DAIM: a Mechanism to Distribute Control Functions within OpenFlow Switches

Ameen Banjar[a], Pakawat Pupatwibuli[b] and Robin Braun[c]
Centre for Real Time Information Networks, School of Computing and Communications, University of Technology, Sydney, Australia
[a]Ameen.r.Banjar@student.uts.edu.au, [b]Pakawat.Pupatwibul@student.uts.edu.au, [c]Robin.Braun@uts.edu.au

Abstract—**Telecommunication networks need to support a wide range of services and functionalities with capability of autonomy, scalability and adaptability for managing applications to meet business needs. Networking devices are increasing in complexity among various services and platforms, from different vendors. The network complexity is required experts' operators. This paper explores an introduction to networks programmability, by distributing independent computing environment, which would be demonstrated through a structured system named DAIM model (Distributed Active information Model). In addition it seeks to enhance current SDN (Software-Defined Networking) approach which has some scalability issues. The DAIM model can provide richness of nature-inspired adaptation algorithms on a complex distributed computing environment. The DAIM model uses a group of standard switches, databases, and corresponding between them by using DAIM agents. These agents are imposed by a set of network applications, which is integrated with a DAIM model databases. DAIM model also considers challenges of autonomic functionalities, where each network's device can make its own decisions on the basis of collected information by the DAIM agents. The DAIM model is expected to satisfy the requirement of autonomic functionalities. Moreover, this paper discussed the processing of packets forwarding within DAIM model as well as the risk scenarios of the DAIM model.**

Index Terms—**Distributed Networks, Information Model, Software-Defined Networking (SDN), DAIM, Self-X properties, Artificial Intelligence.**

I. Introduction

In the last few years network technologies have been increasing significantly in performance, complexity, functionality, driven by the needs of the modern world. However, existing network infrastructure lacks adaptability, and demands device centric centralized management paradigms. Networks have become massive and intractable due to complexity, leading to challenges of scalability.

New network management paradigms may take several years to develop, and much longer to become widely spread. In addition, there is a gap between market requirements and network capabilities from vendor's side, where network operators need to design the network according to the requirements of users, which limits their abilities. Moreover, vendors lack standards and open interfaces [1].

Hence, there is a need for open and flexible architectures to implement autonomic management functionality [2], which has been considered as a solution to ameliorate the complexity of network management. This has given rise to a new network paradigm called Software-Defined Networking (SDN). It is aimed at reducing the complexity of management (see Fig. 1) [1]. The main idea of SDN is to separate the functionality of data path from control path. The data path remains in a switch whereas the high-level routing decisions are separated into a device called a controller, basically a routing server. The first industry developed standard for SDN is OpenFlow-based [1]. OpenFlow has a protocol, which is used between the switches and the controller to communicate and exchange messages such as get-stats, packets-receive, and packets-sent-out [3]. So, companies get the network programming ability to control the network with high scalability and flexibility, which can adapt easily according to ever changing circumstances. According to Fig. 1, the SDN structure has layers including an application layer, control layer, and infrastructure layer. In more detail the control layer has APIs (Application Programming Interfaces) ability, so it is possible to implement autonomic functionality such as self-protection and self-optimization [4].

One of the means to implement autonomic functionality within SDN is a new nature inspired active information model. It allows local decision-making in each network device which creates a complex distributed network environment. This paper proposes that a new information model called the Distributed Active Information Model (DAIM) can be implemented in OpenFlow-based SDN to meet the requirements of the autonomic components of the distributed network, such as self-management [5]. The benefits of implementing the proposed approach, include control devices and the rapid configuration of the entire network autonomically, without reconfiguring each individual device. In addition, the DAIM model can manage complex systems in any distributed network, which makes it possible to be autonomous, adaptable, and scalable.

DAIM is a sustainable information model, which collects, maintains, updates and synchronizes all the related information. Moreover, the decision making ability within each device locally, on the basis of collected information,

This work was supported by the Centre for Real-Time Information Networks (CRIN) in the Faculty of Engineering and IT at the University of Technology, Sydney (UTS). Corresponding author: Ameen Banjar, Ameen.r.Banjar@student.uts.edu.au

