# Latency Patterns in Infiniband

Alex Mordkovich

# Agenda

Motivation

RAMCloud ping-pong

Simple ping-pong client/server

    Detour: Infiniband architecture

Effect of Infiniband parameters

Impact on RAMCloud

# Motivation

While profiling RAMCloud's RPC performance, observed alternating WRITE latencies in ClusterPerf test.

Stripped away RPC layer in RAMCloud and other variables to arrive at bare-bones transport-level "ping-pong".

# RAMCloud ping-pong

1 client, 1 master, 0 backups, log cleaner disabled

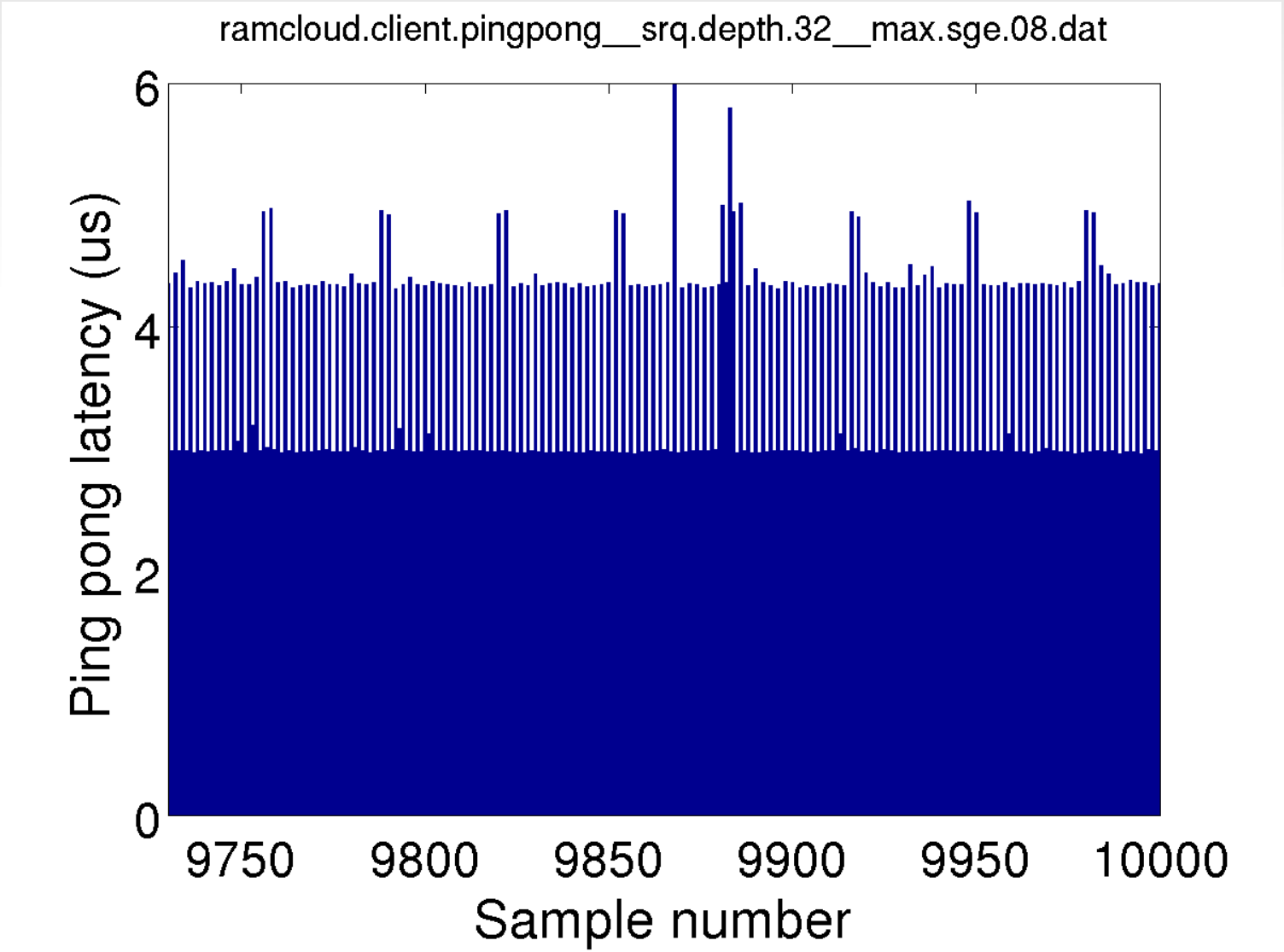
Intercept client/master communication at Infiniband transport layer.

Bypassing RPC layer, ping-pong 10,000 128-byte messages between client and master.

Ping-pong'ing is single-threaded (no workers; everything done in polling thread).

# RAMCloud ping-pong: Client

Default configuration



# RAMCloud ping-pong: Client

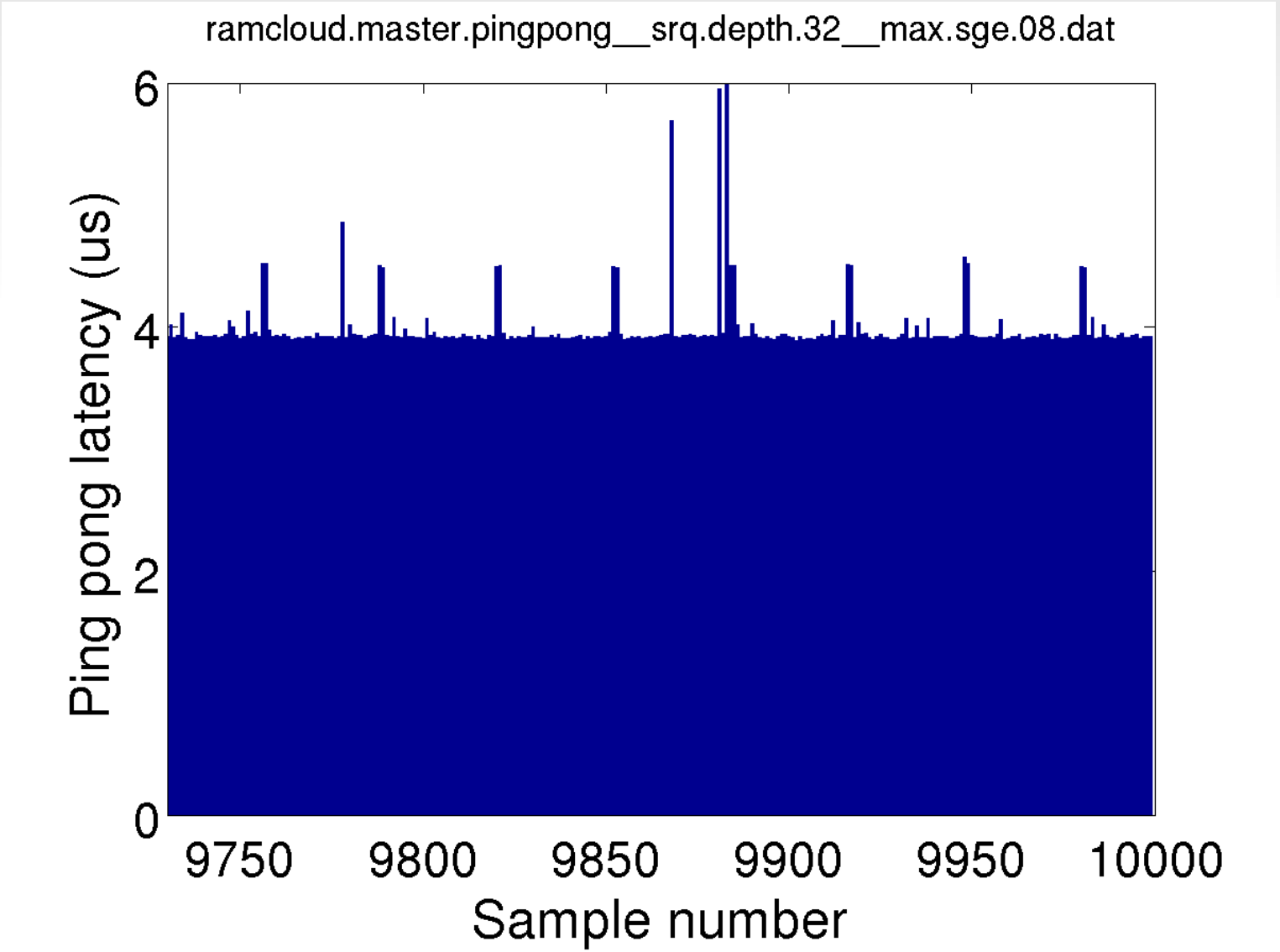
Notable features:

- Base latency just under 3.0us
- Every other round-trip is 4.4us
- There is a 32-round-trip cycle
  - Samples #30 and #32 are just under 5.0us

- Noticeable degree of random noise

# RAMCloud ping-pong: Master

Default configuration

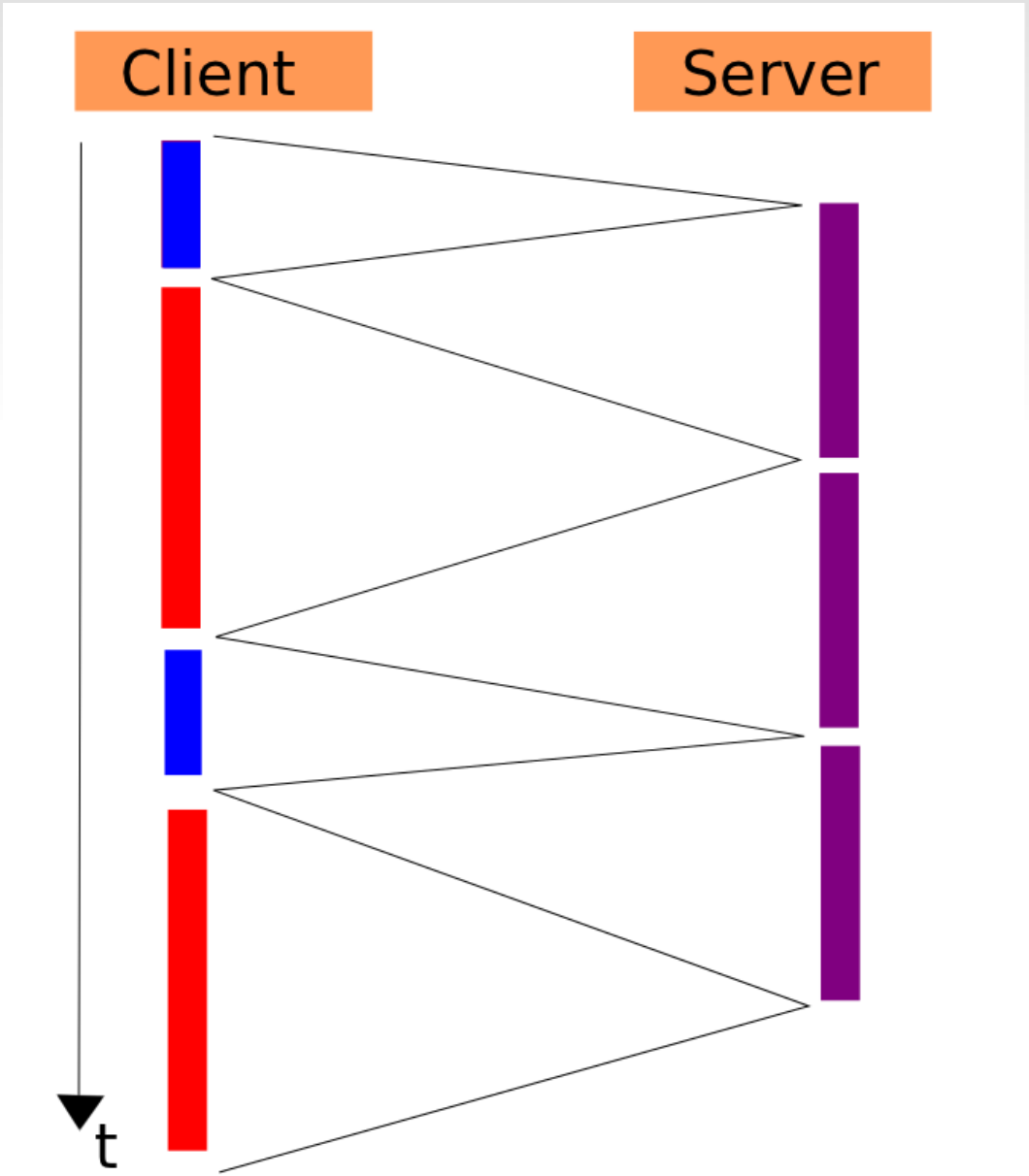


# RAMCloud ping-pong: Master

Notable features:

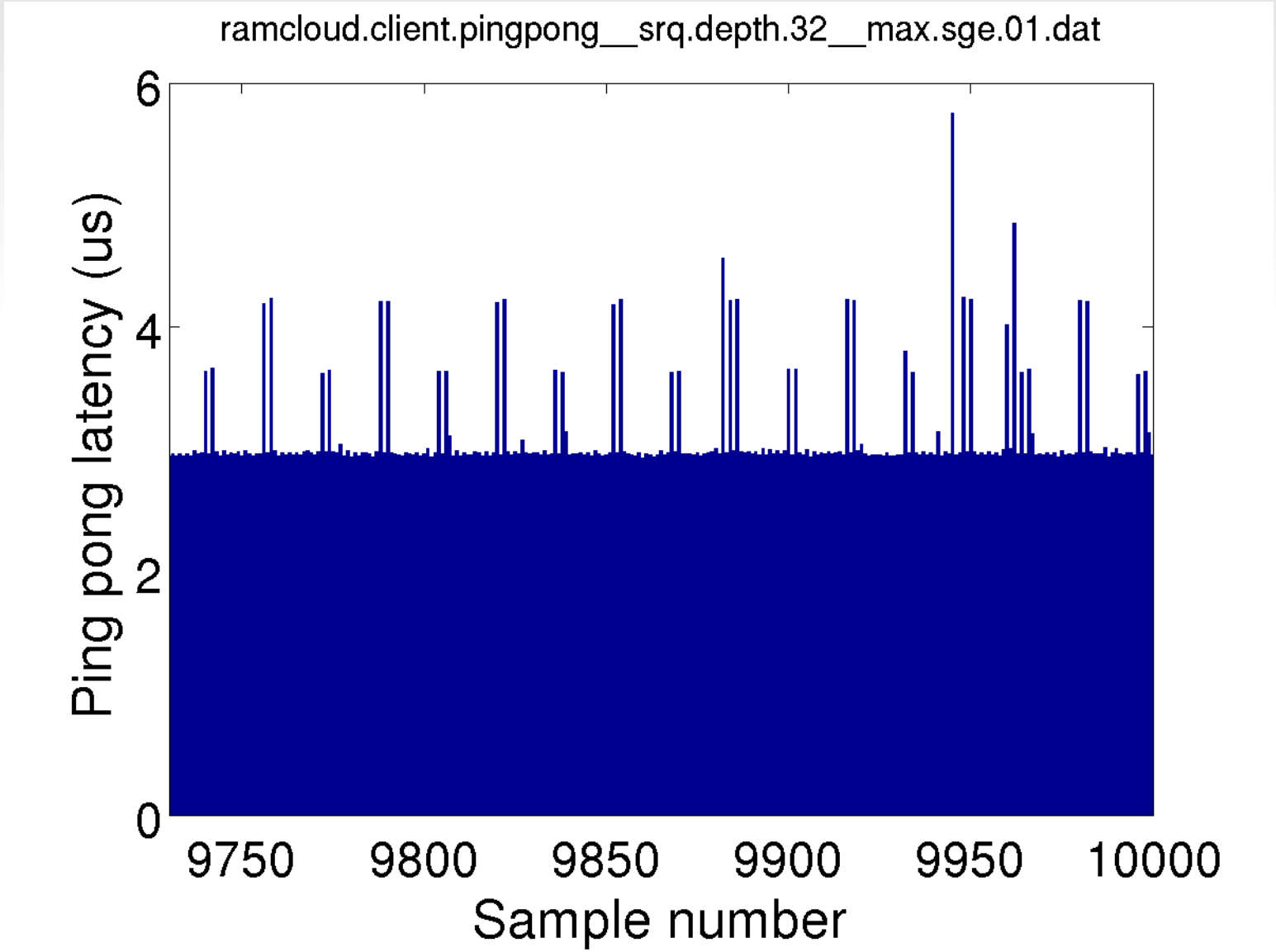
- Base latency about 3.9us
- No obvious alternation
- There is a 32-round-trip cycle
  - Samples #31 and #32 are 4.5us
- Noticeable degree of random noise

