Ajin's "Empty Transaction" patch V11

- Performance and Traffic Test Results

Background

The purpose of the patch is to reduce the network traffic of the "empty transactions". i.e., those transactions which were only for table not participating in the subscription.

The patch removes the empty transactions for both normal commit and two-phase commit transactions, so both kinds are included in this testing.

Patch discussion/reference

https://www.postgresql.org/message-id/flat/CAFPTHDaQFuASQPjxrYTcRPjF6exewjxXVyfuz1hCWJeCJpOSsQ%40mail.gmail.com#d9dbe3d195f1acccddbd81e46ded2315

Test Overview

Clearly the patch will behave differently for different user scenarios.

- Different amounts of empty transactions
- Different operations performed by the transaction
- Different network speeds
- etc

I have tried to keep the scenarios simple so that a fair comparison can be made. YMMV.

Main Features

- There are 2 similar tables. **One table is published; the other is not.**

- **Equivalent simple SQL operations** are performed on these tables. E.g.
  - INSERT/UPDATE/DELETE using normal COMMIT
  - INSERT/UPDATE/DELETE using 2PC COMMIT PREPARED

- **pg_bench** is used to measure the throughput for **different mixes of empty and not-empty transactions** sent. E.g.
  - 0% are empty
  - 25% are empty
  - 50% are empty
  - 75% are empty
  - 100% are empty

- The **apply_dispatch** code has been temporarily **modified to log the number of protocol messages/bytes being processed**.
  - At the conclusion of the test run the logs are processed to extract the numbers

- Each **test run is 15 minutes elapsed time**.

- The tests are **repeated without and with the patch applied**
  - So, there are 2 (without/with patch) x 5 (different mixes) = 10 test results
  - Transaction throughput results are from pg_bench
  - Protocol message bytes are extracted from the logs (from modified apply_dispatch)

- Also, the entire set of 10 test cases was repeated with **synchronous_standby_names** setting enable/disabled.
  - **Enabled**, so the results are for total round-trip processing of the pub/sub.
  - **Disabled**. no waiting at the publisher side.

Test Details

<table>
<tr><td>postgresql.conf configurations</td><td>PUB-node

wal_level = logical
max_wal_senders = 10
logical_decoding_work_mem = 64kB

checkpoint_timeout = 30min
min_wal_size = 10GB
max_wal_size = 20GB
shared_buffers = 2GB

synchronous_standby_names = 'sync_sub' (for synchronous testing only)</td><td>SUB-node

max_worker_processes = 11
max_logical_replication_workers = 10

checkpoint_timeout = 30min
min_wal_size = 10GB
max_wal_size = 20GB
shared_buffers = 2GB</td></tr>
<tr><td>SQL
test_empty_not_published.sql</td><td colspan="2">-- Operations for table not published
BEGIN;
INSERT INTO test_tab_nopub VALUES(1, 'foo');
UPDATE test_tab_nopub SET b = 'bar' WHERE a = 1;
DELETE FROM test_tab_nopub WHERE a = 1;
COMMIT;

-- 2PC operations for table not published
BEGIN;
INSERT INTO test_tab_nopub VALUES(2, 'fizz');
UPDATE test_tab_nopub SET b = 'bang' WHERE a = 2;
DELETE FROM test_tab_nopub WHERE a = 2;
PREPARE TRANSACTION 'gid_nopub';
COMMIT PREPARED 'gid_nopub';</td></tr>
<tr><td>SQL test_empty_published.sql</td><td colspan="2">Exactly same as above but uses different table (test_tab)</td></tr>
<tr><td rowspan="2">TABLES
(published and not publised)</td><td colspan="2">CREATE TABLE test_tab (a int primary key, b text, c timestamptz DEFAULT now(), d bigint DEFAULT 999);</td></tr>
<tr><td colspan="2">CREATE TABLE test_tab_nopub (a int primary key, b text, c timestamptz DEFAULT now(), d bigint DEFAULT 999);</td></tr>
<tr><td>pg_bench command example using a 25:75 mix of not-published table operations</td><td colspan="2">pgbench -s 100 -T 900 -c 1 -f test_empty_not_published.sql@5 -f test_empty_published.sql@15 test_pub</td></tr>
</table>

Result Data

Raw data.