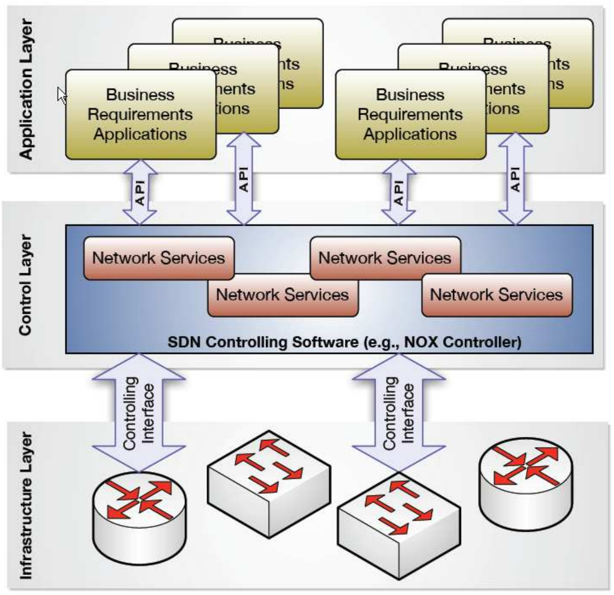

Figure 1: Software-Defined Networking Architecture

allows it to autonomically adapt according to the ever-changing circumstances [6]. The DAIM model structure is proposed with the hope that it addresses the limitation of previous network protocols such as Simple Network Management Protocol (SNMP) [7], Common Information Model (CIM) [8] and Policy-Based Network Management.

Ultimately, the proposed DAIM model will address the limitations of current approaches and future distributed network systems, creating an autonomic computing management strategy. The DAIM model approach will also satisfy the requirements of autonomic functionality for distributed network components like self-learning, self-adaptation and self-CHOP (configuration, healing, optimization and security). Each component can be adaptable according to any changed conditions of the dynamic environment without human intervention.

The remainder of the paper is organized as follows. In Section II we explain the limitations of current OpenFlow networks. Section III has details of related works to address those limitations. Our proposed new information model, which supports autonomic management for distributed systems is introduced in Section IV. Followed by Section V, which is the processing of the packets within the DAIM model. Section VI presents risk scenarios of DAIM model. Finally, we conclude the paper by summarizing the main contribution and future work in Section VII.

II. OpenFlow-Based SDN scalability issues

There are number of tests to prove that the current OpenFlow is lacking in scalability because of using a centralized controller. This part presents scalability issues of the current SDN approach. Also it covers related works that are relevant for addressing the SDN scalability issue.

One of the major limitations of a centralized OpenFlow controller is lack of scalability. The fundamental feature of the NOX controller is that it is responsible for establishing every forwarding rule in the network. As the size of production networks deploying OpenFlow increases, so will the number of flows that need to be processed [9]. If the NOX does not have the capacity to handle all these flow setups, it can present a scalability bottleneck. For example, an enterprise data center's networks may have 100 edge switches. The NOX controller could expect to see around 10 million flow requests per second [10], [11]. This could create significant challenges for deploying centralized OpenFlow controllers in large-scale data centers [10]. Another drawback of the NOX is that each flow request is processed individually, and all packets created accordingly are forward individually. In addition, sending out messages individually takes about 80% of the flow request processing time. This can cause an overhead of multiple socket write operations to forward each packet to the same destination individually instead of a single batched process. Moreover, the NOX does not provide sufficient flexibility to achieve scalability for application developers, nor adequately address reliability as the control platform must handle equipment and other failures gracefully [12].

Reference [13] explains that relying heavily on only one centralized controller for the whole network may not be feasible for a number of reasons. Firstly, the amount of control traffic destined for the controller increases according to number of switches. Secondly, despite where the controller is placed, if the networks have large diameter, some switches will face long flow setup latencies. Thirdly, since the network is bounded by the processing capacity of the controller, flow setup times can grow rapidly as demand grows in terms of network size and complexity [13]. Therefore, improving the performance of the NOX controller to keep up with the rising demand becomes a significant challenge. On the other hand, [14] highlight the difficulties of writing programs for the OpenFlow-based NOX platform as follows:

• Interactions between concurrent modules – Networks often process multiple tasks such as routing, monitoring, and access control. These functions cannot be processed independently unless they perform on non-overlapping portions of the traffic, since a rule (un)installed by one module could undermine the proper functioning of other modules

• Low level interface to switch hardware – OpenFlow maintains a low-level interface to the switches. Applications must establish rules that match on bits in the packet header. Because rules can have wild cards, a packet may match different overlapping rules with multiple priorities. This may translate high-level policy into multiple low-level rules.

• Two-tiered programming model – Controllers only receives packets that switches do not know how to process. This can limit the controller's visibility into the underlying traffic. Essentially, the execution of application is split between the controller and the switches. Applications must avoid installing rules that hide vital information from the controller.

Software Defined Networking (SDN) is simple to manage and it is flexible. That depends on abstracting control and management functions as network applications. In addition, the controller has a wide view of the entire network [15]. Moreover, the SDN architecture has that control layer and forwarding layer, this architecture relies on central point, which control the forwarding layer. So, either interface has scalability bottleneck in the controller and in the switches [16]. For example bottleneck within switch can be in forwarding table memory. Thus, the current architecture lacks scalability when facing extended networks which include all the networks used in data transfer and information from distant places and in a wide geographical area (several kilometers to thousands of kilometers). Any such links between computer devices in places far away from each other, and which can connect branches of the institution within or outside the country with each other and allows users to exchange information and e-mail.

