnormal equivalence classes that cause the loop conflict are then marked and delivered as new deny rules. When the abnormal equivalence class generation rules marked with deny rules are matched during packet forwarding, these rules discard data packets to prevent network loops. In this way, only equivalence classes that do not cause loops will be delivered in new rules.

Packet Forwarding involving Packet Header Modifications

In another embodiment of the invention, a newly introduced rule may involve an action in its action list that causes a field of an affected packet's header to be modified, e.g. an action such as a Network Address Translation (NAT) command. For example, the new rule's action may cause the entirety or selected parts of a destination IP address in the packet's header to be altered.

With regard to Figure 3 and under the assumption that packet forwarding rule 301 now includes an action to modify packet headers, rule 301 that has been intercepted by LDS module 115 will similarly be provided to extended Trie tree module 120. As mentioned above, one skilled in the art will recognize that rule 301 may refer to the addition, modification or deletion of a rule. For brevity, although the subsequent parts of the description may refer to rule 301 as an addition of a new rule; one skilled in the art will recognize that the following parts in the description may be applied to a rule 301 that comprises a modification or deletion of the rule as well without departing from the invention. Further, one skilled in the art will recognize that the

action associated with rule 301 may involve any type of packet header modification action without departing from this invention.

Trie tree module 120 will then perform a check to determine if rule 301 includes an action that causes a packet's header to be modified. In this embodiment of the invention, as rule 301 includes such an action, Trie tree module 120 will then trigger module 306 (instead of module 305) to apply rule 301 to the network's existing Trie tree.

This means that module 306 will start by finding directly affected equivalence classes and this is done by first determining which existing rules will overlap with rule 301. This can be done by traversing the Trie tree, subtree by subtree, to determine rules (located at the leaves) that overlap with new rule 301. For each subtree, module 306 will limit the area to be traversed by selectively traversing branches of the subtree that fall within the range indicated by rule 301's filter.

Existing leaf nodes (rules) that overlap with rule 301 and rule 301 itself are then provided to module 311, together with their respective action lists, priority orders, associated switches and filters. Module 311 then uses this received information to generate directly affected equivalence classes containing rules that are affected by rule 301. For example, with reference to Figure 4, if it is assumed that leaf nodes A, B, C overlap with rule 301 (i.e. rules A-C are affected by rule 301), this means that rules in leaf nodes A-C will be used to form directly affected equivalence classes. It should be noted that a leaf node may store multiple rules whereby each rule has the same filter but each rule is associated with different switches. For example, if it is assumed that leaf node A overlaps with rule 301, leaf node A may have rules A1, A2, A3 (not shown) whereby each rule is associated with its own switch.

Module 311 is then configured to find the connected equivalence classes for each directly affected equivalence class. Module 311 achieves this by finding the header modification rules (if any) connecting a directly affected equivalence class and its connected classes. Module 311 generates a forward filter for each header modification rule taking effect in the directly affected equivalence class. The function of the forward filter is to define the equivalence class's packets with headers that have been modified by the action of the corresponding header modification rule. Module 311 then utilizes the Trie tree to find leaf nodes (rules) having filters that overlap with the forward filters. These overlapping leaf nodes (rules) are then gathered together to form one or more connected equivalence classes that are indirectly linked with the

original directly affected equivalence class. These equivalence classes are downstream classes. Similarly, module 311 finds upstream classes through generating a backward filter for each header modification rule connecting a directly affected equivalence class and upstream class(es). The function of the backward filter is to define the packets that will become the packets of the directly affected equivalence class after header modification by the action of the corresponding header modification rule. Module 311 then utilizes the Trie tree to find leaf nodes (rules) having filters that overlap with the backward filter. These overlapping leaf nodes (rules) are then gathered together to form one or more connected equivalence classes that are indirectly linked with the original directly affected equivalence class. Any downstream (resp. upstream) class's downstream (resp. upstream) classes could be connected equivalence classes. Therefore, module 311 also finds downstream (resp. upstream) classes for each downstream (resp. upstream) class with above method until all connected classes are found.

Module 311 then provides all this information to forwarding graph module 125. The following example is used to better illustrate the above concept.

First, it is assumed that Rule-A (55.1.0.0/16) and Rule-B (100.0.0.0/8) exists in the existing Trie tree. A new rule, Rule-N (55.0.0.0/8) is then to be added to the network whereby Rule-N contains an action to modify affected packet headers' destination IP address by amending the 8-Most Significant Bits in this field to "100". In other words, when Rule-N is applied to a packet's header, under the assumption that the packet header's original destination IP address is 1.1.1.1, this IP address will then be modified to 100.1.1.1 by Rule-N.