# Results on RAMCloud: Reconciling perspectives



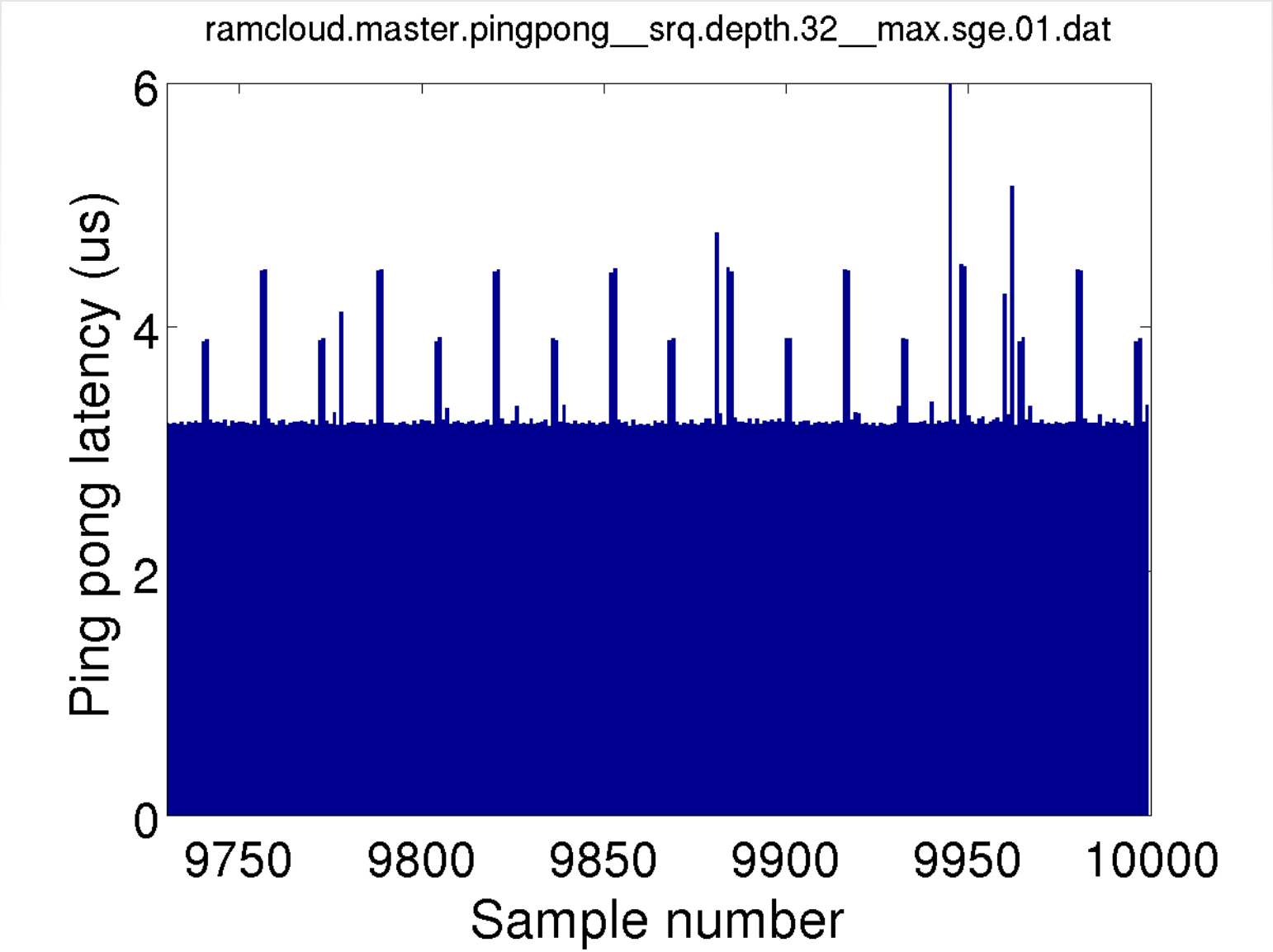
# RAMCloud ping-pong: Client

New configuration



# RAMCloud ping-pong: Master

New configuration



# Simple ping-pong client/server

Simple client (single-threaded):

- Send 128-byte message to server
- Wait for a response from server
- Repeat 10,000 times

Simple server (single-threaded):

- Wait for message from client
- Send 128-byte response to client
- Repeat

# Infiniband Architecture

Each endpoint has a *queue pair* consisting of

- Send queue
- Receive queue

Each endpoint also has a *completion queue* for notification of completed work requests.

