A Testbed For Policy Driven Closed Loop Network Management

Joseph McNamara*, John Keeney†, Liam Fallon‡, Sven van der Meer† and Enda Fallon*

*Software Research Centre, Athlone Institute of Technology, Athlone, Ireland
j.mcnamara@research.ait.ie efallon@ait.ie
†Network Management Lab, Ericsson, Athlone, Ireland
john.keeney@ericsson.com liam.fallon@ericsson.com sven.van.der.meer@ericsson.com

ABSTRACT—Due to the increase in the dynamicity, programmability, scope and complexity of modern networks there is a greatly increased requirement that network management systems control, orchestrate and manage networks in a much more automated and dynamic manner. This drive towards automation and dynamicity requires autonomic network management that continuously analyses network state and continually steers the network in accordance with changing high level goals and policies. As dynamicity increases, it is proving increasingly difficult to test and validate the analytics routines and policies that drive today’s network management systems. With more automation, the potential for unanticipated network incidents increase, for example where multiple automation features interact and conflict.

There is no substitute for seeing how a network management feature actually performs in a real network, ideally allowing iterative authoring/validation development cycles. However, due to the high stakes involved in degrading or disrupting network performance, this is not usually feasible until the very final testing and deployment stages. The next best option is a testbed that accurately represents a live network scenario to support authoring and validation development cycles in a low-risk environment.

In this work we present our experiences of building a networked testbed that incorporates an emulated network, a production-grade network controller, an analytics function, and a policy execution environment. This allows users to develop policies for adaptive (closed loop) management of a realistic emulated network. We also present two scenarios where the testbed is used to emulate and mitigate against a temporary and prolonged failure occurring on a network.

I. INTRODUCTION

Since networks are now truly open and programmable, the time for dynamic, adaptable and adaptive networks has arrived. It is important for the systems managing these networks to be equally open, dynamic and adaptive. There are numerous network controllers available to mediate programmable access to these networks. Policy-based management can be used to drive these controllers, but now policies must support a more holistic view of how the network should be managed, taking users, services and context into account. The performance of networks must also be analysed in a holistic manner, again taking users, services and context into account. This combination of network, controller, analytics and policy allows us to manage and optimise networks using adaptive closed loop automation (e.g. COMPA [1], ONAP [2]), realising autonomic networks where new user- and service-aware network configurations (e.g. slices) can be adaptively controlled, orchestrated and managed.

Dynamic networks, dynamically analysed and driven by adaptive high-level goals or policies are hard to plan and manage, resulting in networks that are not correctly dimensioned or demonstrate unanticipated behaviours once deployed. Dynamic systems that are not adequately modelled are hard to simulate or evaluate analytically before being deployed, or alternatively must be tuned during testing or after deployment and field. A better alternative is a realistic testbed that emulates the network, interacts with real controllers using real policies, and carries real traffic, so when the policies change, their effects can be analysed in a realistic manner.

Such a testbed has a number of key requirements. a) The testbed must be lightweight and easy to use, preferably using a DevOps type approach for implementing network management scenarios. b) The behaviour of the testbed must be realistic, so it should use the actual analytics framework, policy system and controller used in the real network. Real scenarios can then be evaluated. c) The testbed should be open in order to facilitate integration of new components. d) Fast loop-times must be supported, where the time for one execution of the control loop could be as low as 50 ms, or as high as 5-10 sec. This makes the testbed suitable for evaluating control-plane options and strategies. e) Fast reproducible experimentation must be supported, allowing rapid evaluation of different management strategies.

While there are numerous network simulators and emulators, several candidate network controllers, many network analytics platforms and various policy/rule systems, there lacks an existing reusable testbed to evaluate closed-loop network management that meets these requirements. In this work we took the first steps to develop such a testbed, with a particular focus on allowing us to quickly evaluate alternative network policies executing in our APEX platform [3]. With this testbed policy authors can deploy and evaluate their policies on an emulated network and monitor in real time the affect their policies have on the network.