When Rule-N (55.0.0.0/8) is introduced, module 306 will start by determining the existing rules that will overlap with rule 301. In this example, module 306 will determine that rules falling within Rule-A's filter of 55.1.0.0/16 overlaps with Rule-N. Module 306 gathers all these rules and provides all this information to module 311. Module 311 then uses this received information to generate directly affected equivalence classes containing rules that are affected by Rule-N. In this example, two directly affected equivalence classes will be generated, i.e. a first directly affected equivalence class having a filter range between 55.0.0.0 - 55.0.255.255 and a second directly affected equivalence class having a filter range between 55.2.0.0 – 55.255.255.255. Rules falling within the overlap filter range of 55.1.0.0 – 55.1.255.255 are disregarded because these rules are affected by Rule-A as Rule-A is assumed to have a higher priority order than Rule-N.

Module 311 then attends to each of these directly affected equivalence classes by generating a forward or backward filter for each header modification rule connecting these directly affected equivalence classes to their downstream or upstream classes. In this example, Rule-N is the only header modification rule, and it connects directly affected equivalence classes to these downstream classes. The function of the forward filter is to store the new address of the destination field in the packet's header that has been modified by the action of Rule-N. It is useful to recap at this stage that Rule-N contains an action to modify affected packet headers' destination IP address by amending the 8-Most Significant Bits in this field to "100".

This means that for packets contained within the first directly affected equivalence class, i.e. packets having destination IP addresses between 55.0.0.0 - 55.0.255.255, these packets' destination fields will be modified by the action of Rule-N to become 100.0.0.0 - 100.0.255.255. As such, a forward filter containing the addresses 100.0.0.0 - 100.0.255.255 will be generated and will be associated with the first directly affected equivalence class.

Similarly, this means that for packets contained within the second directly affected equivalence class, i.e. packets having destination IP addresses between 55.2.0.0 - 55.255.255.255, these packets' destination fields will be modified by the action of Rule-N to become 100.2.0.0 - 100.255.255.255. As such, a forward filter containing the addresses 100.2.0.0 - 100.255.255.255 will be generated and will be associated with the second directly affected equivalence class.

Module 311 then utilizes the Trie tree to find leaf nodes (rules) having filters that overlap with the forward filters. These overlapping leaf nodes (rules) are then gathered together to form one or more connected equivalence classes that are indirectly linked with the original directly affected equivalence class. In this example, there exists a Rule-B having a filter of 100.0.0.0/8 that overlaps with the forward filters of the rules belonging to the first and second directly affected equivalence classes.

This means that a first connected equivalence class comprising Rule-B (100.0.0.0 - 100.0.255.255) will be connected with the first directly affected equivalence class (55.0.0.0 - 55.0.255.255) as rules contained within the first directly affected equivalence class have forward filters that overlap with this range "100.0.0.0 - 100.0.255.255" of Rule-B's filter. Similarly, a second connected equivalence class comprising Rule-B (100.2.0.0 - 100.255.255.255) will be

connected with the second directly affected equivalence class (55.2.0.0 – 55.255.255.255) as rules contained within the second directly affected equivalence class have forward filters that overlap with this other range “100.2.0.0 – 100.255.255.255” of Rule-B’s filter. Rule-B’s action will then subsequently be applied to the packets contained within the first and second connected equivalence classes.

Forwarding graph module 125 then utilizes module 315 to generate a forwarding graph for each affected equivalence class and its related connected equivalence class. To recap, a forwarding graph is essentially a graph showing the path of how data packets within the equivalence class flows from a source network device to a destination network device.

The forwarding graph for each affected equivalence class and its related connected equivalence class is then forwarded onto module 320. Module 320 is configured to determine if a loop occurs in an affected equivalence class and its related connected equivalence class by deeply traversing a forwarding graph associated with the affected equivalence class and its related connected equivalence class. In particular, during traversal of a forwarding graph associated with the affected equivalence class and its related connected equivalence class, the switches or nodes passed by packets in an equivalence class are recorded using a cache. If a node or a switch is passed twice by packets of the equivalence class, i.e. if the node or switch has been recorded in the cache, this means that a loop has occurred. Module 320 then marks the equivalence class accordingly and then proceeds to inform SDN controller 105 of the occurrence of the loop for the affected equivalence class or module 320 will then forward the equivalence class having the detected loop to adaptive resolution module 130.