Scalability means that the network can increase the number of nodes and the length of links very widely, with the performance of the network is not affected. So, ensure network scalability necessary to use additional communications equipment and specially structured network. For example, good scalability has multi-segment networks, built using switches and routers and has a hierarchical structure of relationships. Such a network may include several thousand computers while providing each user the right network service quality. Thus, this research notes that by using the DAIM model, the scalability of the system could be solved by using a distributed environment as the control platform.

III. Related Works

Related work shows that, three approaches to scale central controller as following:

A. Using optimization techniques

• **Maestro** is a control system for centralized network, which has been developed to solve the limitations of OpenFlow control plane. Maestro has a central controller for a flow-based routing network, with an increasing of flow processing; it requires being extremely scalable, Maestro can achieve that and can coordinate between centralized controls and distributed routing protocols. Maestro approach works as hybrid control plane, which is robust than the centralized control plane. It uses parallelism technique to alleviate packet processing and solve the bottleneck on the controller of OpenFlow structure by using applications, such as “routing” or “learning switch” [10].

Maestro has programmable environment with a high-level language, which can deal with distribution and concurrency without involving the developer [17]. Maestro has a user interface to control hardware and also includes an analysis tools. It can auto-detects the attached hardware. Moreover, Maestro can make local decisions without involving master, and synchronize certain actions by using the master [18]. Developer is not involved which Maestro insert locks. As a result Maestro accepts as input high-level program actions, for protected structure [10]. However, Maestro still relies on only a single controller and send batching messages to the switches not in run time configuration. These challenges can significantly affect the packet forwarding process.

B. Devolving some control functions back to the switches

• **DevoFlow** (Devolved OpenFlow) modifies the Open Flow model to redistribute as many decisions as possible to the switches, in ways to enable simple and cost-effective hardware implementations [19]. DevoFlow can solve the bottleneck of the OpenFlow switch within high-performance networks. Where an unknown packet is usually forwarded to the controller. Assuming there are several thousand flows being forwarded per-second. DevoFlow proposes to tackle the problem by addressing short-lived (mice) and long-lived (elephant) flows separately. Switches only inform the controller about specific flows, which required more security or any other policies [20].

DevoFlow is forcing to use wild-carded rules of OpenFlow, so it can reduce interactions between switch-controller. DevoFlow mechanisms, allows the switches to make local decisions for routing when these do not actually require per-flow checking by the controller [19]. DevoFlow involves the OpenFlow controller, but puts too much load on the control plane so forcing wild-cards flow-match is reducing that load, however the controller disable to check some events [20].

• **DIFANE** is a distributed flow-based architecture built on OpenFlow switches. It aims to resolve the centralized issue by distributing the functionalities across “authority switches” and calculating matching rules at the switches themselves. The division of labor is changed by DIFANE between the centralized management system and the switches, by pulling some rule processing functions back to the switches, to achieve better scalability. DIFANE can solve the bottleneck of the current OpenFlow centralized controller by distributed some of the controller functionalities across authority switches. This authority can handle unknown packet received from other switches instead of sending packet to the controller [21].

DIFANE downgrades simpler tasks from the controller by translating high-level policies to low-level rules, and distributed the rules and processing all the packets in the switches. DIFANE reduces the memory usage for rules at the switches. So, DIFANE builds a distributed rule directory service among the switches by partitioning the rules between switches. DIFANE can be easily implemented with small software medications to commercial OpenFlow switches [22].

DIFANE architecture composed of a controller and authorized switches. The controller provides rules and

Wild card Rules	Description
Cache rules	Within the switches, the data traffic stays in the cache, where ingress switches are responsible for processing. Authority switches install the cache rules in the network.
Authority rules	The controller can update and install the authority rules in authority switches which is able to store authority rules.
Partition rules	Partition rules are installed in each switch by the controller, so packet could match one rule overall and stay in the forwarding plane.

Table I: DIFANE wild card rules

install them in the authority switches. The authority switches are receiving the packet and forward it according to the rules or encapsulate it and send it to other authority switches [23].

Ref. [21] indicate that DIFANE has three sets of wild-card rules, which can keep packet processing within forwarding plane instead to send to the controller. Wild-card rules include cache rules, authority rules, and partition rules as the following table I:

However, devolving control functions back to the switches is not easy to deploy a set of relatively rules and configurations installed in OpenFlow switches in terms of security perspectives.