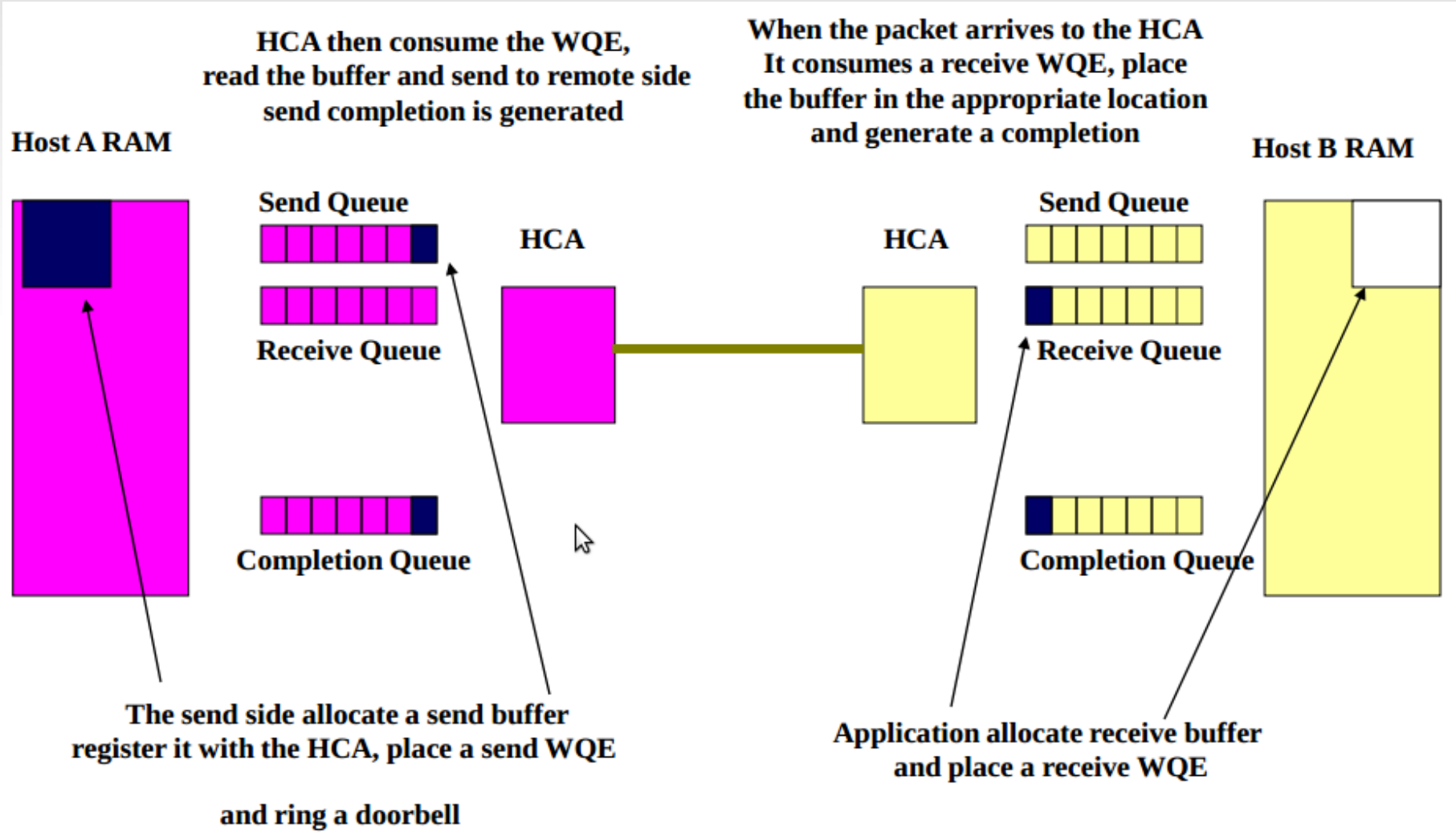
# Infiniband Architecture

To send a message, post a *send work request* on the send queue.

To receive a message, post a *receive work request* on the receive queue.

When a send or a receive request completes, a notification is posted on the host's *completion queue*.

# Sending and Receiving




[ Diagram courtesy of http://www.hpcadvisorycouncil.com/events/switzerland_workshop/pdf/Presentations/Day%201/9_InfiniBand%20Arch.pdf ]

# The Shared Receive Queue

Multiple queue pairs on a given host can share a *shared receive queue*.

The SRQ is used instead of the receive queue in the queue pair.

To send, each QP's own send queue is still used.

# Simple ping-pong setup

Simple client and server set up queue pairs using the RDMA Connection Manager library (`librdmacm`).

Client's and server's receive queues pre-populated with some number of receive work requests (*N*).

When message is received, corresponding buffer is returned to the tail of the receive queue.

Message reception cycles through N receive buffers.

# Effect of Infiniband parameters

Some of the settings tested:

- Receive queue depth (N)
- Using the SRQ versus the default receive queue in QP
- Granularity of receive buffer registration
- Maximum number of SGE in an SRQ request

# Effect of receive queue depth

N = 1

Using receive queue in QP

- Two clear modes: 17us and 12.5us
- No obvious pattern between the two modes

Using SRQ

- Almost all round trips take 5.4us
- Some noise, but no obvious pattern

# Effect of receive queue depth

N = 16

Using receive queue in QP

- Every 14th, 15th, and 16th round-trip is about 7.2us
- Remaining round-trips in the cycle are 3.15us.

Using SRQ

- Every 15th and 16th round-trip is 4.4us
- Remaining round-trips in the cycle are about 3.17us.

# Effect of receive queue depth

N = 32

Using receive queue in QP

- Every 30th, 31st, and 32nd round-trip is about 7.4us
- Remaining round-trips in the cycle are 3.15us.

Using SRQ (*RAMCloud configuration*)

- Every 31st and 32nd round-trip is 4.4us
- Every 15th and 16th round-trip is 3.85us
- Remaining round-trips in the cycle are about 3.15us.

# Effect of receive queue depth

N = 10,000 (receive queues fully pre-populated)

Using receive queue in QP

- Every 31st and 32nd round-trip is 3.8us
- Every 127th and 128 thround-trip is 4.5us
- Every 255th and 256th round-trip is 5.5us.
- Remaining round-trips in the cycle are about 3.15us.

Using SRQ

- Every 15th and 16th round-trip is 3.8us.
- Every 127th and 128th round-trip is 4.5us.
- Remaining round-trips in the cycle are about 3.15us

# Granularity of buffer registration

Receive (and transmit) buffers must be registered with HCA.

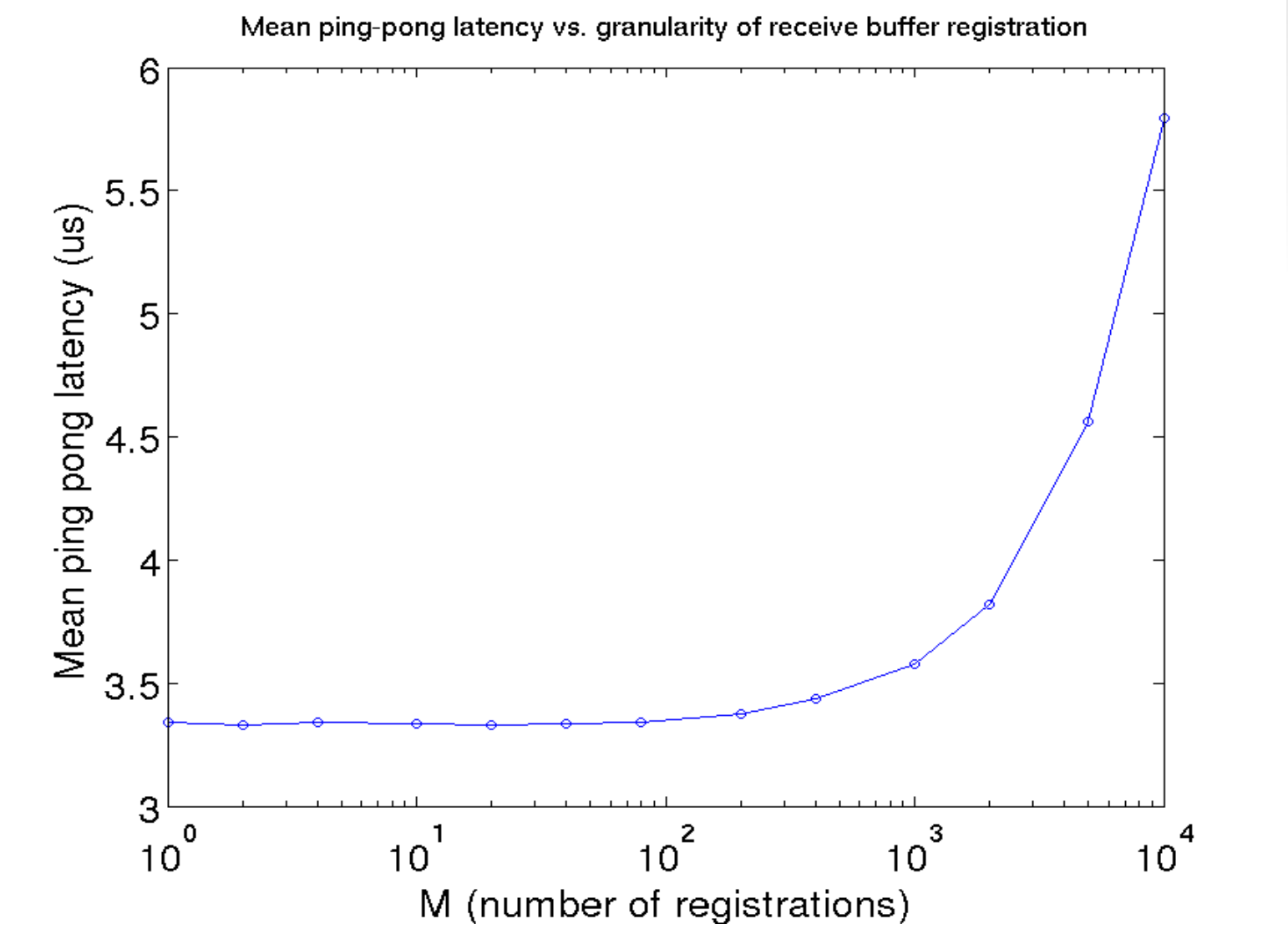
If we want N receive buffers of fixed size, we can:

- Register 1 contiguous chunk of memory and split it into N individual buffers.
- Register 2 contiguous chunks of memory and split each into N/2 individual buffers.
- …
- Register M contiguous chunks of memory and split each into N/M individual buffers.

*Previous results use M=1 (as does RAMCloud).*

# Granularity of buffer registration

N = 10,000, Using SRQ



# Maximum SGE in an SRQ request

A receive work request can specify multiple scatter/gather entries.

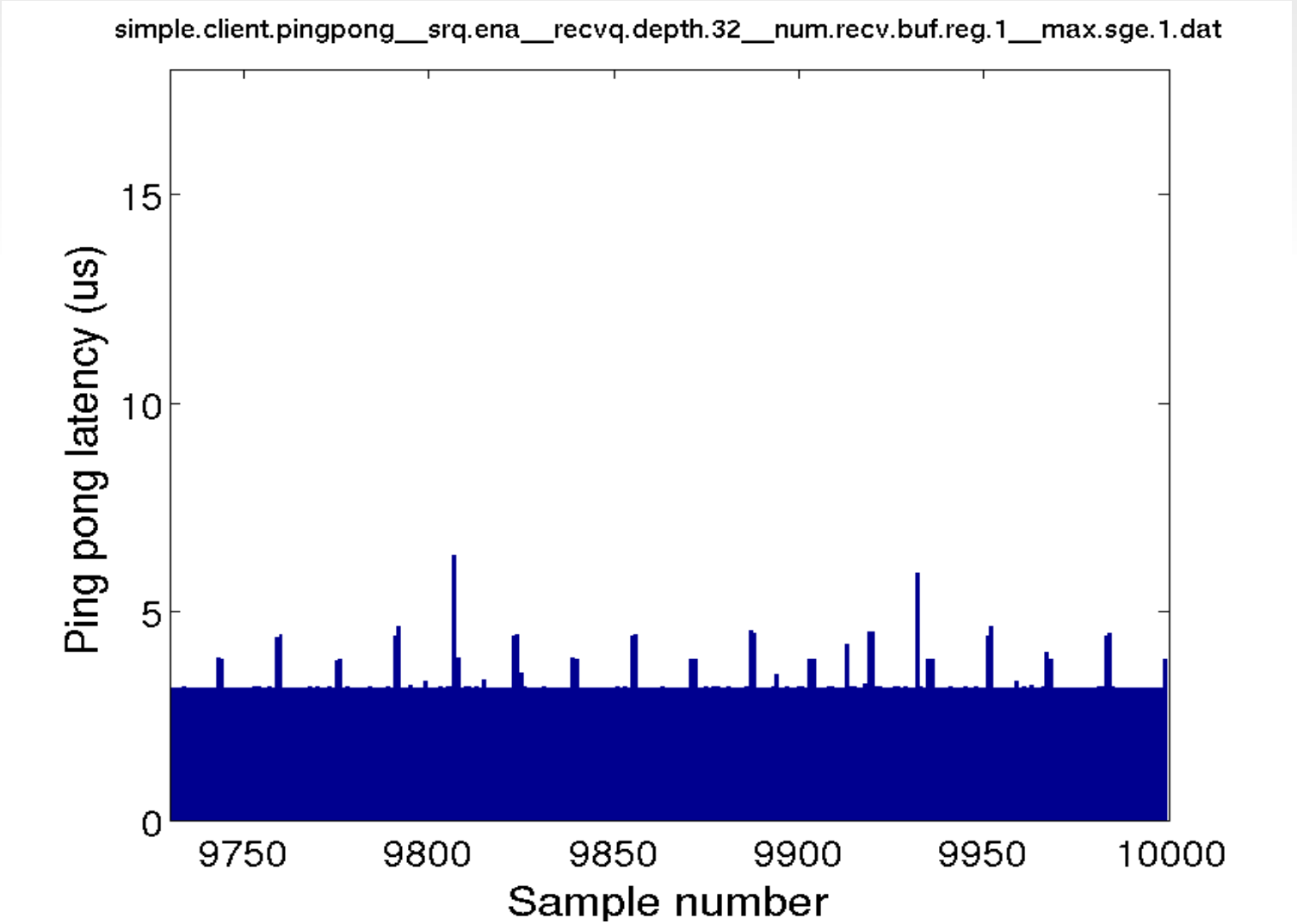
When an SRQ is created, the *maximum* number of SGE entries per (future) request is specified.

Both RAMCloud and simple program only actually use one SGE per receive request.