If module 320 does not detect the occurrence of a loop in an affected equivalence class and its related connected equivalence class, the new header modification rule will be inserted in the leaf node associated with the rule’s filter and in the leaf node associated with the rule’s forward filter. (In case of modifying/deleting a header modification rule, the rule stored in these two leaf nodes will be modified/deleted.) If the new rule is not a header modification rule, the new rule will be inserted in the leaf node associated with the rule’s filter only. (in case of modifying/deleting a non-header-modification rule, the rule will be modified/deleted in this leaf node.)

Once forwarding graph module 125 has detected an equivalence class having a loop, module 125 may either collect all such equivalence classes together before processing them or

module 125 may process each equivalence class individually. One skilled in the art will recognize that processing the equivalence classes individually or as a group will not depart from the invention.

Module 125 will then forward an equivalence class having a loop onto adaptive resolution module 130 or onto SDN controller 105.

In an embodiment of the invention, once the equivalence class having the loop has been received by SDN controller 105, controller 105 will then notify a system administrator of the occurrence of the loop in the affected equivalence class. Although such a notification is effective and efficient, this embodiment does not provide for ways for the loop conflict to be resolved in real time and as such, manual intervention is still required to be carried out by the system administrator.

In embodiments of the invention, the equivalence class having the loop conflict may be forward to adaptive resolution module 130 instead. Module 130 provides a method for loop conflict resolution. In particular, resolve loop module 325 as contained within module 130, addresses loop conflicts by combining and delivering parts of the equivalence class that are classified as normal or by marking abnormal equivalence classes (equivalence classes having loop conflicts) as deny rules. In other words, when a conflicting equivalence class is located at the head or tail of a new rule, the normal equivalence classes are directly combined and delivered with the conflicting equivalence classes being omitted. When a conflicting equivalence class is in the middle of a new rule, the original rule (allow rule) is delivered first and the abnormal equivalence classes that cause the loop conflict are then marked and delivered as new deny rules. When the abnormal equivalence class generation rules marked with deny rules are matched during packet forwarding, these rules discard data packets to prevent network loops. In this way, only equivalence classes that do not cause loops will be delivered in new rules.

In embodiments of the invention, the following algorithm may be implemented in module 305, 306, 310 and/or module 311 to find the rules for building an affected equivalence class's connected equivalence classes. Input for this Algorithm 1 comprises a Trie tree having root *rp* which was pre-generated by the Trie tree module and the equivalence class's filter. The workings of algorithm 1 are generally self-explanatory.

Algorithm 1

```

algorithm find_rules_for_connected_EC(EC's filter ft, trie tree root rp):
{
    Find_related_backward_rules(ft, tree root rp);
    Find_related_forward_rules(ft, tree root rp);
}

Find_related_forward_rules(filter ft, tree root rp) {
    from the trie tree, find all leaf nodes with associated filters overlapping with filter ft, and
    get all rules in these nodes;
    among above rules, if there are header modification rules with filters overlapping with ft,
    for each rule X,
        create the forward filter of (X's filter and ft)'s joined filter y
        utilizing trie tree, find rules with filters overlapping with y
            use these rules to construct ECs with filters overlapping with y. These
            ECs are connected ECs i.e. indirectly affected ECs.
            construct these ECs immediately or save these rule and associated filter
            y for EC construction later.
    find y's related forward rules recursively by running find_related_forward_rules(y) }

Find_related_backward_rules(filter ft, tree root rp) {
    from the trie tree, find all leaf nodes with associated filters overlapping with filter ft, and
    get all rules in these nodes;
    among above rules, if there are header modification rules with forward filters
    overlapping with ft, for each rule X,
        create the backward filter of (X's forward filter and ft)'s joined filter y
        utilizing trie tree, find rules with filters overlapping with y
            use these rules to construct ECs with filters overlapping with y. These
            ECs are connected ECs i.e. indirectly affected ECs.
            construct these ECs immediately or save these rule and associated filter
            y for EC construction later.
    find y's related backward rules recursively by running find_related_backward_rules(y) }

```

Note: two filters' joined filter is the overlapping part of two filters. Packets matching the joined filter will match both original filters.

