A Hybrid Machine Learning/Policy Approach to Optimise Video Path Selection

Joseph McNamara*, Liam Fallon†, 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
liam.fallon@ericsson.com

***Abstract*—Services such as interactive video and real time gaming are ubiquitous on modern networks. The approaching realisation of 5G as well as the virtualisation and scalability of network functions made possible by technologies such as NFV and Kubernetes pushes the frontiers of what applications can do and how they can be deployed. However, managing such intangible services is a real challenge for network management systems. Adaptive Policy is an approach that can be applied to govern such services in an intent-based manner.**

In this work, we are exploring if the manner in which such services are deployed, virtualized, and scaled can be guided using real time context aware decision making. We are investigating how to apply Adaptive Policy to the problem of optimizing interactive video streaming delivery in a virtualized environment. We utilise components of our previously established test bed framework and implement a single layer neural network through Adaptive Policy, in which weights assigned to network metrics are continuously adjusted through supervised test cycles, resulting in weights in proportion to their associated impact on our video stream quality. We present the initial test results from our Perceptron inspired policy-based approach to video quality optimisation through weighted network resource evaluation.

***Index Terms*—Service Assurance, Video Optimization, OTT, Adaptive Policy**

I. Introduction

The rapidly changing capabilities and performance challenges of emerging telecommunications network requires a re-evaluation of traditional network configuration paradigms. Typically, networks adopted a policy based management system to govern their behaviour. This policy based approach has simplified the complex task of managing a network by specifying a set of preconfigured rules based on known scenarios [1]. However the constant development of new technology along with the rise of virtual networks, NFV and SDN has resulted in an exponential increase in complexity for network management systems [2]. This has created issues for the current policy based approach. Traditional policy based management systems encode logic to select from a set of predefined options rather than dynamically make a context-aware adaptive decision. Therefore, as changes occur and the scale of management tasks increase, existing management systems inevitably become static and brittle as they get more complicated. [3] Ericsson has developed an APEX (Adaptive Policy EXecution) engine to addresses the issue [4]. This paper introduces control logic for the APEX engine that implements a directed feed forward neural network to enable network path selection for multimedia streaming applications. Further control logic is introduced to evaluate the service QoS characteristics through a MOS (Mean Opinion Score), which is fed back to the neural network adjusting QoS weights, completing the optimization closed loop. Initial results illustrate how service specific metrics can be used to inform adaptive control within the context of a real network control scenario implemented by the APEX engine.

This paper is organised as follows: §II describes our closed loop supervised learning test framework, comprising of the APEX engine, Mininet network emulator [5] and the Eval-Vid video evaluation tool-set [6]. §III details the implementation of the core components of the framework. Algorithms are provided for policies, configuring the Mininet network and video evaluation. §IV discusses our preliminary test results from our 50 cycle test, accessing the applicability of our approach and identifying insights to improve the application of our Perceptron inspired policy to the emulated network. §V describes our experience and plans for future work in this area.

II. System Architecture

The diagram in Fig.1 shows our system architecture. The system runs a closed loop supervised learning cycle, with each cycle starting with initialisation of a predefined Mininet emulated network and closing with the update of the network metric weights stored in the Adaptive Policy EXecution engine. In this section we describe the primary components of system architecture and their role within the system.

Policy Context is handled by the APEX (Adaptive Policy Execution) engine. In this work we have defined two policies. The Perceptron Policy applies weights for specific Quality of Service metrics to predefined network configurations, both of which are stored as context information on the policy engine. The policy outputs the optimum network configuration for a video, based on weights stored in context. This configuration event is sent to Mininet which builds the appropriate network for the cycle and the video is stream and evaluated. The returning event triggers the Feedback Policy. The Feedback Policy stores the video evaluation metric in context and compares this value with the metric received from the previous cycle. The degree of change from the video evaluation metrics is used to adjust the weights stored in the policy context.