Synchronous mode

	Mixture		Transactions								apply_dispatch					
Patch	**NoPub**	**Pub**	**TxNoPub**	**TxPub**	**TxTotal**	**Gain**	**TpsNoPub**	**Gain**	**TpsPub**	**Gain**	**msgs**	**bytes**	**msgs/tx**	**Gain**	**bytes/tx**	**Gain**
No	0%	100%	0	1562	1562		0.00		1.73555		15621	992815	10.0		635.6	
No	25%	75%	526	1589	2115		0.584116		1.7646		17998	1112213	8.5		525.9	
No	50%	50%	1637	1677	3314		1.818577		1.863		23329	1384028	7.0		417.6	
No	75%	25%	5002	1672	6674		5.555316		1.857		36734	2033539	5.5		304.7	
No	100%	0%	116221	0	116221		129.13457		0.00		464892	22547262	4.0		194.0	
Yes	0%	100%	0	1579	1579	1.09%	0.00	0.00%	1.754	1.06%	15792	1003818	10.0	0.01%	635.7	0.02%
Yes	25%	75%	526	1626	2152	1.75%	0.584206	0.02%	1.8059	2.34%	16262	1033620	7.6	-11.20%	480.3	-8.66%
Yes	50%	50%	1703	1661	3364	1.51%	1.891083	3.99%	1.8444	-1.00%	16614	1055946	4.9	-29.84%	313.9	-24.84%
Yes	75%	25%	4999	1682	6681	0.10%	5.550849	-0.08%	1.8677	0.58%	16828	1069734	2.5	-54.24%	160.1	-47.45%
Yes	100%	0%	124621	0	124621	7.23%	138.46789	7.23%	0.00	0.00%	0	0	0.0	-100.00%	0.0	-100.00%
						2.34%		2.23%		0.60%						