C. Designing a distributed control platform

• **HyperFlow**

HyperFlow aims to have multiple controllers to manage the entire network each controller is respectable of its portion of the network [13]. HyperFlow is an application implemented on top of the NOX controller, where the implementation is changed operations, and allows reuse of existing NOX applications with minor modifications [24].

In addition, each switch makes a local decision (relies on its flow table and its controller) using the HyperFlow to passively synchronize state upon whole network of OpenFlow controllers. This can provide a local serve by controller to all packets flows, and thus significantly reduces the response time of control plane for data plane requests. Each controller in HyperFlow network has the ability to control the whole network because it has a coherent wide view of the network. If any controller fails, all affected switches have to reconfigure by themselves to join other nearest controller [13]. Thus, HyperFlow works as a centralized paradigm with centralized benefits, however it is scalable physically distributed network [25].

• **Onix**

Onix approach is to have reliability by distributed controller [12]. It provides programming API to build the network applications. Onix contents are distributed to the applications to ease distributed coordination. Onix deal with register per-packet instead of dealing with per-packet events for less frequent. Onix has been made for scalability purposes where controller can not forward packets faster than switch. Onix provides a Network Information Base (NIB), which gives access to several state synchronization frameworks with different consistency and availability requirements.

Onix has programming ability to access the network by providing control logic. In addition, Onix instance communicating with other instances via cluster, which is responsible for distributing the network state. Thus, Onix can scale large networks and provide flexibility for production deployments.

However, distributing the control platform in HyperFlow and Onix are not reliable in large data centres as they are not fully distributed.

IV. Distributed Active Information Model Theory

This section will introduce the designed architecture of the candidate system (the DAIM model). The objectives and uniqueness of the proposed DAIM model will also be discussed. It starts by describing the design goals for the DAIM model, the architectural overview of the DAIM model including databases using augmented OpenFlow protocol (DAIM protocol), within the DAIM model, and DAIM agents which is an important component of DAIM model.

A. Objectives of designing DAIM

The goals of designing DAIM is to address the research challenges such as managing the complexity of distributed electronic environment, and construct a reactive interpreter network with self-X autonomic functionalities for business needs. So, the design requirements to build the DAIM model according to [6], [26], [2] will be as following:

• Compatibility: Considering the complexity and the future grows of the networks, where there are varieties of network devices and business needs. So, DAIM will use OpenFlow-Based SDN environment as a programmable network to meet different varieties of needs. DAIM can abstract the network management model and services as network applications.

• Model simplicity: DAIM allows switches to make decisions locally. That ability is called autonomic network management. This means that the distributed self-adaptation strategies can maintain the system in the face of changing requirements and unexpected threats to provide for the defined requirement. So, Operators and programmers are no long required to handle any changes of the requirement either actively or re-actively. Network management model and services will be abstracted as network applications.

• End hosts modification: The DAIM model does not require software or hardware changing of the end hosts, where DAIM mainly focuses in forwarding packets.

• Security: DAIM is supporting security by using network security protocols. For example, the messages between System Requirement Database and OpenFlow switches, are encrypted by using Transport Layer Security (TLS).

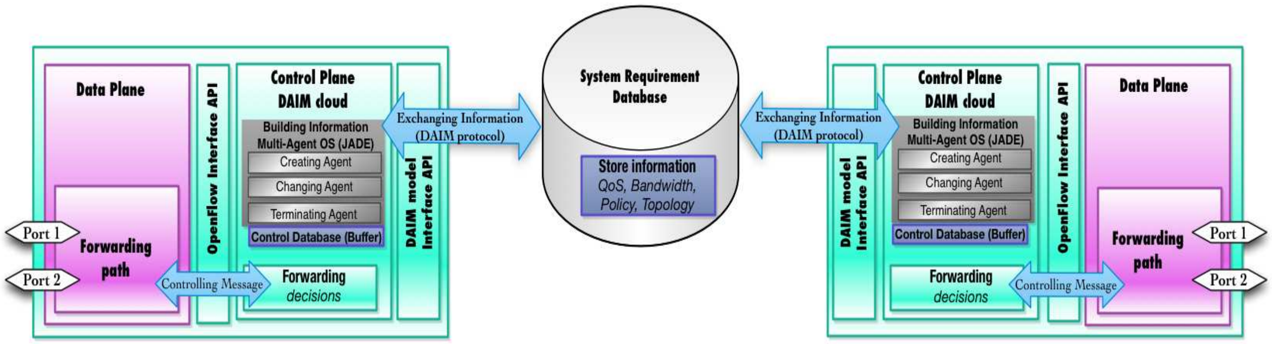

Figure 2: DAIM model architecture as an intelligent computational environment

B. *Uniqueness of DAIM model*

The DAIM model derives its design principles from all previous approaches. That is, dividing the control functions, and implementing them on data forwarding layer, as a distributed control plane so the switches will have more control functions. Furthermore, the DAIM model tries to solve the current issues of a centralized network control plane and difficulties to manage the network by having autonomic behaviors for network management based on DAIM agents [6], [26], [27].