In §II we present the components initially used in our testbed, and discuss how closed control loops can be realised. In §III we present two user- and service-aware scenarios which showcase the functionality of the testbed and in §IV show...
how these scenarios were implemented in the testbed. In §V we show the results of how those scenarios operated in the testbed, then describe our experiences in building the testbed and designing the scenarios. Finally in §VI we close the paper with conclusions and a short discussion of future directions for this work.

II. TESTBED ARCHITECTURE

![Diagram of the Closed Loop System Testbed Architecture](http://mininet.org)

This section describes the architecture of our closed loop testbed. The architecture, shown in Figure 1, is designed to be as open and repeatable as possible, in line with the requirements described in section I. The testbed is designed as an autonomic closed loop system. Inspired by the COMPA [1] reference architecture, it has the four main components shown in Figure 1. Communication in the architecture is realised using a distributed component approach [4].

The testbed provides a closed loop system which enables an adaptive policy to continuously manage the virtual network in an autonomic manner. When the virtual network experiences a change, information gathered in the SDN controller is packaged as an event. This event is analysed by the Analytics component to assess its significance. Significant events trigger adaptive policies in the Policy component. The response of the policies is sent to the SDN controller of the virtual network for deployment as a reconfiguration on the network.

The Managed Network: The Managed Network is a virtual network executing in the Mininet\(^1\) virtual network emulator. We selected Mininet because it provides a well documented and extensible Python API (Application Programming Interface) and supports a wide range of features such as hosts capable of running basic networking applications and virtual switches that support the OpenFlow standard [5] for SDN. Mininet’s lightweight nature avoids the need for installing, configuring and administering multiple orchestration systems [6] and the ease of building new virtual networks through the Python API allows for the quick and easy testing of different network scenarios. Another important capability for the testbed was Mininet’s ability to scale to emulate very large networks.

The Network Controller: We selected the Floodlight\(^2\) SDN controller as our network controller. The Network Controller is used to read the network configuration and status from the network and to configure modifications on the network. Floodlight supports OpenFlow on its southbound side through a well-defined *forwarding instruction set*. Floodlight exposes a RESTful (REpresentational State Transfer) northbound API, which is used by clients to monitor and configure the underlying network.

The Analytics Component: This component implements the domain specific analytic logic for the domain in our closed loop system. It monitors the network and if it arrives at an insight for its domain that may require intervention, it forwards that insight to the Policy component. As we used relatively straightforward scenarios in our testbed to date, we developed our own domain specific LinkMonitor analytics component as a Python program. There are many network analytic toolkits and platforms that can be used in more complex closed loop domains such as the DCAE component from ONAP [2].

The Policy Component: This component makes decisions on whether interventions should be made on the network. It uses the insights produced by the Analytics component to check if the operational goals set up for the network are being breached. If the goals are breached, the Policy component decides if and how to intervene in the network to mitigate those breaches. In our testbed, we used the APEX [3] adaptive policy engine running a set of adaptive policies designed for our scenario. We used a 4 stage Match, Establish, Decide and Act (MEDA) pattern based on the well-known OODA loop [7] for our adaptive policy. The Match stage links an external trigger to a task, Establish reads the processed trigger and relevant context information, Decide identifies a response for the situation and finally Act takes the decision and generates an actionable response. During each of these stages/states, context information is continually being updated resulting in policies making informed decisions throughout each stage of the decision making process.

In addition to the main components, we used the Kafka\(^3\) stream processing platform as a message bus for reliable communication between our distributed components. As Floodlight uses a RESTful interface, we developed a Kafka/REST interface in Python for interaction between Floodlight and the Analytics and Policy components.

III. SCENARIO

We have used a VPN networking domain to illustrate how our testbed can be used to build and actively manage a network in a closed loop manner using SDN, Analytics, and Policy. We have identified two scenarios, one where a VPN link failure threatens to breach customer SLAs (Service Level Agreements) and another for when SLAs become breached. We

---

1^http://mininet.org

2^http://www.projectfloodlight.org/floodlight/

3^https://kafka.apache.org/
have developed and deployed solutions that mitigate against the failure in an adaptive and autonomic manner for those two scenarios.