NOT Synchronous mode

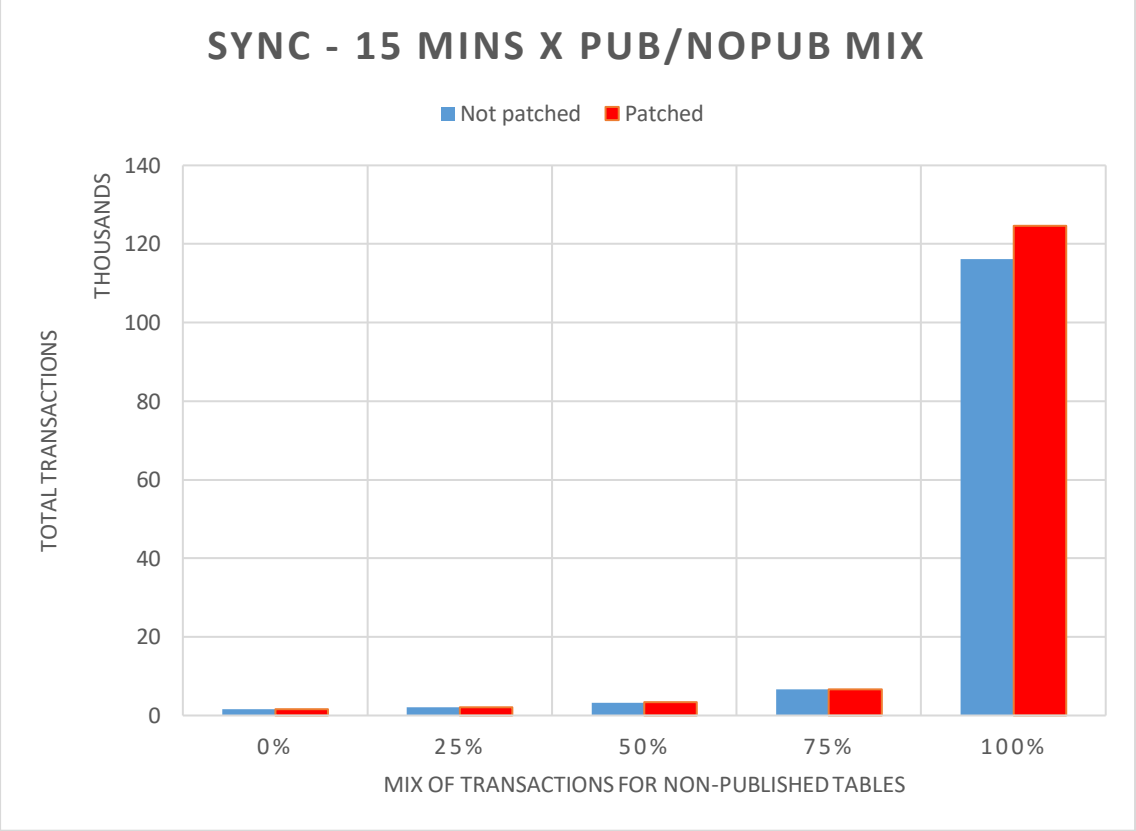
	Mixture		Transactions								apply_dispatch					
Patch	**NoPub**	**Pub**	**TxNoPub**	**TxPub**	**TxTotal**	**Gain**	**TpsNoPub**	**Gain**	**TpsPub**	**Gain**	**msgs**	**bytes**	**msgs/tx**	**Gain**	**bytes/tx**	**Gain**
No	0%	100%	0	138545	138545		0.00		153.93926		1385495	88056849	10.0		635.6	
No	25%	75%	33911	101041	134952		37.67873		112.2673		1146112	70799881	8.5		524.6	
No	50%	50%	67845	67896	135741		75.38358		75.44024		950394	56317227	7.0		414.9	
No	75%	25%	108557	36027	144584		120.619		40.03005		794543	43959927	5.5		304.0	
No	100%	0%	148945	0	148945		165.49503		0.00		1165522	11734575	7.8		78.8	
Yes	0%	100%	0	142143	142143	2.60%	0.00	0.00%	157.9368	2.60%	1421468	90342786	10.0	0.00%	635.6	0.00%
Yes	25%	75%	36091	108277	144368	6.98%	40.10088	6.43%	120.3071	7.16%	1082797	68818487	7.5	-11.69%	476.7	-9.14%
Yes	50%	50%	70398	70842	141240	4.05%	78.22022	3.76%	78.71356	4.34%	708448	45027264	5.0	-28.36%	318.8	-23.16%
Yes	75%	25%	109355	36209	145564	0.68%	121.5059	0.74%	40.23234	0.51%	362104	23013958	2.5	-54.73%	158.1	-48.00%
Yes	100%	0%	153738	0	153738	3.22%	170.8199	3.22%	0.00	0.00%	0	0	0.0	-100.00%	0.0	-100.00%
						3.50%		2.83%		2.92%						

Observations (Synchronous mode)

Total Transactions

- As the percentage mix of empty transactions increases, so does the transaction throughput. I assume this is because we are using *synchronous* mode; so when there is less waiting time, then there is more time available for transaction processing

- The performance was generally similar before/after the patch, but **there was an observed throughput improvement of ~2%** (averaged across the all mixes)