There are five unique aspects of the DAIM model. Firstly, the DAIM model is a programming framework for creating a distributed control functions within the SDN environment. The DAIM model can be applied in a flow-based routing network such as OpenFlow. Secondly, DAIM model provides clear and direct control over interactions with the system requirement database, and over network state synchronization by using DAIM agents to gather information and set instruction. Thirdly, the DAIM model could solve the scalability issue of the centralization, by distributed control functions within OpenFlow switches. Fourthly, using the DAIM model in distributed network environment can solve the robustness and responsiveness issues of the current centralized paradigm. The adaptation algorithms can adapt the distributed nodes by synchronizing the state from the system requirement database. Finally, the DAIM model is not similar to the cloud computing model, where cloud computing generally has many separated computing entities, presented as a one computational infrastructure. While, the DAIM model has many of network entities, which are distributed and working independently, where each represented as independent computing environment.

C. *DAIM model architecture*

The DAIM model architecture is composed of a logically centralized System Requirement Database, Discover Routes Database, distributed Control Database, and DAIM agents residing in each switch as an independent computational environment. These components use augmented OpenFlow protocol called DAIM protocol for corresponding messages between them. Intelligent DAIM agents are implemented in a Java Virtual Machine environment (JVM). They interact with the databases, neighbouring switches, and exchange information with other agents to compute their own local decisions according to the business needs defined in those databases as shown in Fig. 2.

The DAIM model is implemented within each OpenFlow switch using a Multi-agent operating system, which is supported by DAIM agents as a field of distributed active artificial intelligent (AI) to enable the self-management. The basic information unit of the DAIM model is the DAIM agents and each of them include [6]:

• **Attributes:** specific variables that represent characteristics of the flow entries such as header fields, counters and actions.

• **Method behaviors:** actions that provide the autonomic functionalities such as self-awareness instantly (temperature, humidity), and self-configuration (switch down).

• **Algorithms:** algorithms for fulfillment a network task, can be embedded into DAIM agents, such as informing DAIM model if any circumstances change within the network and synchronize information between databases.

• **Messaging:** messages that can be created by DAIM model as a response of requests to get information (track host location, track topology changing and shortest route). The Control Databases are connected together and use DAIM protocol for corresponding. At the same time each switch is connected to a System Requirement Database (SRD) and Discovered Route Database for optimizing the performance of the network.

To meet our approach of managing distributed environment autonomically, we need to analyze network operating system components. These components are supporting the DAIM agents to make local decisions in terms of forwarding, maintaining, and adapting to the unexpected changes. Components could be distributed within OpenFlow switches and databases.

The DAIM database schema holds the network information such as host identifier, business requirements, topology discovery, QoS, bandwidth, users, and global view of the entire network. The network business requirements are stored in the System Requirement and Discover Route Databases' tables, which must have well defined requirements. Records in other tables are significant only when they can be reached directly or indirectly from

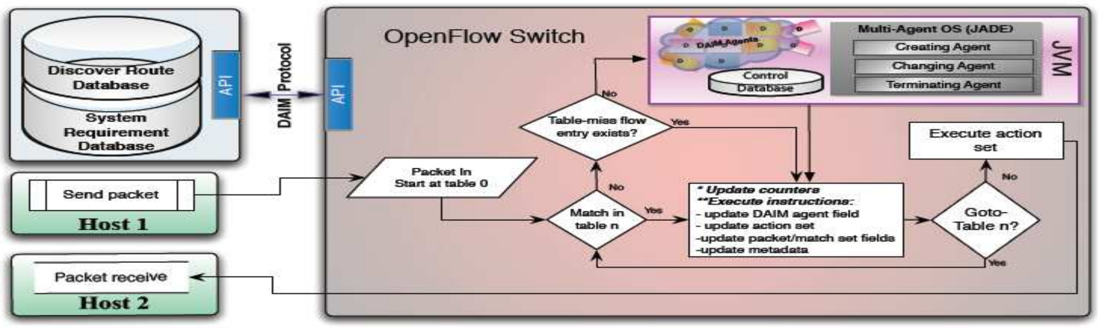

Figure 3: Flow chart detailing Packet processing within DAIM model

the System Requirement and Discover Route Databases' tables. Records that are not reachable from the root tables are automatically deleted from the databases; except for records in a few distinguish root tables.

