Poster: Scaling Data Plane Verification with Throughput-Optimized Atomic Predicates

Dong Guo [1] Jian Luo, Kai Gao [2] Y. Richard Yang [3]

[1]Tongji University [2]Sichuan University [3]Yale University

Abstract

Atomic predicate is a key enabler for data plane verification, usually with BDD as underlying data structure. However, BDD libraries hinder verification from real-time because they do not scale well, especially in large-scale networks.

Crux of scaling verification

Small changes may have significant impacts. For large-scale networks, data plane verification (DPV) system should accommodate millions of FIB rules, and be able to handle FIB updates that may overlap with existing rules in the same order of magnitude. Thus, there is a strong demand for a scalable BDD infrastructure to achieve scalable DPV.

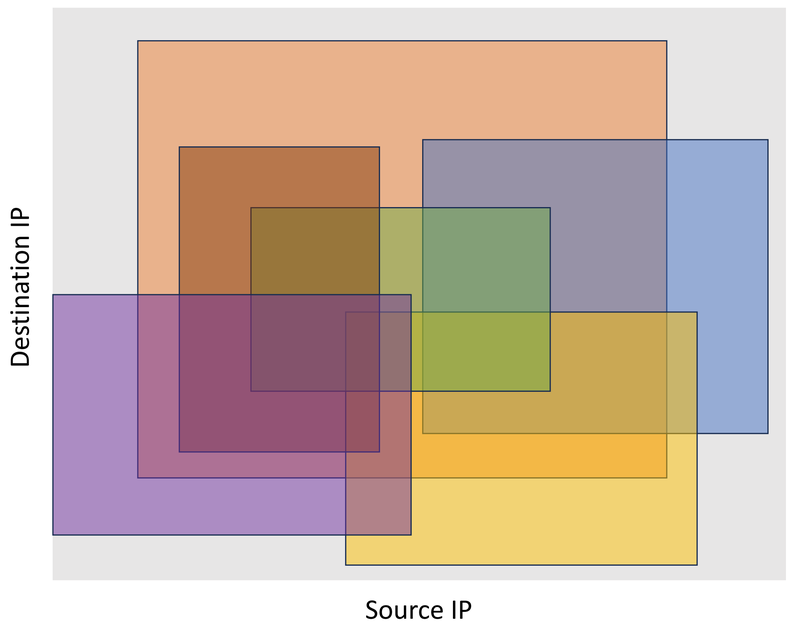

Figure 1. FIB rules splits headerspace into non-overlapping subspace, represented by atmoic predicates.

Bottleneck lies in BDD package.

The scalability of DPV systems is restricted by the scalability of BDD operations for two reasons:

1. Previous work Flash have shown that BDD operations can account for the majority of the verification time (i.e., the number of BDD operations reaches the order of $O(10^9)$ and takes over 99% of the overall time).
2. Many parallelizable processes are hindered due to the absence of parallelism support. In cases of thread-safe packages, they either adopt expensive synchronizations or unsuitable scheme. In root, existing BDD packages are not designed for throughput.

Table 1. Publicly available BDD libraries.

Name	Lang	Thread-safe	Scalability	Synchronization	Execution
BuDDy	C/C++	N	Poor	User-lock	RtC
CUDD	C++	N	Poor	User-lock	RtC
JDD	Java	N	Poor	User-lock	RtC
BeeDeeDee	C++	Y	Medium	Mutex-lock	RtC
PJBDD	Java	Y	Medium	Lock-free	FJ
Sylvan	C++	Y	Medium	Lock-free	FJ
HermesBDD	C++	Y	High	Lock-free	FJ
TOBDD	C++	Y	High	Lock-free	RtC

Designs of Throughput-Optimized BDD

TOBDD is designed to achieve high throughput parallel BDD operations by leveraging the multi-core capabilities of modern hardware.

- Reschedulable Thread-pool to eliminate the overhead of frequent thread creation and destruction,
- Lock-free Node Table and Cache. Inside it leverages CAS to achieve lock-free and dynamic linked lists to avoid resizing
- Run-to-Completion to avoid the overhead of interruption.
- Commutative Hash Function to reduce cache misses.

Use case: Scaling DPV with TOBDD

We implement TOBDD [a] and a throughput-optimized version of Flash called TOFlash in C++. We compare the data plane model construction time of TOFlash with our previously implemented Flash artifact on various datasets. Figure 2 shows the evaluation results.