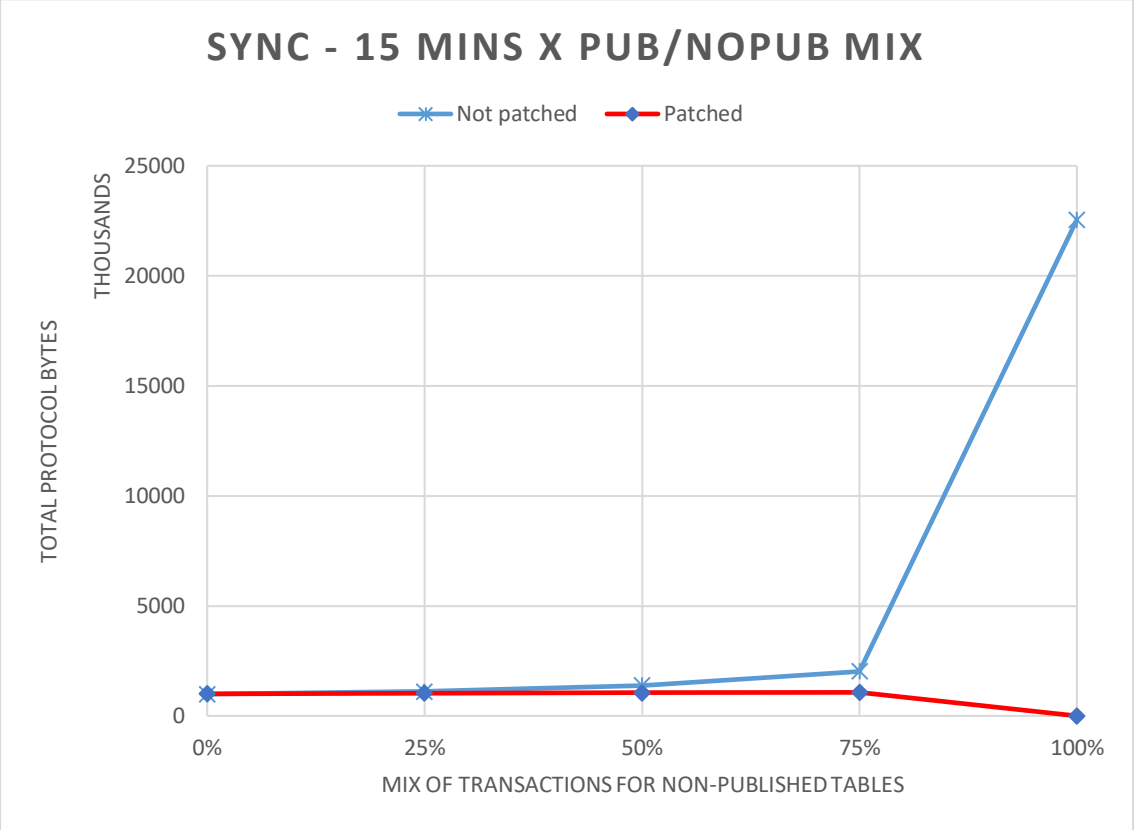


Protocol (Network) Bytes

- The number of protocol bytes is associated with the number of transactions that are processed during the test time of 15 minutes. This adds up to a significant number of bytes even when the transactions are empty.

- For the **unpatched** code as the transaction rate increases, then so does the amount of traffic bytes.

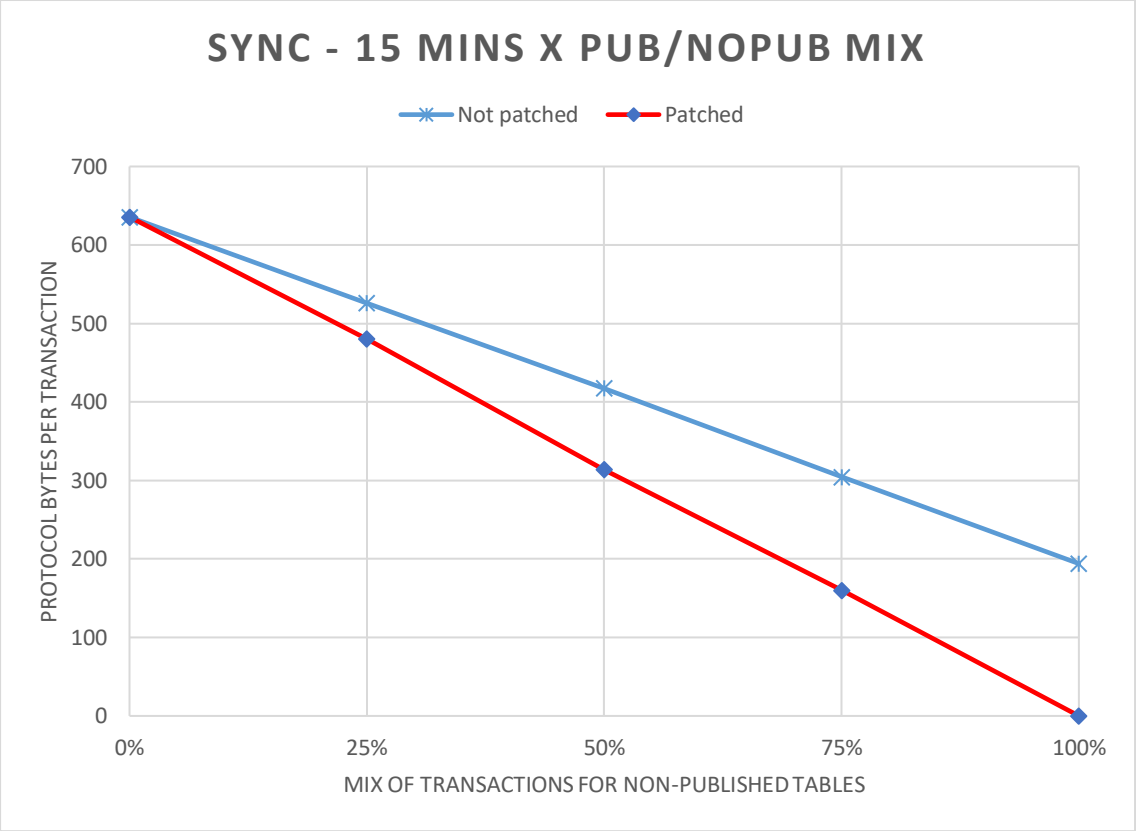
- **The patch improves this significantly by eliminating all the empty transaction traffic**.