In addition, the DAIM model uses this information to make management and local decisions. The observation includes changing of network links, network topology, changing of host location, so that would facilitate the calculation and install shortest route. The Control database resides within each switch, which synchronizes all information with other databases. Moreover, the Control database can also enhance agents to make management decisions locally and maintain the system in case uncertain changes such as failure of the main databases. There are three Databases of the DAIM model include: (1). System Requirement Database; (2). Discovered Rout Database; (3). Control Database.

We are proposing that by creating a DAIM model on the networks we could give effect to what we are calling a Reactive Interpreter Network. So it would be a truly distributed computing environment, where these DAIM agents reside in the network elements, which would be OpenFlow switches. The actual values in the OpenFlow tables, reside in the OpenFlow switches, and would then be the properties of DAIM agents. These agents would then have to do the work of modifying or adapting their values so as to implement the requirements of the network. So the whole DAIM model stretches across all these network elements and then could be thought of as reactive distributed interpreter that is interpreting the system requirements to enable the infrastructure to provide for the business needs.

V. Packet processing within DAIM model

Packet sending in this structure, using the OpenFlow environment, include DAIM, would typically depended on other structural components such as databases to obtain all network information needed, and to achieve autonomic functionalities as well as obtaining information of the entire network. Moreover, each database actively synchronizes with others according to the events registered. The DAIM cloud can publish events to databases and actively synchronize, so that other switches can reconstruct the whole information about the network. Individual switch can serve any coming packets locally or from other switches. As a benefit of databases, they give ability of self-configuration if any local change happens within individual switches, to adapt other switches. Thus, the distributed system structure has the feasibility to deploy the DAIM cloud, which has the ability to synchronize information of the entire network. As a result, it is possible to achieve that self-management when enabling all autonomic functions.

When the packet hits the OpenFlow switch, it performs the operations shown in Fig. 3. Packet headers are used for table look-ups depending on the packet type, and typically include various packet match fields, such as Source IP, Destination IP, and MAC destination address. The switch begins with performing a table look-up in the first flow table, and may perform table look-ups in other flow tables [28]. For example, the flow tables are sequentially numbered, so the packet is matched against flow entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the table 0. If a flow entry is matched, the instruction set included in that flow entry is executed and the counters associated with the selected flow entry must be updated. Those instructions may direct the packet to another flow table, where the same process is repeated again. On the other hand, the instructions could forward the packet if not matched to Table-miss flow entry or the DAIM cloud.

If there is no matching rule in the flow table for that particular packet, it sends it to a Table-miss entry. The behaviour on a table-miss depends on the table configuration and using wild-carding rules. The table-miss flow entry in the flow table may specify how to process unmatched packets by other flow entries in the flow tables. This may include sending to the DAIM cloud or direct packets to a subsequent table. Moreover, the table-miss flow entry behaves similarly to any other flow entry. Where it does not exist by default in a flow table, the DAIM cloud may add it or remove it at any time, and it may expire. However, if the table-miss flow entry does not exist, by default packets unmatched by flow entries are sent to DAIM cloud.

The DAIM cloud has a multi-agent operating system such as JADE (Java Agent DEvelopment Framework) that can create, change, and terminate the intelligent

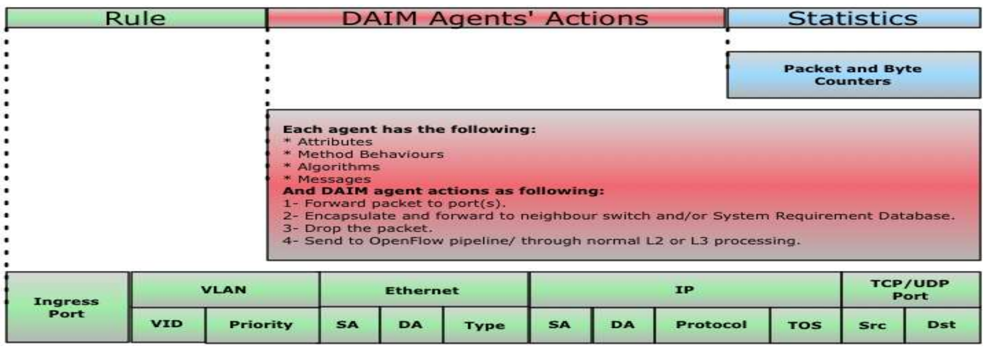
Rule
DAIM Agents' Actions
Statistics
Packet and Byte Counters
Each agent has the following:
* Attributes
* Method Behaviours
* Algorithms
* Messages
And DAIM agent actions as following:
1- Forward packet to port(s).
2- Encapsulate and forward to neighbour switch and/or System Requirement Database.
3- Drop the packet.
4- Send to OpenFlow pipeline/ through normal L2 or L3 processing.
Ingress Port
VLAN
VID
Priority
Ethernet
SA
DA
Type
IP
SA
DA
Protocol
TOS
TCP/UDP Port
Src
Dst

