CCR Paper #731

#731 In-Band Synchronization for Distributed SDN Control Planes

Accepted 237kB Tuesday 1 Sep 2015 6:44:39pm EDT |
c2e9224b03eeee46d49b98abe8e28412ba9d2754

You are an **author** of this paper.

Comment notification

If selected, the system will email you when updated comments are available.

PC conflicts

None

+ Other conflicts

+ Abstract

Control planes of forthcoming Software-Defined Networks (SDNs) will be distributed: to ensure availability and fault tolerance, to improve load-balancing, and to reduce overheads, [more]

+ Authors

L. Schiff, S. Schmid, P. Kuznetsov [details]

+ Topics and Options

	Tim	Nov	Cla	Rec
Review #731A	4	3	3	1
Review #731B	4	3	3	2

1 Comment by S. Schmid (Response)

Edit paper | **Edit response**

Reviews and comments in plain text

Review #731A

Modified Saturday 24 Oct 2015 11:38:13am EDT

Plain text

Timeliness (?)

4. Hot topic with a considerable amount of active work

Novelty (?)

3. Distinct addition to the state of the art

Clarity (?)

3. Clear, but with rough patches

Recommendation (?)

1. Accept

Detailed comments

I asked one questions in my earlier review that I wish the paper answered:

- Do real switches have the space to use all these extra rules for coordination?

I had also remarked:

"A deeper issue that this article doesn't address is whether this kind of abstraction is even desirable. A simpler abstraction may be to have one controller for a small set of switches and

have the controllers coordinate with each other."

The response essentially concedes this point by saying,

"While it is difficult to prove that our abstraction is the best or even the right one, we believe that our approach represents a meaningful first step."

A meaningful first step for what? The response to this point really don't make sense.

However, I think the paper is technically correct and may even amuse some OpenFlow enthusiasts. But, I don't think the abstraction presented is useful. But, that should not stop papers from being published. I recommend accepting with a minor revision: state whether switches support the features used.

Review #731B

Modified Tuesday 27 Oct 2015 5:28:01am EDT

Plain text

Timeliness (?)

4. Hot topic with a considerable amount of active work

Novelty (?)

3. Distinct addition to the state of the art

Clarity (?)

3. Clear, but with rough patches

Recommendation (?)

2. Revise-and-resubmit to next issue

Summary of contribution

The paper discusses an interesting way to controller synchronization. Instead of doing out-of-band synchronization between controllers, the switch could mediate the synchronization as it is the element implementing the controller policies anyway.
The authors first motivate the problem using two load-balancing controllers both modifying the switch configuration of independent flows (so there are no "conflicts" in a standard sense) and how can this lead to undesired state.
The problem could be solved if the switch supported traditional concurrency primitives such as compare-and-swap.
Finally, authors show how the compare-and-swap can be implemented right now by abusing rule overlap checking in OpenFlow.

Detailed comments

The paper is in much better shape than in the first submission and normally I would recommend accepting it. However, after re-reading it, I feel that its readability and usefulness for a general audience could be still improved. My suggestion would be something like this:
-- Turn the paper more into a vision paper on the need of in-band synchronization than the current "we can hack it in

OpenFlow". The idea is certainly interesting, examples are motivating enough but if we really need this kinf of synchronization, I would rather add a special locking/synchronization primitive to OpenFlow than hack our way through the switch tables (which, after having experiences with several hardware OpenFlow switches, I would bet that your solution does not work on half of the switches simply because they do not follow the spec as they should).
-- This means moving section 5 (Applications) before the section 4 (Implementation). Note that Algorithms 6 and 7 would be moved sooner and this is definitely fine -- they would illustrate the usefulness of Compare-and-swap and claim primitives
-- By this, you would set up a stage especially for claim operations which kind of look ad-hoc without a proper background on why they could be useful and we should care implementing them.

-- Important issue: Algorithm 5 seems to work *only* if we already wrote some value to an address. Otherwise, it happily inserts the flow (and thus actually writes value k to the address!)