Protocol (Network) Bytes / Transaction

- The reduction in traffic can also be shown (below) by plotting the average bytes processed by **apply_dispatch** *per transaction*. Of course, the number of bytes/transaction decreases as there are more empty transactions in the mixture.

- Before the patch, even "empty transactions" are processing some bytes, so it can never reach zero. **After the patch, empty transaction traffic is eliminated entirely.**



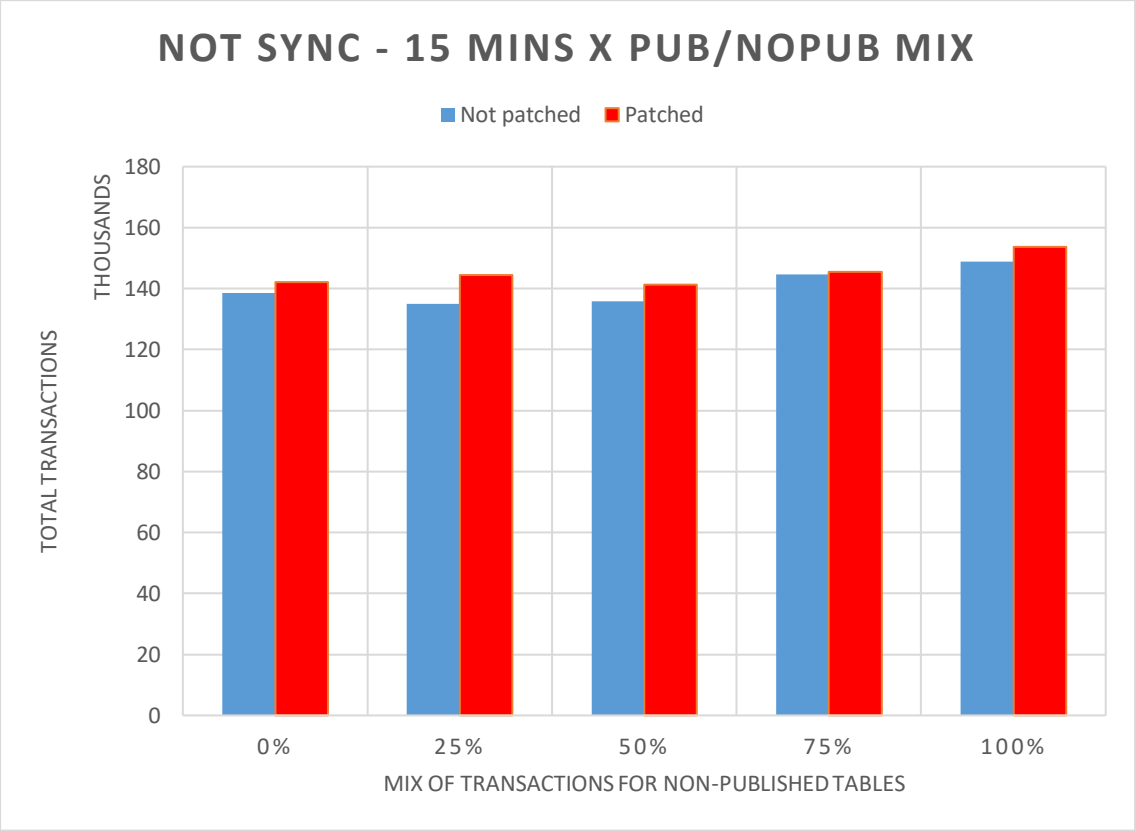
Observations (NOT Synchronous mode)

Total Transactions

- Since there is no synchronous waiting for round trips, the **transaction throughput is generally consistent** regardless of the empty transaction mix.