Figure 4: DAIM agent owns flow entries in the flow table

DAIM agents [29]. Essentially, the DAIM agents have the responsibility to maintain their own values, and they can adapt and modify their own value. According to the collected information DAIM agents can make their own local decisions based on the system requirements. In addition, DAIM agents will be bounded to a particular variable such as flow entry variables and have some level of self-adaptation strategy to manage the variables for forwarding according to the business needs. The properties or values are familiarity notions of object-oriented programming. Therefore the DAIM agents have the ingredients to implement autonomic behaviours. For example, when the DAIM cloud receives an unmatched packet, it creates DAIM agents which can access and control network elements such as the databases, and other switches to determine the forwarding rules. The DAIM agents should be able to check this flow against system requirements and other policies to see whether it should be allowed, and if allowed the DAIM agent needs to compute a path for this flow, and install flow entries on every switch along the chosen path. Finally, the packet itself will be forwarded (see Fig. 4)

The DAIM agents provide a distributed environment where the network information is the property (values) of software agents residing in virtual machines that are distributed throughout the network elements.

VI. Risk Scenarios of DAIM model

DAIM's current implementation comprises with central databases, which are System Requirement Database and Discover Route Database master. However, the DAIM model being distributed by synchronizes between the central databases and Control Database, which reside on each switch. So, the switch is responsible to fully serve all packets within its site, unless failure happens. If a failure happened then hosts that connected to the failure switch should be reconfigured to the nearest switch instead of that failed switch. The new switches can actively synchronize among all databases to know all the information and the requirements to serve connected hosts using adaptation strategies.

The SDN architecture relies heavily on a centralized paradigm, whereas the DAIM model is distributed. The failure consequences of the System Requirement Database and Discovered Route Database are the following:

Firstly, if the unknown flow arrives, it will not be able to forward and calculate a path to the destination. Furthermore, the switch will converts flow to be handled by Ethernet switching operation. However, the system will not perform optimally because autonomic functions will be disabled. For example, traditional Layer 2 switching capabilities, VLAN isolation, and QoS processing.

Secondly, the self-X autonomic functionalities are not able to store accumulated information in those databases, to perform some autonomic actions such as self-adaptation, self-configuration and self-protection [30].

However, the above issues can be avoided by the design and functions of Control Database (Buffer) and DAIM cloud, to maintain network state in case of any failure. Initially, DAIM model can be actively synchronized with the rest of the system components upon starting by using DAIM agents. The network information collected by these DAIM agents is served as a heartbeat of the proposed model. DAIM agents also exchange information generated by any switch and immediately synchronise them within all databases. Thus, collaboration of network elements can provide autonomic services such as self-adaptation and self-learning.

VII. Conclusions

Our approach was a combination of previous work approaches to gain a distributed active environment. This paper described the limitations of current OpenFlow-based SDN. In addition, it introduced programmability into the distributed network environment, illustrated in the SDN concept, with some level of distributed functionalities. SDN has a flow-based forwarding and separation of the control plan from the data plane, and provides new flexibility in network innovation. However, the new system requires some changes in the SDN approach. In this regard, implementation of the DAIM model through the compiled interpreted reactive paradigm within SDN environment have been proposed. Moreover, this paper introduced the concepts of autonomic communications and suggested how to implement them using the DAIM model. The new system can enable the development of different network services as network applications embedded with autonomic agents. This new paradigm can be applied to other infrastructures or distributed environments that provide global services such as the National Broadband Network (NBN). The future work will be experimenting and evaluating the DAIM model using Omnet++ simulation, focusing specially on Open VSwitch capabilities.

Acknowledgment

This work is sponsored by the Centre for Real-Time Information Networks (CRIN) in the Faculty of Engineering & Information Technology at the University of Technology, Sydney (UTS).

References

[1] ONF, 2012, "Open network foundation white paper, market education committee, software-defined networking: The new norm for networks, viewed 11-07-2012 www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf."

[2] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise wlans with odin," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 115–120.

[3] L. R. Bays and D. S. Marcon, "Flow based load balancing: Optimizing web servers resource utilization," *Journal of Applied Computing Research*, vol. 1, no. 2, pp. 76–83, 2011.

[4] F. Chiang and V. Mahadevan, "Towards the distributed autonomy in complex environments," in *Information and Multimedia Technology, 2009. ICIMT'09. International Conference on*. IEEE, 2009, pp. 169–172.

[5] F. Chiang, "Self-adaptability, resilience and vulnerability on autonomic communications with biology-inspired strategies," *PhD thesis University of Technology Sydney, Australia*, 2008.