*Previous results use value* max_sge = 1

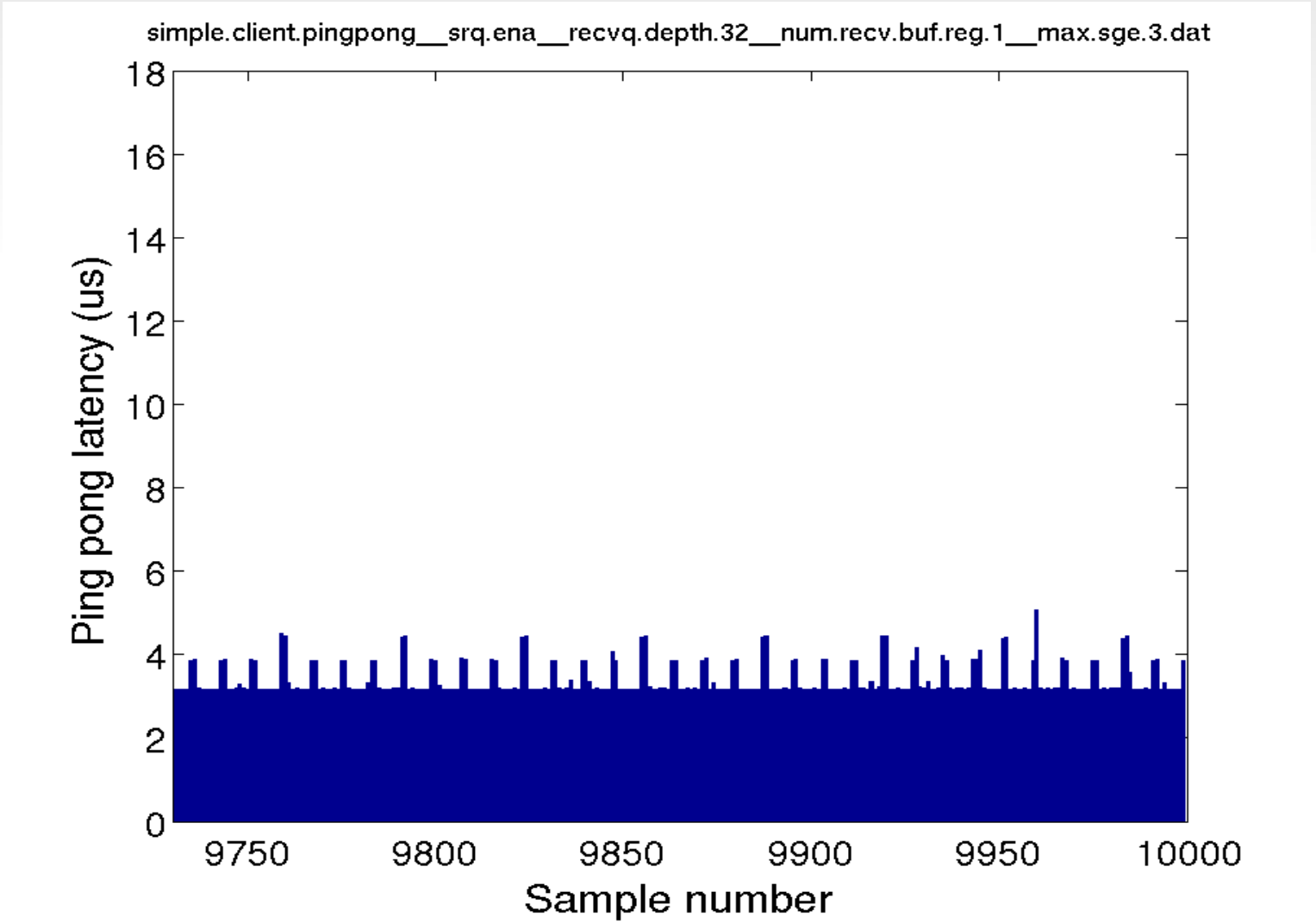
# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ max_sge = 1



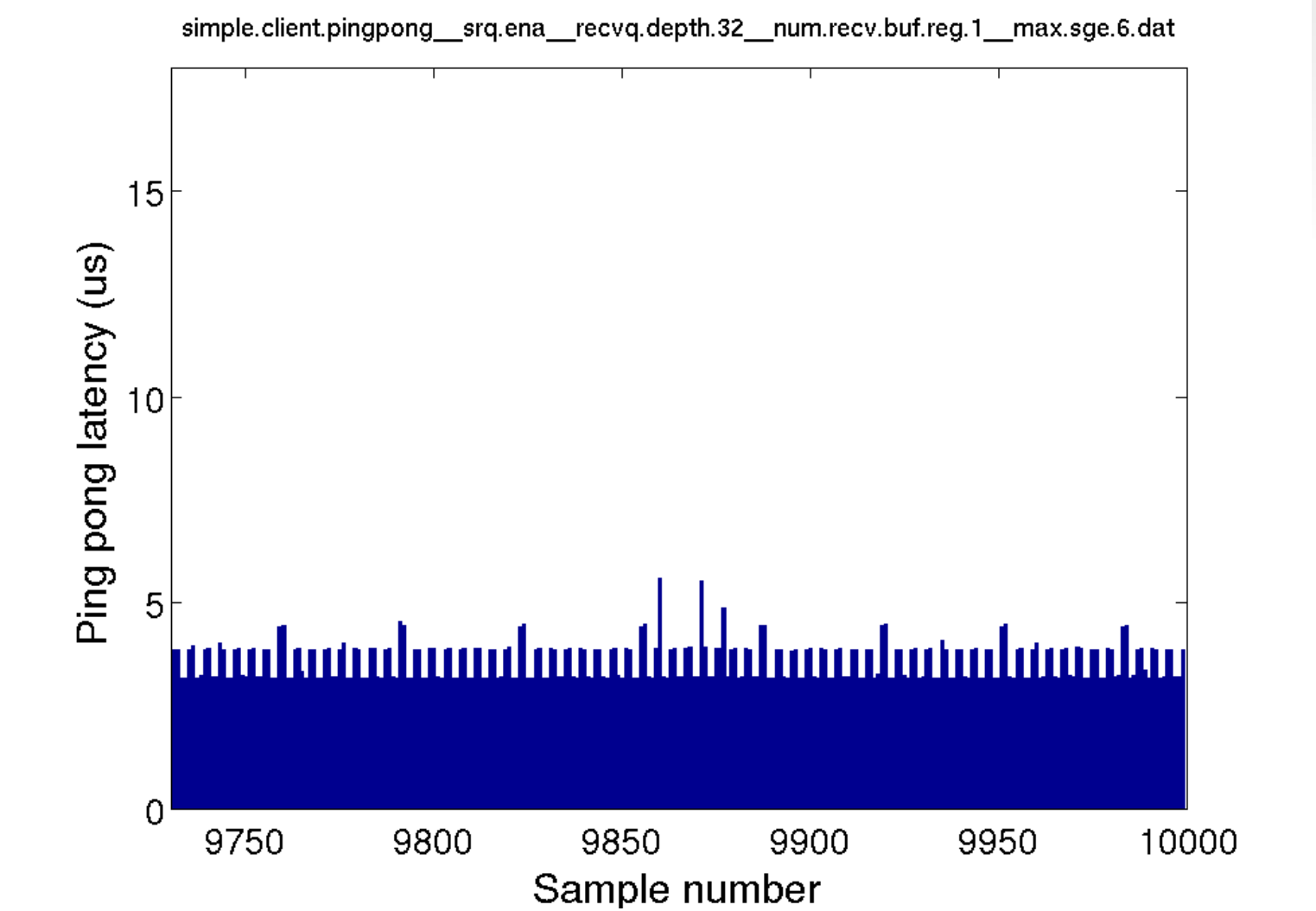
# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ max_sge = 2 thru 3