- There is a hint of a small overall improvement in throughput as the empty transaction mix approaches near 100%. For my test environment both the pub/sub nodes are using the same machine/CPU, so I guess is that when there is less CPU spent processing messages in the Apply Worker then there is more CPU available to pump transactions at the publisher side.

- **The patch transaction throughput seems ~3% better than for non-patched**. This might also be attributable to the same reason mentioned above - less CPU spent processing empty messages at the subscriber side leaves more CPU available to pump transactions from the publisher side.

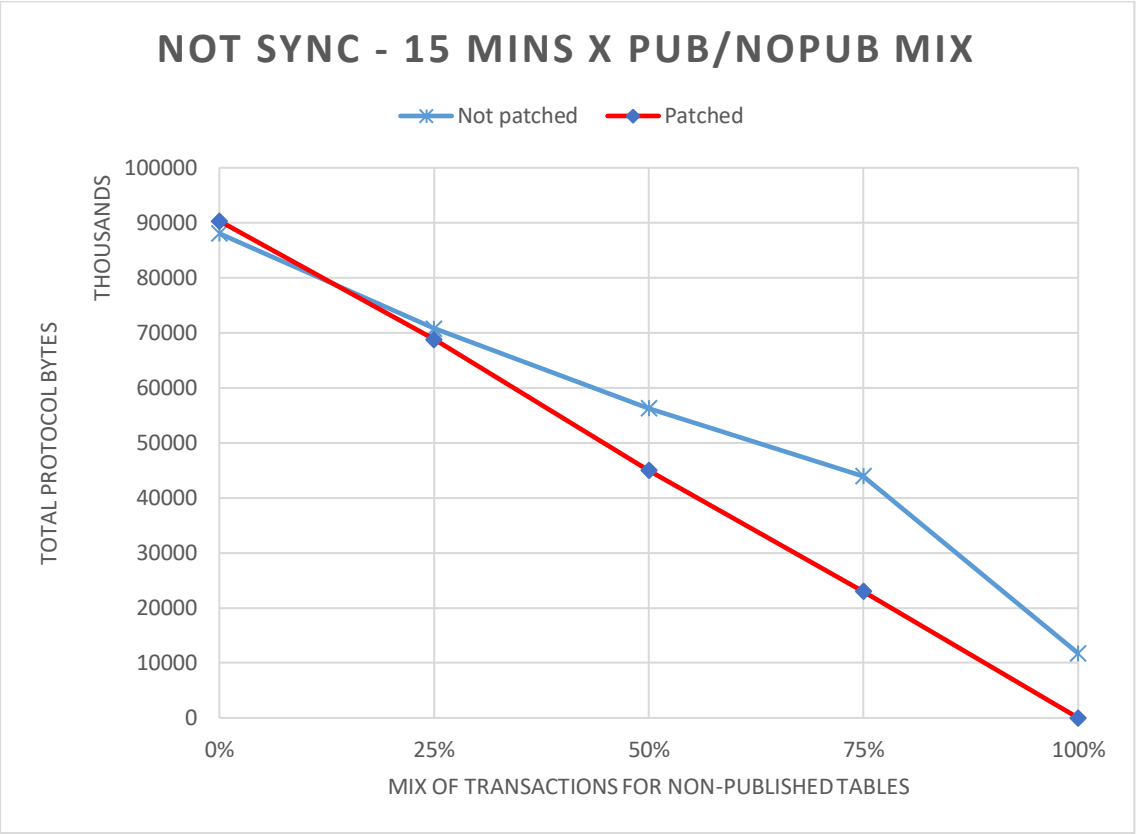


Protocol (Network) Bytes

- The number of protocol bytes is associated with the number of transactions that are processed during the test time of 15 minutes.

- Because the transaction throughput is consistent, the traffic of protocol bytes here is determined mainly by the proportion of “empty transactions” in the mixture.