The affected equivalence classes; and if packet header modification has occurred, the connected equivalence classes associated with the affected equivalence classes are then provided to forwarding graph module 125. Forwarding graph module 125 then utilizes module 315 to generate a forwarding graph for each affected equivalence class and their connected equivalence classes if required.

As described previously, module 320 is configured to determine if a loop occurs in an affected equivalence class by deeply traversing a forwarding graph associated with the affected equivalence class and checking for switches that are passed or cached more than once. Module 320 then marks the equivalence class associated with the forwarding graph accordingly. Module 320 will then traverse a next forwarding graph to determine if a node or a switch has been passed twice by packets of the associated equivalence class to detect if a loop has occurred. Module 320 then marks the equivalence class associated with the next forwarding graph accordingly. If no loops were detected for either forwarding graphs, this means that no loop has occurred. Alternatively, if a loop was detected for either forwarding graph, module 320 is then configured to inform SDN controller 105 of the occurrence of the loop for the affected equivalence class or module 320 will then forward the equivalence class having the detected loop to adaptive resolution module 130. The equivalence class with the loop will then be handled as previously discussed.

Figure 5 illustrates a flow diagram showing how a rule adding/modifying/deleting operation that is introduced to a SDN network is intercepted by a LDS module in accordance with embodiments of the invention. It is useful to note that SDN application 505 refers to any type of SDN application that may reside in SDN controller 510, SDN controller 510 is a controller of a SDN network, LDS module 515 refers to a Loop Detection System (LDS) module that has been previously described, administrator 520 refers to a system administrator of the SDN network and SDN switch 525 refers to any network device residing the SDN's data plane that may be used to receive and forward data packets.

In operation, SDN application 505 will generate a request to add/modify/delete a rule. This rule is then transmitted at step 550 to SDN controller 510. Upon receiving the rule, SDN controller 510 will then attempt to transmit the rule adding/modifying/deleting request to the relevant SDN switch however; as LDS module 515 has been configured to intercept such rule operations, LDS module 515 will intercept the transmission of the new rule operation at step 555. It should be noted that at this stage, SDN controller 510 is unaware whether SDN application 505 is an authorized application or a malicious application as SDN controller 510 has not been configured to perform any validation checks on the veracity of SDN application 505.

Once LDS module 515 has intercepted the new rule, LDS module 515 will utilize the system's existing Trie tree to determine if the new rule operation will cause a loop to occur and this takes place at step 560. Affected equivalence classes are then obtained as previously discussed.

If LDS module 515 determines that a loop has occurred in an affected equivalence class, steps 565, 570 and 572 will take place. In particular, at step 565, LDS module 515 will send a warning to administrator 520. Administrator 520 will then assess the conflict that occurred and attend to the cause of the conflict at step 570. Alternatively, administrator 520 may decide that the conflict may not be resolved and administrator 520 may then inform SDN controller 510 to reject the new rule introduced by SDN application 505.

Conversely, if LDS module 515 determines that no loop has occurred in the affected equivalence class, LDS module will then allow the new rule to be forwarded onto SDN switch 525 and this takes place at step 580. Once the new rule has been implemented at SDN switch 525, switch 525 then notifies SDN controller 510 of the implementation of the new rule at step 582. SDN controller 510 then in turns informs SDN application 505 of the implementation of the new rule at step 584.

Figure 6 illustrates a flow diagram showing how a rule operation (i.e. an adding, modifying or deleting rule operation) that is introduced to a SDN network is intercepted by a LDS module. In this embodiment of the invention, if a loop is detected in an affected equivalence class, the detected loop is handled by an adaptive resolution module. It is useful to note that SDN application 605, SDN controller 610, LDS module 615, and SDN switch 625 refer to modules and components as described above in relation to Figure 5. As for adaptive resolution module 620, this module is configured to provide a method for loop conflict resolution

by reconstructing the new rule as previously described. In short, module 620 addresses loop conflicts by combining and delivering parts of the equivalence class that are classified as normal or by marking abnormal equivalence classes (equivalence classes having loop conflicts) as deny rules.

In operation, SDN application 605 will generate a new flow table rule operation request or a new rule operation request. This request is then transmitted at step 650 to SDN controller 610. Upon receiving the request, SDN controller 610 will then attempt to transmit the rule operation request to the relevant SDN switch however; as LDS module 615 has been configured to intercept such new rules, LDS module 615 will intercept the transmission of the new rule operation request at step 655. Once LDS module 615 has intercepted the new rule operation request, LDS module 615 will utilize the system's existing Trie tree to determine if the rule operation will cause a loop to occur and this takes place at step 660. Affected equivalence classes are then obtained as previously discussed.