978-3-903176-24-9 © 2019 IFIP

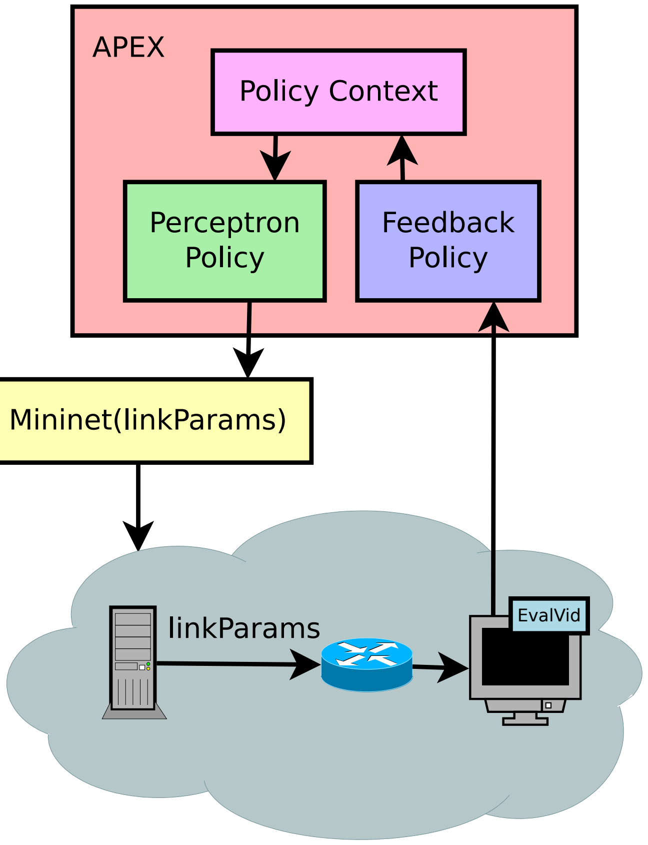

Fig. 1: System Architecture

The Mininet Framework creates an emulated network. Mininet supports rapid configuration and emulation of a virutal network running real kernel, switch and application code[1]. Transmission of real video streams over this network is the foundation of our testing environment. As network characteristics and scenario configuration can be automated without the need for simulation, emulation maintains the integrity of obtaining real data in a flexible networking environment.

Video Evaluation is carried out by the Eval-Vid toolset[2]. This application runs on the client side of the network, generating Peak Signal to Noise Ratio, the ratio of signal power and corrupting noise of both the original and streamed videos, The toolset functionality enables automated generation of a Mean Opinion Score. This Mean Opinion Score is generated by comparing the PSNR of each frame in the original reference video with the streamed video, counting the number of frames with a MOS worse than the original in a given interval. The MOS generated by Eval-Vid provides an insight to whether changes made to network characteristics have contributed to a positive or negative impact on video stream quality.

III. Implementation

This section describes the implementation of the policies, network emulation and the video evaluation process for our approach, detailed as four algorithms in pseudocode.

[1]http://www.mininet.org

[2]http://www2.tkn.tu-berlin.de/research/evalvid/fw.html

The *Perceptron Policy* (Algorithm 1) presents the four states used in the policy's application of a single layer neural network approach. *Match*: Processes a list of optional paths, one of which will be recommended for video streaming. *Establish*: Queries policy engine context information for weights corresponding to predefined link characteristics/attributes. *Decide*: Applies a perceptron single layer neural network approach through normalisation of path attributes, application of respective weights and extraction of the path identifier with the largest weighted sum of attributes. *Act*: Parses the output of the *Decide* state, prepares and outputs an event from the policy.

The *Mininet Topology & Video Stream* (Algorithm 2) shows initialisation of the Mininet emulated network topology and recording of the video streamed across the network. The network is configured with parameters for bandwidth (*bwMetric*), latency (*ltMetric*), and packet loss (*lsMetric*). A *Pingall* is executed to ensure successful initialisation of the network. *VLC* is used as the video streaming tool, executed on Node A1 and A2, configured to record the streamed mp4.

The *Video Evaluation* (Algorithm 3) shows how the video MOS (Mean Opinion Score) is generated using the Eval-Vid toolset. Firstly, a PSNR (Peak Signal to Noise Ratio) is generated for the original video file, described in the reference procedure. Next a PSNR is generated for the recorded streamed video. The comparison of the PSNR files results in a MOS value, a QoS metric with a range 1 to 5 with 1 representing exceptionally poor quality and 5 representing no degradation in quality.

