# CCR Paper #698



## #698 In-Band Synchronization for Distributed SDN Control Planes

**Revise and resubmit** 255kB Monday 1 Jun 2015 4:40:53pm EDT |
f4d88fcf62b206fe9f7574b4c8dd4b15d213ae72

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

**Comment notification**

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

**PC conflicts**

None

**+ 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 #698A | 4 | 3 | 4 | 1 |
| Review #698B | 4 | 3 | 2 | 2 |
| Review #698C | 4 | 3 | 3 | 2 |

**1 Comment** by anonymous

**Edit paper** | **Add response**

Reviews and comments in plain text

## Review #698A

Modified Wednesday 22 Jul 2015 5:56:54pm 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** (?)

**4.** In good shape, with minor improvements necessary

**Recommendation** (?)

**1.** Accept

**Summary of contribution**

Centralized control is a powerful abstraction of Software-defined networking, but performance and reliability requirements dictate that actual implementations are distributed. How such distributed controllers should coordinate and interact with the common resource (i.e. the network configuration/state) is an active research question. Previous papers from the authors have proposed _transactions_ as a fundamental abstraction for network coordination: controllers treat the network configuration of a single switch as a database, and update it through a transactional interface. This paper shows how the coordination primitives necessary for enforcing consistency/isolation of transactions can be implemented directly in the OpenFlow protocol, without special support.

The key idea is to use the limited, built-in transactional mechanism in OpenFlow (bundles) for atomicity and to use the ability to detect overlapping rules to detect conflicts between transactions and abort. Transactions are wrapped in bundles, which automatically provides atomic semantics. To detect conflicts, they carefully use a space of "ghost" rules that do not affect forwarding but are instead used as a small communication channel between controllers.

They then use these mechanism to implement a mechanism for controllers to claim unique ids on each switch, and then implement the compare-and-swap primitive that can be used in many different higher-level abstractions.

**Detailed comments**

Overall, the work seems useful and correct. My main concern is scope. The paper is essentially a note on OpenFlow hacking, and thus might be of limited interest to many CCR readers. However, OpenFlow and similar SDN approaches have seen sufficient adoption and research interest in recent years that there is utility in this paper appearing in a venue with broad reach.

A lesser concern is novelty. With the addition of atomic bundles to OpenFlow, it's not completely surprising that the atomic CAS operation can be implemented, but there is definite value in having it spelled out cleanly.

I found the brief argument for in-band control vs out-of-band control unconvincing. There are many high-performance systems that use consensus, and a full architectural comparison of the two approaches to SDN is beyond the scope of this paper. I certainly believe that in-band control is a useful and interesting design choice, but not just because of FLP. It also seems almost certain that a full system would require out-of-band coordination between controllers: a full transactional system needs application specific ways of resolving conflicts, which will require coordination.

I also found 2nd example of the limitations of bundles confusing. A simple concrete example would be greatly helpful, and strength the argument.

Typos/corrections:

p.1 col. 2 para. (Contributions): 2nd "conflicts" -> "a conflict", "does not affect" -> "affects neither"
" " para. 3: "promote" -> "propose" (?)

p. 3 col. 2 para (Configuration space): "provided a total addressing scheme" sentence is confusing.

## Review #698B

Modified Friday 24 Jul 2015 9:39:52am 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** (?)

**2.** Needs major improvement for readability

**Recommendation** (?)

**2.** Revise-and-resubmit to next issue

**Summary of contribution**

This paper discusses an interesting issue of coordinating multiple controllers when interacting with an OpenFlow switch.
In particular, the paper shows how one can implement compare-and-swap and locking/reserving identifiers purely using a current version of OpenFlow protocol.

**Detailed comments**

The idea of the paper is really nice and I believe it can be used in practice. However, the presentation of the paper needs major work -- it feels to be switching between high-level overview versus low-level details
- is background section necessary? Cannot you introduce details like bundles, overlap_checking and actual OpenFlow flags progressively and on examples?
- maybe start with another example with controllers updating overlapping flows. Then explain ability to do overlap_check. Then introduce load-balancing example and the need for external synchronization. Transactions/bundles are kind-of orthogonal and could be explained after you state that you want execute-atomic{ops} and that there is a native support for it in OpenFlow if ops are OF commands.

- Figure 1 -- time axis is too inconspicous
- when you described execute{op1,...} and execute-atomic{op1, ...} I was totally confused what is the definition good for. Only later on I realized that operations there are actually not OpenFlow operations but your custom synchronization primitives. Please, clarify that in the text.

