RedPlane: Enabling Fault-Tolerant Stateful In-Switch Applications – *Public Review*

Mythili Vutukuru
Indian Institute of Technology Bombay
mythili@cse.iitb.ac.in

The advent of programmable dataplane hardware has opened up many exciting directions in the area of networked systems research. Programmable dataplane switches and NICs allow for network elements to be programmed using high-level languages like P4, resulting in greater flexibility in the way these network elements parse and process network traffic. Today, a variety of middleboxes, and even end-host applications, are being offloaded to and accelerated using programmable dataplanes. Traditionally, applications or middleboxes running in the software controller populate various packet processing rules in the match-action tables of the switch hardware using standard controller-switch communication interfaces, and the switch dataplane uses these rules to process network traffic. However, recent research is moving some of this "control plane" functionality that configures the dataplane into the dataplane itself, by offloading some application state to the switch. This design paradigm has been motivated by the observation that the software control plane is many orders of magnitude slower than the switch dataplane. However, such "stateful" in-network applications are susceptible to incorrect operation when switch failures occur, especially when the state in the switch is written to often, because the modified application state in the switch dataplane can no longer be recovered from the control plane. While prior work has resorted to a variety of adhoc, application-specific techniques to overcome this problem, RedPlane aims to provide a more general and comprehensive solution.

The key idea of RedPlane is to perform state replication from within the dataplane itself, without involving the slower software control plane. Applications developed using RedPlane APIs will checkpoint state to an in-memory key-value store, and this replication is done entirely from the dataplane. While the idea sounds simple at first glance, realizing it is quite challenging due to the unique constraints imposed by the programmable dataplane hardware, and the paper has some interesting and elegant ideas to overcome these challenges. For example, it is not practical to use scarce switch memory for output buffering, and so RedPlane piggybacks such packets onto messages exchanged with the state store, thereby using the network itself as a temporary buffer. RedPlane uses a lightweight communication protocol with the state store, to avoid running TCP in the switch dataplane. The paper also proposes practical correctness models that leverage the fact that network applications already tolerate some amount of packet loss.

While the reviewers appreciated the problem being solved, as well as the solution proposed, a key concern was the impact of these mechanisms on application performance. The results in the paper show that while the impact on performance is minimal for applications that rarely modify the state stored in the dataplane, certain workloads do impose a significant performance overhead. However, this overhead may be a reasonable tradeoff to make for application developers, in return for the ability to achieve fault tolerance out of the box. Whether one can come up with better mechanisms for in-switch application fault tolerance that have a lower overhead is a problem for future researchers.

In summary, RedPlane addresses a very important problem overlooked by recent research on stateful application processing in programmable dataplanes. The paper has several interesting ideas that will lead to engaging discussions at the conference, as well as open up avenues for further research.