If LDS module 615 determines that a loop has occurred in an affected equivalence class, steps 665, 670 and 672 will take place. In particular, at step 665, LDS module 615 will send the affected equivalence class with the loop conflict and the new rule to adaptive resolution module 620. At step 670, adaptive resolution module 620 receives the rule operation and the associated equivalence class and then attempts to combine or segregate the equivalence class as necessary. If the loop conflict occurs to a head or tail part of the operated rule's filter, the rule operation is only delivered after a domain where the conflicting equivalence class is located has been removed. If the conflicting equivalence class is in the middle of the rule's filter, the rule operation is directly delivered and new rules that are used to eliminate the conflict at delivered at the same time. The delivering of the relevant new rules takes place at step 672.

Conversely, if LDS module 615 determines that no loop has occurred in the affected equivalence class, LDS module will then allow the rule operation request to be forwarded onto SDN switch 625 and this takes place at step 680. Once the rule operation has been implemented at SDN switch 625, switch 625 then notifies SDN controller 610 of the implementation of the rule operation at step 682. SDN controller 610 then in turns informs SDN application 605 of the implementation of the rule operation at step 684.

Figure 7 illustrates a flow diagram showing how a new rule operation that is introduced to a SDN network is intercepted by a LDS module. In this embodiment of the invention, the

newly introduced rule operation includes a command to modify a domain of a packet header. It should be noted that SDN application 705, SDN controller 710, LDS module 715, adaptive resolution module 720 and SDN switch 725 refer to modules and components as described above in relation to Figures 5 and 6.

In operation, SDN application 705 will generate a new flow table rule or a new rule. This rule is then transmitted at step 750 to SDN controller 710. Upon receiving the rule, SDN controller 710 will then attempt to transmit the rule to the relevant SDN switch. As mentioned above, LDS module 715 will intercept such new rules at step 755. Once LDS module 715 has intercepted the new rule, LDS module 715 will utilize the system's existing Trie tree to determine if the new rule will cause a loop to occur and this takes place at step 760. Affected equivalence classes are then obtained as previously discussed. At this step, LDS module 715 will also determine whether the newly introduced rule includes a command that causes a packet header's domain or field to be modified. If LDS module 715 determines that the newly introduced rule does not include a packet header modification command, LDS module 715 will then carry out the steps as described above in relation to Figure 6. Alternatively, if LDS module 715 determines that the newly introduced rule includes a packet header modification command; LDS module 715 will generate an affected equivalence class at step 762, and its associated connected affected equivalence class at step 764. A forwarding graph is then generated for the pair of indirectly connected equivalence classes at step 766. LDS module 715 then uses the forwarding graph to determine if a loop has occurred.

If LDS module 715 determines that a loop has occurred, steps 768, 770 and 772 will take place. In particular, at step 768, LDS module 715 will send the affected equivalence class with the loop conflict and the new rule to adaptive resolution module 720. At step 770, adaptive resolution module 720 receives the rule and the associated equivalence class and then attempts to combine or segregate the equivalence class as necessary. If the loop conflict occurs to a head or tail part of the new rule, the new rule is only delivered after a domain where the conflicting equivalence class is located has been removed. If the conflicting equivalence class is in the middle of the new rule, the new rule is directly delivered and the new rules that are used to eliminate the conflict are delivered at the same time. The delivering of the relevant new rules takes place at step 772.

Conversely, if LDS module 715 determines that no loop has occurred in the affected equivalence class, LDS module will then allow the new rule to be forwarded onto SDN switch

725 and this takes place at step 780. Once the new rule has been implemented at SDN switch 725, switch 725 then notifies SDN controller 710 of the implementation of the new rule at step 782. SDN controller 710 then in turns informs SDN application 705 of the implementation of the new rule at step 784.

One skilled in the art will recognize that an administrator module may be utilized in place of adaptive resolution module 720. The function of the administrator module would be similar to that of administrator module 520 as described above in relation to Figure 5 whereby instead of adaptively resolving the detected loop conflict, an administrator is instead notified of the conflict loop occurring.

The above is a description of embodiments of a device and process in accordance with the present invention as set forth in the following claims. It is envisioned that others may and will design alternatives that fall within the scope of the following claims.