- what does "mask=1...1" mean? Is it a) exact match or b) everything wildcarded? From analysing algorithm 5 I believe it is a)

- Do the algorithms 2-4 really need to use value=x.x?
-- Cannot it be "0..0 . x" ? Or "1...1 . x"? Or for an arbitrary fixed constant "c.x" ? Maybe elaborate in the text on that point because it looks like magic to me.
-- In any case, you should explain more why "x masked by self1" overlaps with "x masked by self2" exactly iff self1 != self2

- On my first reading I missed that claim() operation can actually allow more controllers to claim the same value. Please clarify in the text.
- What is the value of such operation? Actually, it looks like a kind-of "read lock".
- following the thought, additionally to claim() and unclaim(), could you have algorithms for exclusive lock() and unlock()? Maybe even explain it sooner than claim() so that people can think about well-known semantic first.

There looks to be an inconsistency of what happens with check_overlap flag for the exactly same rule:
- You say "a FlowMod command can be equipped with a the check overlap flag: if the switch maintains a flow entry with an overlapping but not identical match part with the same priority but a different action, and if the two rules have the same priority, then FlowMod will fail."
- OF 1.4 spec says "For add requests (OFPFC_ADD) with the OFPFF_CHECK_OVERLAP flag set, the switch must first check for any overlapping flow entries in the requested table. Two flow entries overlap if a single packet may match
both, and both entries have the same priority. If an overlap conflict exists between an existing flow entry and the add request, the switch must refuse the addition and respond with an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMFC_OVERLAP code."
- the consequence is that I believe your explanation allows for no error if we add exactly the same rule but the spec requires an error. How do real switches implement this? Does this affect the validity of your algorithms?
- in Algorithm 6 you seems to be assuming that same match with same actions won't cause an error but same match different actions will. This is neither consistent with your explanation of overlap check nor the spec

- Algorithm 3 -- I believe commands cmd1 and cmd2 must be either bundled together with guaranteed ordering or separated by a barrier, otherwise the switch might re-order them and render the cleanup

operation inefficient

- Why is Algorithm 8 "more efficient alternative to Algorithm 7" ? It looks to be doing much more work

## Review #698C

Modified Sunday 26 Jul 2015 6:26:09am 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 broad problem that this article tackles is to develop abstractions that will allow several independent controllers to update the data-plane of a single switch with almost no inter-controller coordination. This article observes that recent versions of OpenFlow have ingredients that allow us to build a transactional-memory like interface for updating data-plane state. The implementation uses flow tables in a peculiar way. In particular, a couple of flow table entries are used not to process packets
but to maintain a consistent timestamp.

**Detailed comments**

This is a intriguing little article that shows that a recent version of OpenFlow
(which version?!) has the raw ingredients needed to implement a very general
transactional interface for updating flow tables. It is obvious that this is an
accident. The OpenFlow primitives that the article uses actually have a very
narrow interface, but the paper uses them in a clever way to build very general
transactions: a transaction is an atomic sequence of flow-table writes and assertions. i.e., the writes succeed or fail atomically only if all the assertions are true.

This article is primarily a theoretical exercise and doesn't address performance concerns:

- Do real switches have the space to use all these extra rules for coordination?
- Do switches actually implement the features of OpenFlow used by this article
  (i.e., check_overlap, bundles, arbitrary bitmasking of the METADATA field,
  etc.) and do they implement them fast?

Being optimistic, I think that the real value of this article may be to show that
OpenFlow might as well provide a more general transactional interface. Without
any empirical evidence, I doubt that the scheme proposed in the article will
actually work in practice.

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.

Transactions are a powerful primitive, but apart from the simple load balancer
in Algorithm 1, the paper doesn't delve into how applications should be built
to take advantage of the transactional primitive. I think this is what the article sorely lacks. Without it, the article just shows off cool hack.

MINOR

- pg 1 col 1: "perspective of [a] logically centralized"
- pg 1 col 1: "a distributed-systems one: [multiple]"
- pg 1 col 2: perhaps paper -> article
- pg 1 col 2: "We [propose] to implement our"
- pg 3 col 1: "as we will argue, --a-- support for more generic" (delete a)
- pg 3 col 2: "a bundle allows to conditionally"
- In Example 1, why not have the two controllers communicate
- Example 2 says bundling and check-overlap don't help modify. But, Section 2
  says that the semantics of add-flow is to modify if a flow with exactly the
  same priority and match exist

## Comments

Thursday 30 Jul 2015 6:16:57am EDT

Dear authors,

Thank you for your submission. The reviews for this paper raise several important questions and provide good feedback. We hope they will help in improving the paper. Beyond addressing the technical questions and presentation-related issues mentioned in the reviews, a revised version of the paper should include:

-- Examples showing how transactions can be used by control applications. This will make the paper storyline richer.

-- Experiments with actual switches to give readers some confidence that the proposed transactions can be efficiently implemented.

Regards. Area Editor.

## Response

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.

☑ This response should be sent to the reviewers.

**Save**