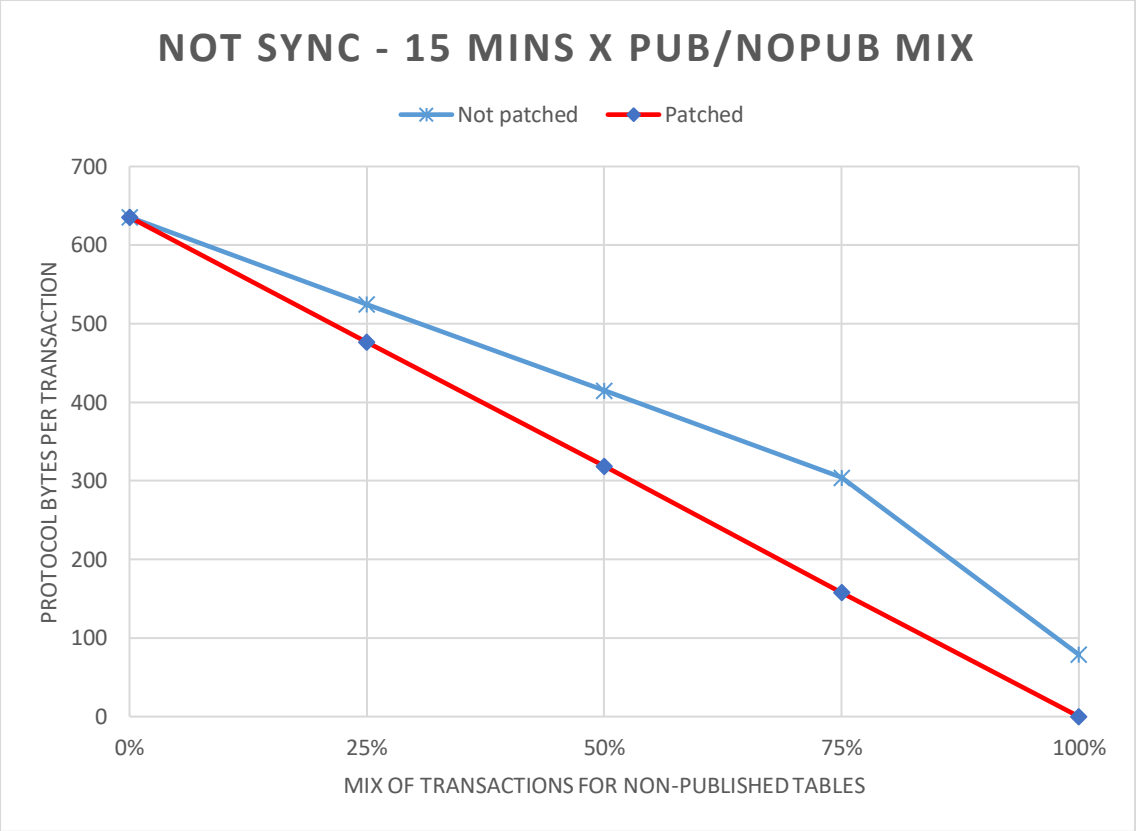
- Before the patch, even “empty transactions” are processing some bytes, so it can never reach zero. **After the patch, the empty transaction traffic is eliminated entirely.**



Protocol (Network) Bytes / Transaction

- This is almost same as the Synchronous graph of bytes/transaction – and this is as expected because the kinds of transactions (and therefore the numbers of bytes per transaction) is same for both the Synchronous/NOT Synchronous testing

- Before the patch, even "empty transactions" are processing some bytes, so it can never reach zero. **After the patch, the empty transaction traffic is eliminated entirely.**



Conclusion

The results clearly show protocol traffic is greatly reduced by this patch.

In practice, how much this reduced traffic will impact actual transaction throughput will depend on many factors (e.g., what was the mix of empty transactions; how much table data is getting replicated; what is the speed of the network etc.).

In my test environment the general observations of the patch are:

- There is a potentially very large reduction of network traffic
- Transaction throughput improved ~2% (average across mixtures) for Synchronous mode
- Transaction throughput improved ~3% (average across mixtures) for NOT Synchronous mode

YMMV.

[END]