CLAIMS:

1. A loop detecting module provided within a verification system for detecting abnormal routing loops in a computer network, the loop detecting module being positioned between network devices of the computer network and a controller of the computer network, the loop detecting module comprising:
 - a processor; and
 - a non-transitory media readable by the processor, the non-transitory media storing instructions that when executed by the processor, cause the processor to:
 - intercept a packet rule modification initiated by the controller before the packet rule modification is received by the network devices;
 - identify equivalence classes affected by the intercepted packet rule modification;
 - determine connected equivalence classes associated with each affected equivalence class; and
 - verify routing of packets affected by the intercepted packet rule modification using the affected equivalence classes and the connected equivalence classes associated with each of the affected equivalence classes.
2. The loop detecting module according to claim 1 wherein the instructions for directing the processor to identify the equivalence classes affected by the intercepted packet rule modification comprises:
 - instructions for directing the processor to
 - group packets sent through the computer network into a plurality equivalence classes whereby each equivalence class comprises a set of packets that are affected by a set of common packet rules; and
 - determine an equivalence class affected by the intercepted packet rule modification.
3. The loop detecting module according to claim 1 wherein the instructions for directing the processor to identify the equivalence classes affected by the intercepted packet rule modification comprises:
 - instructions for directing the processor to:
 - apply the intercepted packet rule modification to a multidimensional prefix tree (Trie tree) generated using existing packet rules, whereby the Trie tree comprises a plurality of sub-trees, whereby each sub-tree represents a packet header domain and each leaf of the Trie tree stores an associated set of packet rules;

identify existing packet rules affected by the intercepted packet rule modification by traversing the Trie tree to determine leaves having existing packet rules that overlap with the intercepted packet rule modification; and

generate the affected equivalence classes based on the overlapping existing packet rules.

4. The loop detecting module according to any one of claims 1 to 3 wherein the instructions for directing the processor to determine the connected equivalence classes associated with each affected equivalence class comprises:

instructions for directing the processor to:

identify affected equivalence classes having a packet rule or a packet that involves a header modification instruction; and

generate connected equivalence classes for each of the identified equivalence classes.

5. The loop detecting module according to claim 4 wherein the instructions for directing the processor to generate the connected equivalence classes for each of the identified equivalence classes comprises:

instructions for directing the processor to:

determine existing packet rules that overlap with packets of the identified equivalence classes before or after header modification of the packets of the identified equivalence classes utilizing the multidimensional prefix tree (Trie tree);

construct connected equivalence classes from overlapping rules.

6. The loop detecting module according to claim 5 wherein the instructions for directing the processor to construct connected equivalence classes from overlapping rules comprises:

instructions for directing the processor to:

for each of the identified equivalence classes,

generate a forward filter whereby the forward filter comprises packets after the header modification;

determine existing packet rules that overlap with the forward filter by applying the forward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the forward filter; and

group packets affected by the existing packet rules that overlap with the forward filter to form connected equivalence classes.

7. The loop detecting module according to claim 5 wherein the instructions for directing the processor to construct connected equivalence classes from overlapping rules comprises:
 - instructions for directing the processor to:
 - for each of the identified equivalence classes,
 - generate a first backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction;
 - determine existing packet rules that overlap with the first backward filter by applying the first backward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first backward filter; and
 - group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes.

8. The loop detecting module according to claim 5 wherein the instructions for directing the processor to construct connected equivalence classes from overlapping rules comprises:
 - instructions for directing the processor to:
 - for each of the identified equivalence classes,
 - generate a first backward filter and a second backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction and the second backward filter comprises a second set of original packets before the second set of original packets are modified into packets of the identified equivalence class by the header modification instruction;
 - determine existing packet rules that overlap with the first and second backward filters by applying the first and second backward filters to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first and second backward filters; and
 - group packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes associated with the first backward filter and group packets affected by the existing packet rules that overlap with the second backward filter to form connected equivalence classes associated with the second backward filter.

9. The loop detecting module according to any one of claims 1 to 8 wherein the instructions for directing the processor to verify the routing of packets affected by the intercepted packet rule modification comprises:
 - instructions for directing the processor to:
 - construct forwarding graphs for each of the affected equivalence classes using connected equivalence classes associated with each of the affected equivalence classes whereby each of

the forwarding graphs illustrate the routing of packets within each of the affected equivalence classes and the associated connected equivalence classes; and

determine occurrence of routing loops based on the forwarding graphs.