[6] R. Braun and F. Chiang, "A distributed active information model enabling distributed autonomics in complex electronic enviornments," in *Broadband Communications, Information Technology & Biomedical Applications, 2008 Third International Conference on*. IEEE, 2008, pp. 473–479.

[7] J. Case, M. Fedor, M. Schoffstall, and C. Davin, *A simple network management protocol (SNMP)*. Network Information Center, SRI International, 1989.

[8] J. Strassner, S. van der Meer, and J. Hong, "The applicability of self-awareness for network management operations," *Modelling Autonomic Communications Environments*, pp. 15–28, 2009.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[10] Z. Cai, A. L. Cox, and T. E. N. Maestro, "Maestro: A system for scalable openflow control," Technical Report TR10-08, Rice University, Tech. Rep., 2010.

[11] J. Werner, "Description of network research enablers on the example of openflow," *New Network Architectures*, pp. 167–177, 2010.

[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, *et al.*, "Onix: A distributed control platform for large-scale production networks," *OSDI, Oct*, 2010.

[13] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[14] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker, "Frenetic: a high-level language for openflow networks," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 2010, p. 6.

[15] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2010, pp. 19–19.

[16] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd International Teletraffic Congress*. ITCP, 2011, pp. 1–7.

[17] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[18] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[19] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," *Proc. INM/WREN*, vol. 10, 2010.

[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *SIGCOMM-Computer Communication Review*, vol. 41, no. 4, p. 254, 2011.

[21] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2010, pp. 351–362.

[22] N. Mohan and M. Sachdev, "Low-leakage storage cells for ternary content addressable memories," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 5, pp. 604–612, 2009.

[23] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Internet Measurement Conference: Proceedings of the 4 th ACM SIGCOMM conference on Internet measurement*, vol. 25, no. 27, 2004, pp. 115–120.

[24] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[25] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2008, pp. 1–9.

[26] P. Pupatwibul, B. Jozi, and R. Braun, "Investigating o: Mib-based distributed active information model (daim) for autonomics," pp. 7–12, 2011.

[27] T. Feng, J. Bi, H. Hu, and H. Cao, "Networking as a service: a cloud-based network architecture," *Journal of Networks*, vol. 6, no. 7, pp. 1084–1090, 2011.

[28] ONF, 2012, "Open network foundation, openflow switch specification version 1.3.0 (wire protocol 0x04) viewed 20-08-2012 www.opennetworking.org/images/stories/downloads/specification /openflow-spec-v1.3.0.pdf."

[29] V. R. Komma, P. K. Jain, and N. K. Mehta, "An approach for agent modeling in manufacturing on jadeŹ reactive architecture," *The International Journal of Advanced Manufacturing Technology*, vol. 52, no. 9, pp. 1079–1090, 2011.

[30] F. Chiang and R. Braun, "Self-adaptability and vulnerability assessment of secure autonomic communication networks," *Managing Next Generation Networks and Services*, pp. 112–122, 2007.

Ameen Banjar received his B.Sc from Taibah University (Saudi Arabia) and M.I.T advanced from University of Wollongong (Australia). He is currently working towards Ph.D. degree in Computing and Communications, at University of Technology Sydney UTS, Faculty of Engineering and Information Technology. He began his working career as a database designer and programmer at Taibah University, Information Technology Centre (ITC) in Saudi Arabia, for two years. He has a research interest in network management, especially in the area of intelligent agent-based network management systems.

Pakawat Pupatwibul is currently working towards the Ph.D. degree in Information Systems, faculty of Engineering and IT from University of Technology Sydney, Australia, having graduated from Naresuan University with a B.Sc. in Computer Science, and Master's of Information Technology from UTS. He has worked attentively as a network administrator for Suan Dusit Rajabhat University, a government sponsored university in Thailand, for 7 years. His research interests include next generation networks, data center network, QoS and network management, especially in the area of intelligent agent-based network management systems.

Robin Braun received his B.Sc (Hons) from Brighton University (UK), and his M.Sc and Ph.D from the University of Cape Town. He holds the Chair of Telecommunications Engineering in the Faculty of Engineering and Information Technology of the University of Technology, Sydney, Australia. He is an executive member of the Centre for Real Time Information Networks (CRIN) at the University of Technology, Sydney (UTS). Prof. Braun was a member of staff of the Department of Electrical Engineering of the University of Cape Town from 1986 to 1998. He was the founder, and Director of the Digital Radio Research Group at the University of Cape Town, which supervised over 50 research degree candidates in the years that he was attached to it. Prof. Braun is currently a Senior Member of the Institute of Electrical and Electronic Engineers of the United States (IEEE).