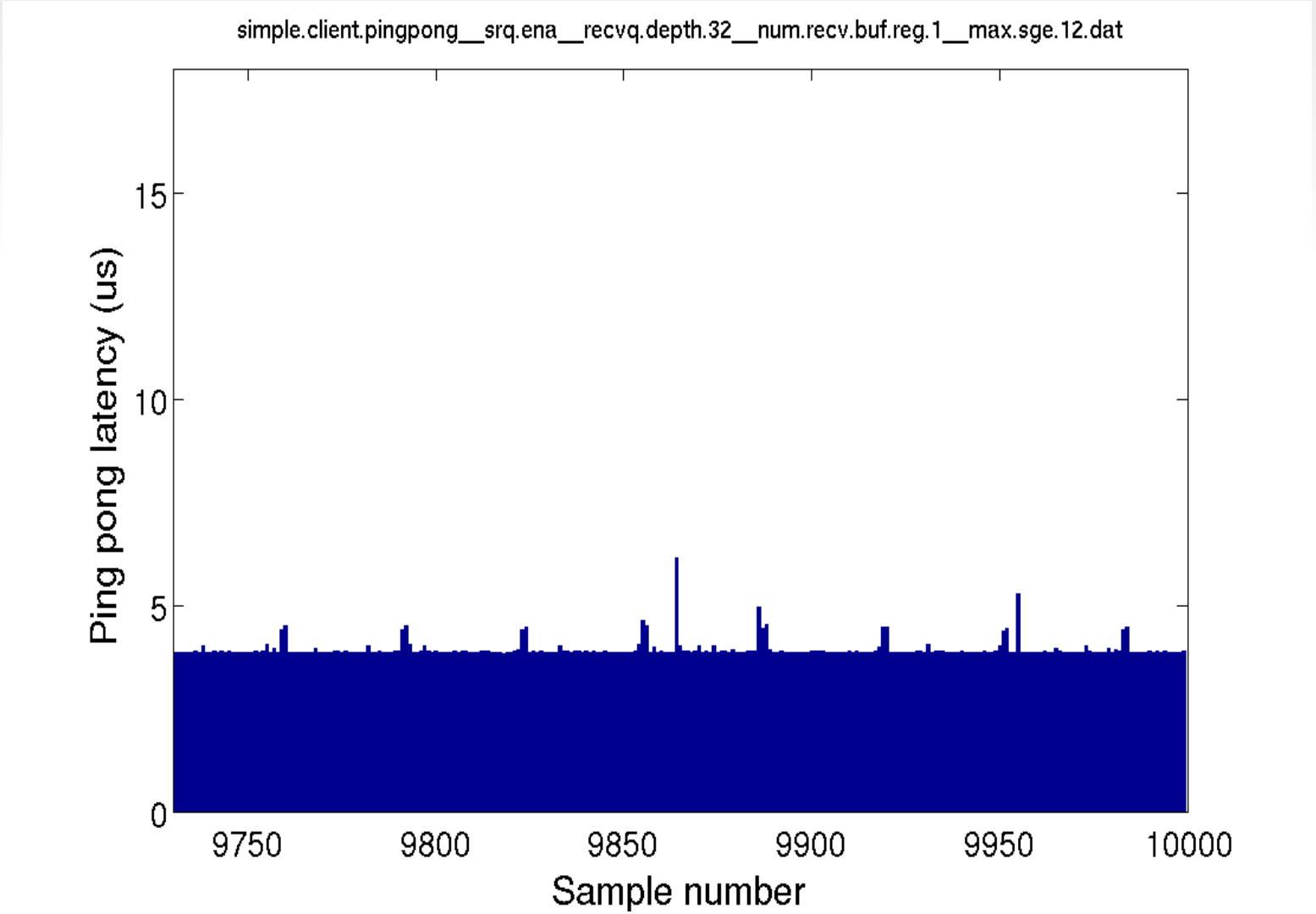
# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ max_sge = 4 thru 7



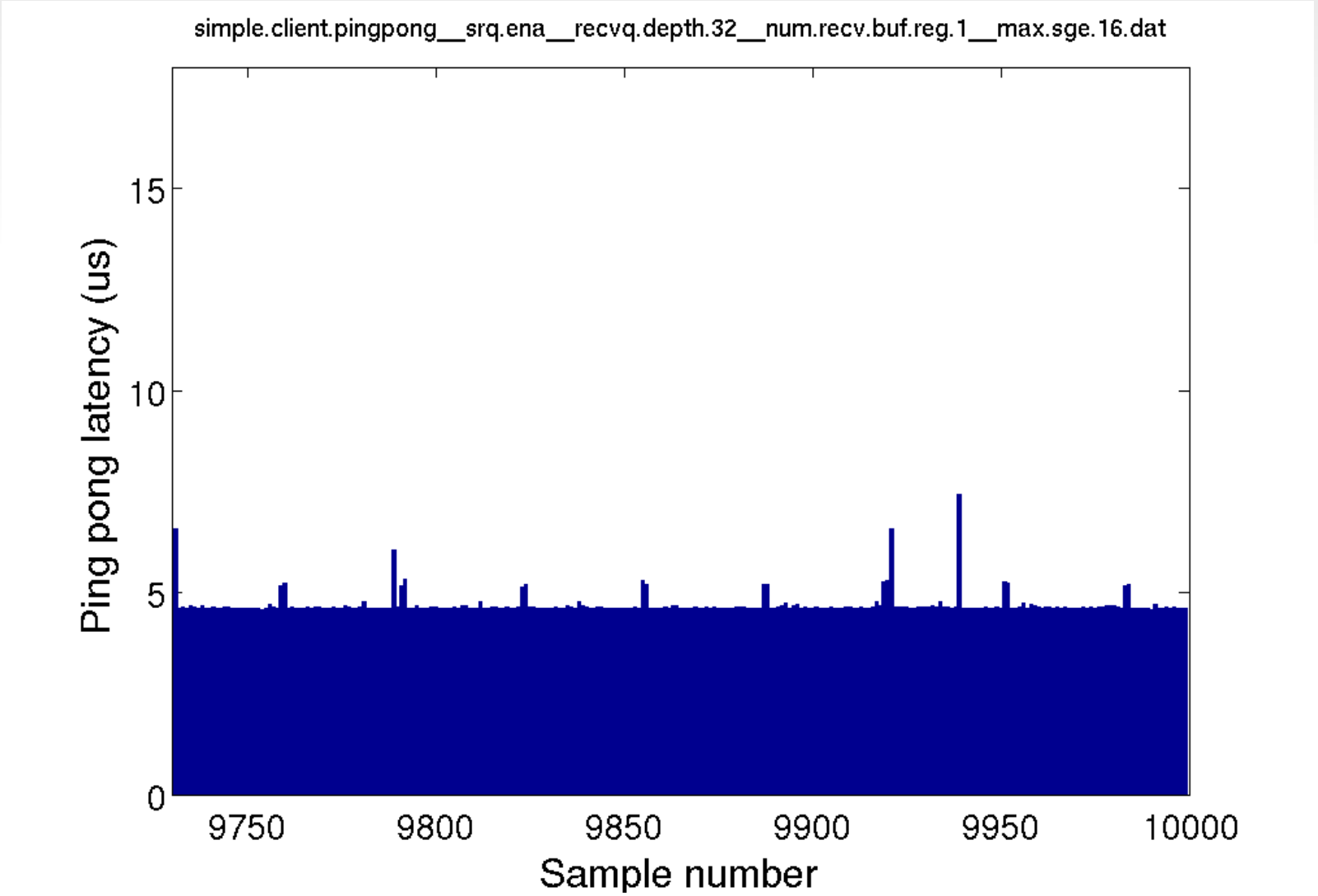
# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ max_sge = 8 thru 15