10. The loop detecting module according to any one of claims 1 to 9 further comprising:
 - instructions for directing the processor to:
 - apply the packet rule modification into a leaf of the Trie tree associated with the packet rule modification when the routing of packets affected by the intercepted packet rule modification are verified.
11. A method for detecting abnormal routing loops in a computer network using a loop detecting module positioned between network devices of the computer network and a controller of the computer network, the method comprising:
 - intercepting a packet rule modification initiated by the controller before the packet rule modification is received by the network devices;
 - identifying equivalence classes affected by the intercepted packet rule modification;
 - determining connected equivalence classes associated with each affected equivalence class;
 - and
 - verifying routing of packets affected by the intercepted packet rule modification using the affected equivalence classes and the connected equivalence classes associated with each of the affected equivalence classes.
12. The method according to claim 11 wherein the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises:
 - grouping packets sent through the computer network into a plurality equivalence classes whereby each equivalence class comprises a set of packets that are affected by a set of common packet rules; and
 - determining an equivalence class affected by the intercepted packet rule modification.
13. The method according to claim 11 wherein the step of identifying the equivalence classes affected by the intercepted packet rule modification comprises:
 - applying the intercepted packet rule modification to a multidimensional prefix tree (Trie tree) generated using existing packet rules, whereby the Trie tree comprises a plurality of sub-trees, whereby each sub-tree represents a packet header domain and each leaf of the Trie tree stores an associated set of packet rules;

identifying existing packet rules affected by the intercepted packet rule modification by traversing the Trie tree to determine leaves having existing packet rules that overlap with the intercepted packet rule modification; and

generating the affected equivalence classes based on the overlapping existing packet rules.

14. The method according to any one of claims 11 to 13 wherein the step of determining the connected equivalence classes associated with each affected equivalence class comprises:

identifying affected equivalence classes having a packet rule or a packet that involves a header modification instruction; and

generating connected equivalence classes for each of the identified equivalence classes.

15. The method according to claim 14 wherein the step of generating the connected equivalence classes for each of the identified equivalence classes comprises:

determining existing packet rules that overlap with packets of the identified equivalence classes before or after header modification of the packets of the identified equivalence classes utilizing the multidimensional prefix tree (Trie tree); and

constructing connected equivalence classes from overlapping rules.

16. The method according to claim 15 wherein the step of constructing connected equivalence classes from overlapping rules comprises:

for each of the identified equivalence classes,

generating a forward filter whereby the forward filter comprises packets after the header modification;

determining existing packet rules that overlap with the forward filter by applying the forward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the forward filter; and

grouping packets affected by the existing packet rules that overlap with the forward filter to form connected equivalence classes.

17. The method according to claim 15 wherein the step constructing connected equivalence classes from overlapping rules comprises:

for each of the identified equivalence classes,

generating a first backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction;

determining existing packet rules that overlap with the first backward filter by applying the first backward filter to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first backward filter; and

grouping packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes.

18. The method according to claim 15 wherein the step of constructing connected equivalence classes from overlapping rules comprises:

for each of the identified equivalence classes,

generating a first backward filter and a second backward filter whereby the first backward filter comprises a first set of original packets before the first set of original packets are modified into packets of the identified equivalence class by the header modification instruction and the second backward filter comprises a second set of original packets before the second set of original packets are modified into packets of the identified equivalence class by the header modification instruction;

determining existing packet rules that overlap with the first and second backward filters by applying the first and second backward filters to the multidimensional prefix tree (Trie tree) to identify the existing packet rules that overlap with the first and second backward filters; and

grouping packets affected by the existing packet rules that overlap with the first backward filter to form connected equivalence classes associated with the first backward filter and group packets affected by the existing packet rules that overlap with the second backward filter to form connected equivalence classes associated with the second backward filter.

19. The method according to any one of claims 11 to 18 wherein the step of verifying the routing of packets affected by the intercepted packet rule modification comprises:

constructing forwarding graphs for each of the affected equivalence classes using connected equivalence classes associated with each of the affected equivalence classes whereby each of the forwarding graphs illustrate the routing of packets within each of the affected equivalence classes and the associated connected equivalence classes; and

determining occurrence of routing loops based on the forwarding graphs.

20. The method according to any one of claims 11 to 19 wherein the method further comprises:

applying the packet rule modification into a leaf of the Trie tree associated with the packet rule modification when the routing of packets affected by the intercepted packet rule modification are verified.