Fig. 2. VPN Domain Network Topology

A. Temporary Failure Scenario

The temporary failure scenario examines how an adaptive policy should handle the failure of a link which is shared between two customers. For the purpose of the scenario both customers are configured with a sample SLA that allows for 90 seconds of downtime per customer per year. One of the customers (Customer B) has already experienced 30 seconds of downtime in the year to date (YTD). Using the context information gathered from the virtual network the adaptive policy tries to mitigate the effect of the network loss on the SLA of Customer B by reducing the level of service to other customers who have experienced less down time in the year to date.

B. Prolonged Failure Scenario

The prolonged failure scenario examines how should adaptive policy handle the prolonged failure of a link which is shared between the a set of customers. In the case of a prolonged failure, the SLAs of all customers will eventually be breached. Using context information gathered from the virtual network, the adaptive policy prioritises a specific subset of customers as a form of damage control.

IV. IMPLEMENTATION

In this section, we describe the algorithms we have developed and implemented on our testbed for the scenarios described in section III.

Algorithm 1 initialises and controls the Mininet and Floodlight components in the testbed. In Algorithm 1, a Mininet virtual network is initialised using the predefined topology and configuration shown in Figure 2. When the virtual network has been built, the algorithm starts the Floodlight controller. It then starts a Kafka Consumer which listens for configuration events sent from the APEX engine. When an event requesting a configuration change is received, it is parsed into a REST request and is forwarded to the Floodlight SDN controller. The controller then executes the configuration change on the network in Mininet.

Algorithm 2 monitors and analyses the networking domain to check if any insights are observed. It is the implementation of the Analysis component of Figure 1 for the testbed. Algorithm 2 sends a REST request to the Floodlight SDN controller every 30 seconds (configurable), returning a summary of all the links in the virtual network. On the first iteration of the loop the information retrieved is set as the current known steady state of the system. In further iterations, the links are checked for irregularities by determining if the system has deviated from its steady state. If an irregularity is found, then this irregularity is forwarded to the Policy component as an event.

Algorithm 3 performs SLA mitigation in the event of link
Algorithm 3: The Adaptive Policy For VPN SLA Mitigation

MATCH:
Read incoming information;
Check fields against Context album;
Update Context album;
Output incoming information with additional status information;

ESTABLISH:
Read information from the Match state;
Use context information to find if the issue is new or reoccurring;
Output link context information along with the specific problem information;

DECIDE:
Read information from the Establish state;
Read SLA information
if Unbroken SLA exits then
  Read customer context information and extract affected customers;
  Select customer who is furthest away from breaking their SLA;
else
  Read customer priority information and extract customer priorities;
  Select customer with the lowest priority value;
end if
Output selected customer context;

ACT:
Read information from the Decide state;
Extract necessary customer context information for the generation of the Floodlight REST request.
Package the information into an event along with any additional information for the transportation of the response;
Output the response;

failures on the VPN network. The algorithm, shown as a flowchart in Figure 3, is composed of four MEDA states.

In the Match state, the content of the incoming event is analysed and an update is made to the current state of the network if required.

The Establish state compares the state of the link reported on the incoming event with the current steady state of the link in its network model. It checks if the update is reporting that a link that was up has failed or if a link that had failed has recovered. It forwards this result to the next state.

The Decide state examines the SLA values for the customers on the link that has failed or has come up and determines in what manner traffic should be steered to give the best possible overall SLA mitigation outcome for the given link failure and customer SLA values.

The Act state packages the appropriate response and sends it to the SDN controller over Kafka.

V. RESULTS & EXPERIENCES

![Fig. 4. Customer A and B Sharing two Links](image)

At the beginning of the scenario, Customers A and B share the bandwidth of two links. In Figure 4 we see both customers experiencing between 100-150KB/s. This is sufficient bandwidth for both streams to be viewed without a reduction in image quality as seen in Figure 5.

