DSML: an SDN-Based Debugging Tool for Distributed Systems

Devin Ekins, Eric Eide, Hadeel Maryoosh, Jacobus Van der Merwe, Yue Fei

School of Computing, University of Utah

Abstract

(here we write the abstract) [1]

Categories and Subject Descriptors

(If applicable)

Keyword

(any keyword in the paper)

1 Introduction

(and here the introduction- test)

2 Solution

Our system allows users to define network protocols in terms of states of a Definite Finite Automaton (DFA) using our Debugger State Machine Language (DSML). Because explaining network protocols often involves a graphical representation as a DFA, we felt this would be a good way to model network status from the perspective of a debugger as well.

DSML supports the definition of states as well as the transitions between them. Transitions are defined by one or more matching operations which performs deep packet inspection to match a fields contents (or a substring of its contents) against a value. Alternatively, a state transition may be defined by a timeout wherein no matching packets were encountered.

As deep packet expansion is expensive, we provide a mechanism to avoid inspecting the entirety of network traffic. To avoid performing these expensive operations on every packet sent through the network, DSML also allows for OpenFlow filters to only funnel relevant traffic to the debugger itself. This funneling is performed at line rate. For example, a user may know that this debugger only need to consider packets with destination port of 80 for HTTP traffic. Then the debugger only performs inspection on this subset of packets.

Importantly, DSML allows for the use of side-effect operations which include logging to a file or printing to a console relevant information as well as displaying its equivalent of a ?stack trace?. This stack trace is a history of network states that were traversed to arrive at the current position as well as the packets that were matched on for each transition. This allows users to get the feedback they need to make informed debugging corrections.

The DSML protocol script written by the user is read into a compiler to produce the Debugger State Machine (DSM) which then may be run on a controller with an OpenFlow-enabled switch to begin debugging network traffic. Switches need not support OpenFlow to send traffic to the DSM, but the optimization gained by the OpenFlow filters will not be present if OpenFlow is not supported on the switch.

3 Implementation

(And here the implementation writeup)

4 Evalution

Test

5 Related Work

Test

6 Conclusion

Test

References

[1] Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D. G., Byers, J. W., Seshan, S., and Steenkiste, P. Xia: Efficient support for evolvable internetworking. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 23–23.