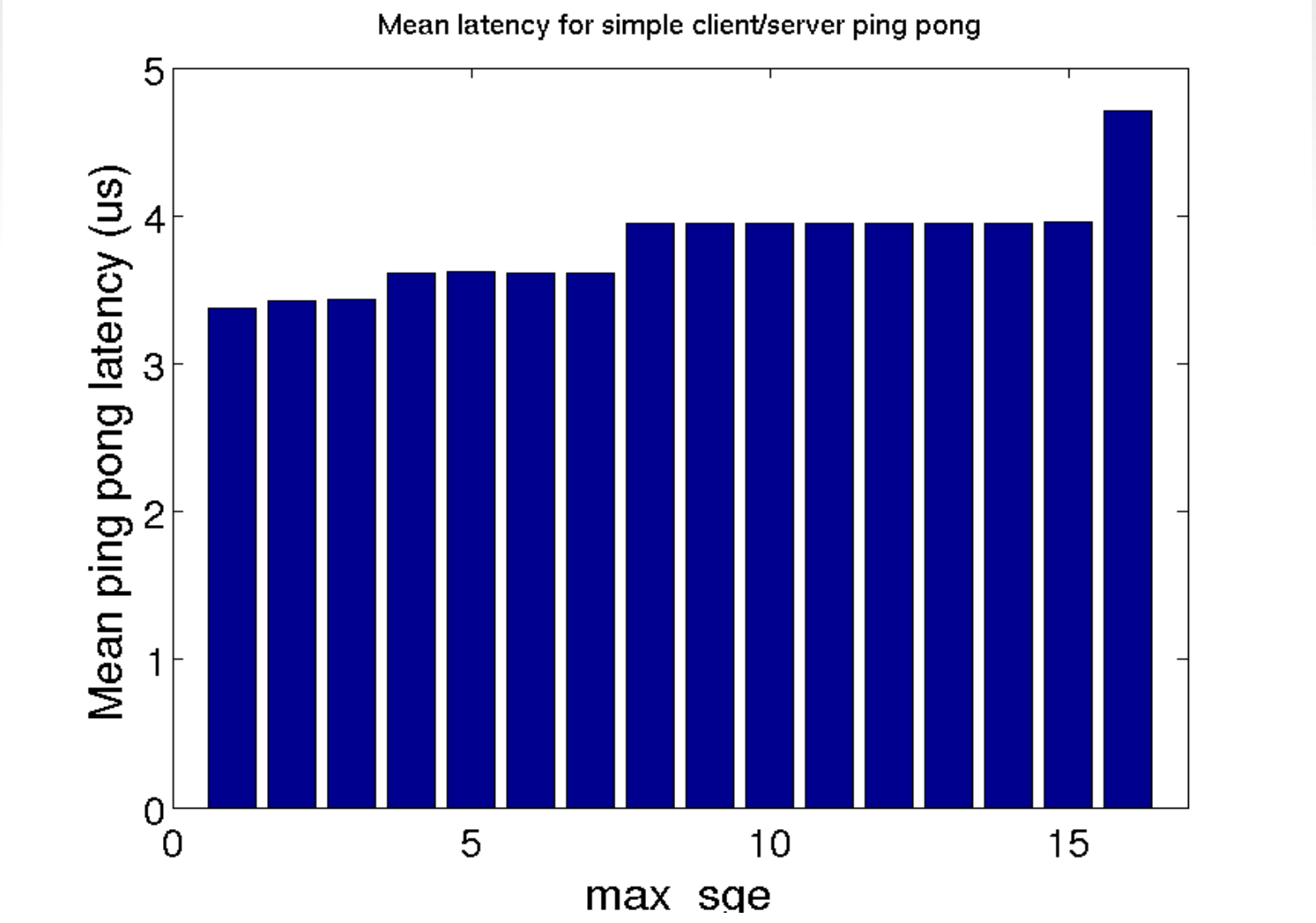
# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ max_sge = 16



# Maximum SGE in an SRQ request

N = 32, M = 1, using SRQ



# Returning receive buffers to SRQ

Individually or in bulk

- Return the receive buffer to SRQ as soon as a message is received, or
- Return all N receive buffers to SRQ when the queue is empty.

Returning in bulk has adverse effect on Nth round-trip.

Returning individually is cheap and has minimal impact on latency.

# Returning receive buffers to SRQ

Before or after sending the response

- Return receive buffer to SRQ, then send response.
- Send the response, then return the receive buffer to SRQ.

Returning receive buffer *after* sending response slightly improves latency, but increases variance.

# Timing

Tested whether latency patterns are time-dependent

Spin client CPU for about 20us between receiving a response and sending the next message.

No effect on round-trip latency patterns.

# More quirks

While running test on RAMCloud, accidentally forgot to return a receive buffer to the SRQ.

RAMCloud ping-pong test thus had 31, rather than 32, receive buffers to work with.

Surprising results.

# Impact on RAMCloud

## ClusterPerf *basic* test (0 replicas)*

| | max_sge = 8 | max_sge = 1 |
|---|---|---|
| basic.read100 | 5.6 us | 5.0 us |
| basic.readBw100 | 16.9 MB/s | 19.1 MB/s |
| basic.read1K | 7.2 us | 6.7 us |
| basic.readBw1K | 132.2 MB/s | 142.9 MB/s |
| basic.read10K | 10.4 us | 9.9 us |
| basic.readBw10K | 918.0 MB/s | 963.4 MB/s |
| basic.read100K | 46.5 us | 46.1 us |
| basic.readBw100K | 2.0 GB/s | 2.0 GB/s |
| basic.read1M | 430.2 us | 430.2 us |
| basic.readBw1M | 2.2 GB/s | 2.2 GB/s |
| basic.write100 | 6.5 us | 5.8 us |
| basic.writeBw100 | 14.7 MB/s | 16.4 MB/s |
| basic.write1K | 8.6 us | 7.9 us |
| basic.writeBw1K | 111.5 MB/s | 120.2 MB/s |
| basic.write10K | 15.1 us | 14.6 us |
| basic.writeBw10K | 630.0 MB/s | 654.7 MB/s |
| basic.write100K | 98.2 us | 97.5 us |
| basic.writeBw100K | 970.8 MB/s | 978.2 MB/s |
| basic.write1M | 987.8 us | 981.9 us |
| basic.writeBw1M | 965.5 MB/s | 971.2 MB/s |

* For more accurate results, modified ClusterPerf to run 1000 iterations of each test rather than run each test for a fixed amount of time.

# Impact on RAMCloud

## ClusterPerf *basic* test (3 replicas)*

| | max_sge = 8 | max_sge = 1 |
|---|---|---|
| basic.read100 | 5.5 us | 4.9 us |
| basic.readBw100 | 17.4 MB/s | 19.4 MB/s |
| basic.read1K | 7.1 us | 6.6 us |
| basic.readBw1K | 133.5 MB/s | 145.3 MB/s |
| basic.read10K | 10.4 us | 9.9 us |
| basic.readBw10K | 920.3 MB/s | 967.8 MB/s |
| basic.read100K | 46.7 us | 46.2 us |
| basic.readBw100K | 2.0 GB/s | 2.0 GB/s |
| basic.read1M | 429.8 us | 439.8 us |
| basic.readBw1M | 2.2 GB/s | 2.1 GB/s |
| basic.write100 | 14.9 us | 14.0 us |
| basic.writeBw100 | 6.4 MB/s | 6.8 MB/s |
| basic.write1K | 18.9 us | 17.9 us |
| basic.writeBw1K | 50.6 MB/s | 53.3 MB/s |
| basic.write10K | 37.4 us | 36.9 us |
| basic.writeBw10K | 254.8 MB/s | 258.3 MB/s |
| basic.write100K | 244.2 us | 244.2 us |
| basic.writeBw100K | 390.6 MB/s | 390.5 MB/s |
| basic.write1M | 2.4 ms | 2.3 ms |
| basic.writeBw1M | 404.9 MB/s | 409.7 MB/s |

* For more accurate results, modified ClusterPerf to run 1000 iterations of each test rather than run each test for a fixed amount of time.

# Q & A