The *Feedback Policy* (Algorithm 4) presents the four states used to adjust weight values when our video evaluation metric is received by the policy. *Match*: Processes the MOS into a workable format for the policy. *Establish*: Queries policy engine context information for last received MOS value. *Decide*: Using context information accompanied by the MOS value, a slope is generated to represent the improvement/degradation in video quality. A learning constant is then applied to create our weight change variable. A random adjustment is implemented every 5 cycles of the policy ranging from -0.05 to 0.05. Our new weights are then stored in our policy engine context information and a report is prepared for the next state. *Act*: Prepares the report from the *Decide* state the policy output.

IV. Preliminary Evaluation

In this section we present results of the preliminary evaluation of our approach. One goal of our preliminary evaluation is to get initial indications on the approach's applicability. Another goal is to assess if the selected learning parameters and weightings are appropriate and to determine the degree and length of time for which learning should be applied.

The first five results of a 50 cycle test are shown in Table I. Bandwidth, Latency and Loss Weights are the configured path metric for the video stream in Mininet for the current test cycle. The initial value of the weights are set manually.

Packet loss of I-frame packets can have a significant impact on video quality [7]. Therefore, we have assigned packet loss the largest weight of the three metrics. Bandwidth is assigned

Algorithm 1 Perceptron Policy

```
1: procedure MATCH                                   ▷ Match state
2:     while $e^i_m.pathList.hasNext()$ do            ▷ process pathList
3:         $e^o_m \leftarrow e^i_m.pathList.next()$
4:     end while
5: end procedure
6: procedure ESTABLISH                               ▷ Establish state
7:     $e^o_e \leftarrow e^i_e.paths$
8:     $e^o_e \leftarrow ctxt(metricWeights.bwWeight)$
9:     $e^o_e \leftarrow ctxt(metricWeights.latencyWeight)$
10:    $e^o_e \leftarrow ctxt(metricWeights.lossWeight)$
11: end procedure
12: procedure DECIDE                                 ▷ Decide state
13:    for $p$ in $e^i_d.paths$ do
14:        $weightedBw \leftarrow normalise(p.bw) * bwWeight$
15:        $weightedLatency \leftarrow normalise(p.latency) * latencyWeight$
16:        $weightedLoss \leftarrow normalise(p.loss) * lossWeight$
17:        $weightedSum \leftarrow weightedBw + weightedLatency + weightedLoss$
18:        if $weightedSum > largestWeightedSum$ then
19:            $largestWeightedSum \leftarrow weightedSum$
20:            $largestPathId \leftarrow p.id$
21:        end if
22:    end for
23:    $e^o_d \leftarrow largestWeightedSum$
24:    $e^o_d \leftarrow largestPathId$
25: end procedure
26: procedure ACT                                    ▷ Act state
27:    $e^o_a \leftarrow parse(e^i_a.largestPathId, e^i_a.largestWeightedSum)$
28: end procedure
```

a lower weight and latency is assigned the lowest weight. In the first cycle we see a reported MOS value of 1.75, as this is the first test the weights are not affected and we continue to cycle two. Cycle two reports a MOS value of 2.33, this increase in MOS from 1.75 generates a slope of approx 0.58, after applying the learning constant of 0.1 we are left with a proposed weight change of 0.058. This cycle is repeated for the remainder of the test, with weights adjusted accordingly.

The observed weights for a 50 cycle test are presented in Fig.2, detailing the weight adjustment made in each cycle and the overall weight trends. From Fig.2 we see the Loss Weight (green) exhibits an overall increase from 0.8 to 0.91, showing that although it had been assigned the largest initial weight, it did not increase significantly after 50 cycles. The Bandwidth weight (red) was assigned the second largest weight and at the end of our 50 cycle test the weight value had increased from 0.5 to 0.86, a notable increase. This increase of 0.36 shows that bandwidth had more of an impact on the MOS generated from video than initially thought, observable in the upward trend depicted in the graph. The Latency weight (yellow) had