Other comments:

-- Is it really needed to have _MAGIC actions? You never rely on the content of the actions nor explain why the actions cannot be simply empty (e.g. drop)

-- the description why claim() is ever needed is somewhat unsatisfying. The same goes with algorithm 7 -- the added complexity simply does not explain "so that policy identifies do not grow unbounded". You definitely should showcase a weakness in Algorithm 6 which is based in potential wrapping of policy ids as they are a fixed-bit-width numbers. Similarly, a discussion of how quickly the ids could be wrapped around might be helpful (if it takes more than a day to wrap my numbers, I do not really care because the controller should definitely re-read the switch state by then)

-- I find "compare" primitive a bit weird to throw an error and abort a transaction/bundle. Maybe call it "assert_value" which should be closer to its semantic.
-- Similarly, "check" should be "assert_unclaimed"
-- Using "pid" as "policy id" is a bit confusing to people working in systems. Maybe rename it to "p_ver" or just "ver" (policy version) which is anyway more close to its semantics

-- In your algorithms 1-5, you try to use as much OpenFlow-ish description as possible. I think this is at the expense of readability. For example, instead of using complicated value & mask for match, why not describe match using 0,1 and * (wildcard)? Similarly, there is no need to represent a match as an x-bit integer, it might be just a tuple
-- Algorithm 1 would be "FlowAdd(match=(*,self,x))", Algorithm 2 would be "flow=FlowAdd(match=(self,*,x))), FlowDelete(flow)". Now, it is much more obvious that (*,self_1,x_1) and (self_2,*,x_2) do not intersect if x_1 !=

x_2 but they do intersect if x_1 == x_2 as they have common match (self_2,self_1,x_1). You might also explicitely state the common match in the text to make it really obvious.
-- It would also showcase the crazyness of what algorithms 4 and 5 are doing. Their mask is "mask= (replace_0_bits_with_wildcards(k), addr)" which uses some very interesting selective bit-wildcarding to do the trick. It took me a lot of time to understand this and more importantly, I almost overlooked that k appears in the mask and not in the value part of the match. Then I was thinking why this cannot be the opposite (k in value, 1^32 in mask) but that makes no sense. And again, a description why matches for different values of k overlap will be usefull (e.g. 1^32.addr always matches both rules)
-- In any case, the current version "value=self.x; mask=1^48" is very hard to interpret as you have two numbers (of unknown widths) in value but a single number in mask. If not something else, please at least concatenate it the same way. This is especially important for Algorithm 3 where you actually do have zeroes at the beginning in the mask and you should explicitely concatenate with zeroes there

-- You should present algorithm 4 and 5 before algorithms 1-3. In particular, only algorithms 4 and 5 are needed for implementing CAS (and thus algorithm 6) while algorithms 1-3 are needed for implementing algorithm 7. This would put the read/write algorithms with easy-to-imagine usecase before less obvious claim()-ing operations.
-- In algorithm 4, you are using OpenFlow "delete" command which is different from delete_strict. I would suggest you use strict version in algorithms 2&3 and make a special remark about non-strict nature of delete in alg 4.

-- Add a note to alg 2 & 4 that the two commands should be executed atomically & sequentially

-- Algorithms 6 & 7 -- "read_config" would be better as "read_state" to hint that we are requesting runtime configuration of the switch, not some static configuration file

-- Do not call algorithm 7 "more efficient" -- both algorithms 6 & 7 can be quite frequent restarts under high contention (after all, they use optimistic concurrency control). I believe the only advantage of alg 7 is that it guarantees
-- Algorithm 7 -- it is quite confusing that you mix "pid" and "my_id" identifies and they both are actually policy versions. Instead of my_id, maybe use new_pid (or even better, new_ver, see comment about about pids)

-- "execute-atomic" -- maybe rename it to "execute-transaction" ? Executing atomically several changes which needs to be applied sequentially sounds a bit weird.

Response Stefan Schmid <stefan.schmid@tu-berlin.de> | Tuesday 1 Sep 2015 6:51:12pm EDT

Please use this form to indicate the changes you have made to your paper in response to the reviews. Please keep the response short and to the point.

Dear Dina, dear reviewers:

thank you very much for your time and good feedback!

We revised our paper according to your suggestions. In particular, we re-structured the paper significantly to better differentiate between the conceptual and the OpenFlow specific parts, and we emphasize more and earlier the aplications. We also performed some experiments of our transactional interface, and briefly report

☑ This response should be sent to the reviewers.

Save Delete response

HotCRP Conference Management Software