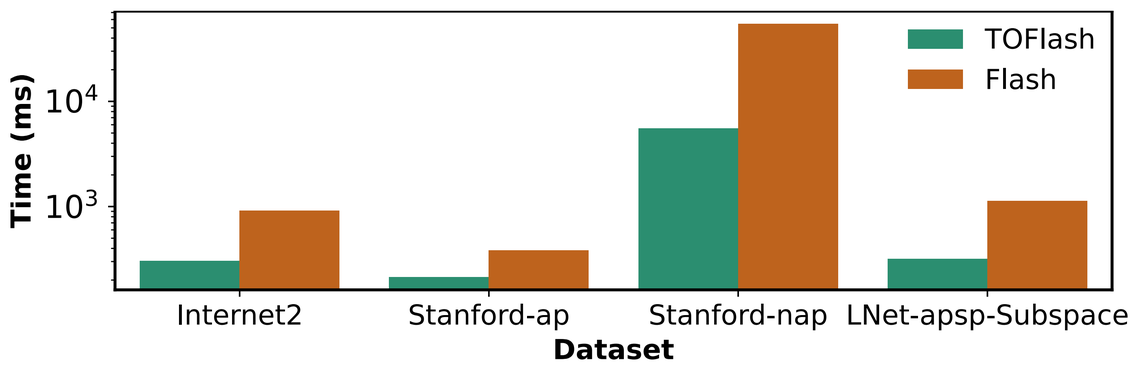

Figure 2. TOFlash achieves 2-10x improvement compared with the baseline.

[a]https://github.com/guodong/tobdd

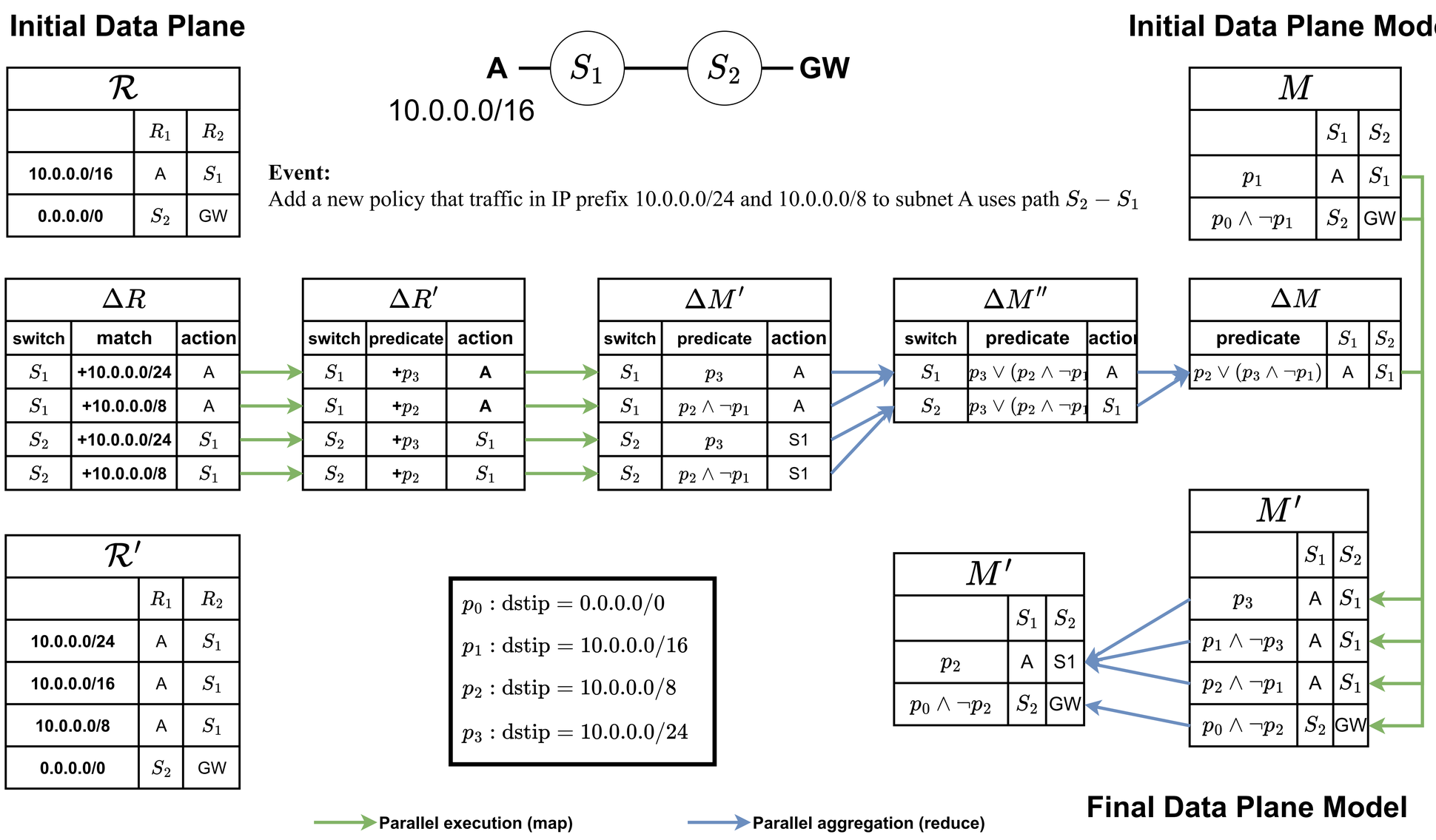

Figure 3. An example of ustilizing parallelism in DPV