Algorithm 2 Mininet Topology & Video Stream

```
1: procedure MININET              ▷ Mininet, Floodlight, and Kafka
2:     MO ← Mininet Object (TCLink)
3:     bwMetric ← (float) sys.arg[1]
4:     ltMetric ← (float) sys.arg[2]
5:     lsMetric ← (float) sys.arg[3]
6:     linkParam ← [bwMetric, ltMetric, lsMetric]
7:     MO ← c1                                     ▷ Controller C1
8:     MO ← n[a1, a2]                              ▷ Node A1, A2
9:     MO ← s1                                     ▷ Switch S1
10:    l1 ← l(linkParam)
11:    MO ← [l1, l2]                               ▷ Link L1 L2
12:    start MO                                    ▷ topology & controller
13:    while $\neg Pingall$ do                      ▷ wait for nodes
14:        pinghosts()
15:    end while
16:    MOCL ← Mininet Object Command Line
17:    XTB ← MOCL.xterm(NodeA2)
18:    XTB ← vlcwrapper(url).record()
19:    XTA ← MOCL.xterm(NodeA1)
20:    XTA ← vlcwrapper(stream).start(XTB.IP)
21:    XTA                          ▷ use RTP/MPEG, deactivate transcoding
22:    stop MO                                     ▷ cleanup
23: end procedure
```

Algorithm 3 Video Evaluation

```
1: procedure REFERENCE                          ▷ reference PSNR
2:     yuv ← decodeYuv(file)
3:     craw ← compRawVideo(yuv, fps, false)
4:     refmp4 ← mp4(craw)                       ▷ hint RTP transport track
5:     yuvMp4 ← decodeYuv(refmp4)
6:     refPsnr ← psrn(yuvMp4)
       STORE(refPsnr)
7: end procedure
8: procedure MOS                    ▷ streamed vs. reference PSNR
9:     if $newStreamedMp4$ then
10:        yuvStream ← decodeYuv(streamedMp4)
11:        streamPsnr ← psrn(yuvStream)
       STORE(streamPsnr)
12:        mos ← generateMOS(refPsnr, streamPsnr)
13:        output ← policyEvent(mos)
14:    end if
15: end procedure
```

Cycle:	MOS:	Bandwidth Weight	Latency Weight	Loss Weight
1	1.75	0.5	0.3	0.8
2	2.33	0.55799997	0.358	0.858
3	2.54	0.579	0.379	0.879
4	2.57	0.582	0.382	0.882
5	3.05	0.63	0.43	0.93

TABLE I: Weight adjustment for cycles 1-5

Algorithm 4 Feedback Policy

```
procedure MATCH                                              ▷ Match state
    $e^o_m \leftarrow parse(e^i_m.mos)$
end procedure
procedure ESTABLISH                                          ▷ Establish state
    $e^o_e \leftarrow e^i_e.mos$
    $e^o_e \leftarrow ctxt(mosValues.size())$
    if $ctxt(mosValues.size()) - 1 < 0$ then
        $isFirstMos \leftarrow true$
        $previousMos \leftarrow 0$
    else
        $isFirstMos \leftarrow false$
        $previousMos \leftarrow mosValues[size - 1]$
    end if
    $e^o_e \leftarrow (isFirstMos, previousMos)$
end procedure
procedure DECIDE                                             ▷ Decide state
    if $e^i_d.firstMos! = true$ then
        $slope \leftarrow \frac{e^i_d.mos - e^i_d.previousMos}{2-1}$
        $learningConst \leftarrow 0.1$
        $weightChange \leftarrow slope * learningConst$
        if $mosValues.size()\%5! = 0$ then
            $randomBw = 0$
            $randomLt = 0$
            $randomLs = 0$
        else
            $randomBw = (random() * 10) - 0.05$
            $randomLt = (random() * 10) - 0.05$
            $randomLs = (random() * 10) - 0.05$
        end if
        STORE()$ctxt(metricWeights.bwWeight) + weightChange + randomBw$
        STORE()$ctxt(metricWeights.ltWeight) + weightChange + randomLt$
        STORE()$ctxt(metricWeights.lsWeight) + weightChange + randomLs$
        $metricWeightsUpdated \leftarrow true$
        $e^o_d \leftarrow metricWeightsUpdated$
    else
        $metricWeightsUpdated \leftarrow false$
        $e^o_d \leftarrow metricWeightsUpdated$
    end if
end procedure
procedure ACT                                                ▷ Act state
    $e^o_a \leftarrow parse(e^i_a.metricWeightsUpdated)$
end procedure
```