As the scenario progresses, one of the shared links is brought down. The Floodlight controller quickly reconfigures the flow tables allowing for the stream to continue. However, as both streams are now using a single link, a bottleneck situation arises. In Figure 6 we see the bandwidth received by both customers drops to 60-80KB/s. This bandwidth drop causes a reduction in the image quality of both customer streams, as we can see in Figure 7.
carried over the remaining link, neither of the streams provide a watchable experience.

To mitigate this situation, one of the streams must be restricted in order to release bandwidth to allow the quality of the other stream to recover. Our adaptive policy uses context information gathered from the Floodlight SDN Controller to decide which stream to restrict. In Figure 8 we can see the effect of the policy on the network. The policy ensures that at least one customer has an acceptable viewing experience during the fault situation.

In Figure 9 we observe the viewing experience of Customer A and B while A has been restricted on the network. Customer B returns to an optimal viewing experience similar to what we saw in Figure 5.

As the scenario progresses, the policy realises that Customer A’s SLA is closer to being violated compared to Customer B. It now changes its restricting decision and restricts the stream of Customer B so that Customer A’s session is restored. This check is performed at 30 second intervals which is reflected in Figure 8. The variation in time of each cycle can be contributed to the time it takes for the execution of the control loop. In our testbed this execution time averaged 2.8 seconds. In Figure 10 we see the result of this decision, with Customer B being restricted and Customer A receiving a clear image.

After a prolonged period of time both customers will violate their SLAs. At this point the policy’s decision making strategy changes, instead focusing on the customer with the higher priority. The prioritised customer’s stream will return to its optimal image quality at the expense of other. Figure 11 shows the prioritisation of Customer B near the end of the test when the policy shifts its decision making criteria from being SLA based to being priority based.

VI. CONCLUSION & NEXT STEPS

The scenarios and results demonstrate how effective the testbed is at managing a virtual network through adaptive policy execution. The testbed gives policy writers a testing
environment where they can push the boundaries of their adaptive policies by applying them to emulated networks. The network configuration can be extended and made more complex by amending and building up the configuration of the virtualised network in Mininet.

A key feature of the testbed was the inclusion of a virtual network emulator rather than a simulator. While both have their advantages, tests carried out on an emulated network provide a level of authenticity to the results of the test. The tester can trust in the integrity of the results as they replicate the hardware and software, and pass real data around rather than simulating it.

The inclusion of an industrial grade SDN controller adds to the legitimacy of the testbed by providing more flexible and dynamic routing capabilities to the emulated network. The Floodlight SDN controller is easy to work with and it is very well documented. However, we have identified some drawbacks with Floodlight. In particular, the REST API lacks expressivity and support for performing many fundamental operations. We have also noticed that since our initial selection of Floodlight there has been little in terms of contributions from the Floodlight community to the development of the controller. Without an active community behind it, the controller will quickly stagnate. Other controllers, such as the OpenDaylight controller [8], have an strong community behind them. OpenDaylight in particular supports most of the features Floodlight provided and has additional features that would be useful for the testbed.

Designing and implementing policies with the APEX engine was relatively simple and the scenarios used for this paper did not push the limits of the testbed nor the potential of the APEX engine. The state machine approach to the policy execution resulted in policies which are easy to implement while also allowing for ongoing iterative authoring/validation cycles to handle more complex scenarios.

In summary, we have presented a testbed which will aid in the testing and development of closed loop network management applications that use analytics and adaptive policies. We have shown the implementation of two closed loop management scenarios showcasing the usefulness of the testbed and the advantages of using a closed loop approach to solving network management domain problems.

We plan to continue development of the testbed, integrating more advanced analytics, adding additional (application, network, radio) controllers to extend the scope of the testbed to much more complex scenarios. We feel that a DevOps inspired iterative authoring/validation approach to verify policy specifications and analytics tasks is a fundamental requirement for supporting autonomic management. While we have chosen not to use a simulator in this testbed implementation we feel that the use of simulators compliments and augments the approach we have presented. Simulators can also play a vital role in validating the properties of management control loops at authoring/pre-deployment time, but with the caveat that properly configured emulators such as Mininet provide a more accurate result.

REFERENCES