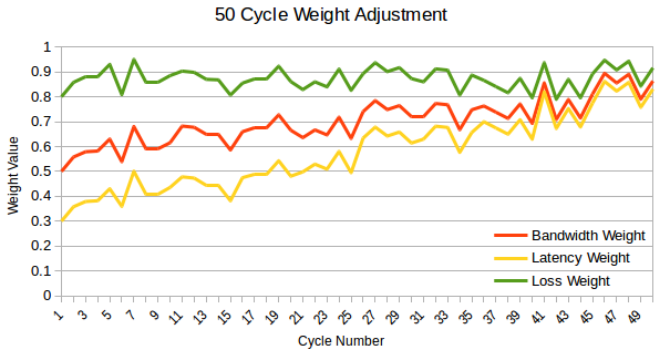

Fig. 2: Weight Adjustment for 50 Cycle Test

the most significant increase of the three weights. Increasing from 0.3 initially to end the 50 cycle test at 0.83. This steady upward trend shows that latency had a much larger impact on of video stream quality than our initial weights represent.

The results indicate that 50 cycles are not sufficient to allow weight to plateau at their representative values. Nevertheless, we were encouraged by the results. With adjustments to the our Perceptron policy learning rate in tests with more cycles, we are confident we can achieve a representative weighted path evaluation algorithm for video streams.

V. Conclusions and Future Work

This paper proposes a directed feed forward neural network for network path selection for multimedia streaming applications within the APEX Adaptive Policy Execution Engine, which outputs the optimum network configuration for a video, based on a context established through a weighted score. The selected configuration is sent to Mininet, which builds the appropriate network for the cycle, over which video is streamed and evaluated. The returning event triggers a video quality evaluation policy. The degree of change in the video evaluation metrics is used to adjust weights for the next cycle. The results generated from our initial test produced some notable insights. A clear trend is apparent in the adjustments of the weights stored in policy context. It is clear that we need to increase the number of cycles in a testing scenario to allow the changes in weights to plateau. The directed feed forward neural network policy would also benefit from a larger array of network configurations during the testing phase which would increase the learning rate of the policy.

In future work, once supervised learning has stabilised the weights for the network metrics, we will investigate whether initiating a phase of alternating between supervised and unsupervised learning to ensure the continuation of weight adjustment over a period of time while monitoring the adjustment to ensure there is no unexplained drift.

Acknowledgments

This work is partly funded by the European Commission via the ARCFIRE project (Grant 687871) under the H2020 program. This work was also supported in part by Irish

Research Council Enterprise Partnership Scheme Postgraduate Scholarship 2017 under Project EPSPG/2017/275.

References

[1] D. C. Verma, “Simplifying network administration using policy-based management,” *IEEE Network*, vol. 16, no. 2, pp. 20–26, March 2002.

[2] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, “Machine learning for cognitive network management,” *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, Jan 2018.

[3] S. van der Meer, J. Keeney, and L. Fallon, “5g networks must be autonomic!” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, April 2018, pp. 1–5.

[4] L. Fallon, S. van der Meer, and J. Keeney, “Apex: An engine for dynamic adaptive policy execution,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 699–702.

[5] F. Keti and S. Askar, “Emulation of software defined networks using mininet in different simulation environments,” in *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, Feb 2015, pp. 205–210.

[6] J. Klaue, B. Rathke, and A. Wolisz, “Evalvid – a framework for video transmission and quality evaluation,” in *Computer Performance Evaluation. Modelling Techniques and Tools*, P. Kemper and W. H. Sanders, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 255–272.

[7] Q. Dai and R. Lehnert, “Impact of packet loss on the perceived video quality,” in *2010 2nd International Conference on Evolving Internet*, Sep. 2010, pp. 206–209.