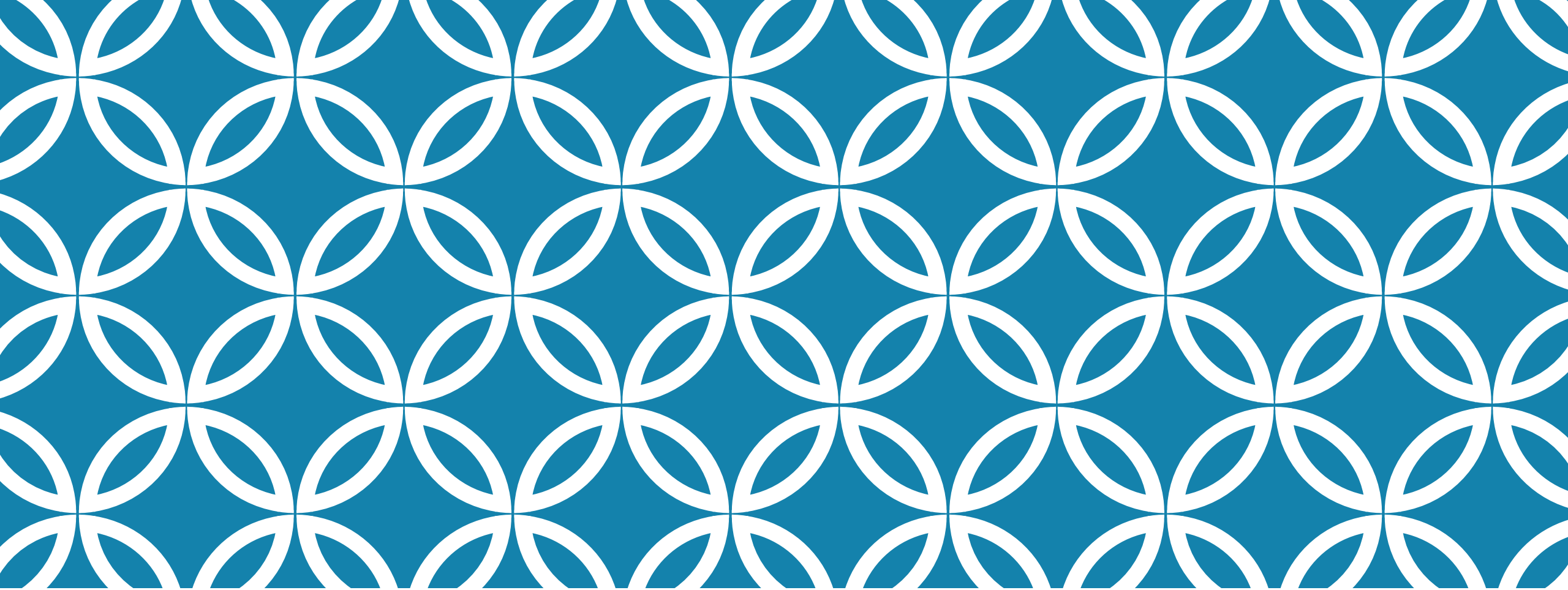


CS5412 / LECTURE 16
THE CHALLENGES OF INTRODUCING RDMA INTO CLOUD DATACENTERS

Ken Birman

Spring, 2019

CONTEXT FOR THIS LECTURE

We saw how the need for performance has pushed some very fancy machine learning components into the edge, like Facebook TAO

As we connect the cloud to sensors, we'll get an even greater demand for real-time updates (hence replication), consistency and coordination at the edge. FFFS and Derecho are examples of a response to that need.

But Derecho's speed comes from RDMA. Does the edge *have* RDMA?

LIFE ON THE EDGE

"Cut through the stack for speed!"

The edge demands disruptive changes.

… early adopters tend to experience a lot of pain.

Nothing works… the hardware may lack programming tools… is undocumented… may even have *hardware bugs*. And "cutting through the stack" may have unexpected consequences elsewhere.

- None of the rosy predictions are as easy to leverage as you might expect.
- Hint: Start by duplicating some reported result for the same setup!

LET'S THINK ABOUT DERECHO

Provides ultra-fast data replication with Paxos guarantees

Key steps:

- Identified a hardware capability that has been overlooked for data replication tools: RDMA transfers
- Studied that hardware closely. It has many capabilities, but used two:
  - Reliable "two-sided" RDMA transfers (Q posts a receive, then P's send can start)
  - Reliable "one-sided" RDMA (Q permits P to write into a memory area)

IS DERECHO AS GREAT AS KEN CLAIMS?

All of our experiments were totally honest.

But… there are complications.

To understand them, we need to understand the hardware better

TODAY'S LECTURE TOPIC

What made it hard to build Derecho?

Where are the surprises?

In what ways did Paxos and virtual synchrony "evolve"

- The underlying concepts were unchanged
- But the implementations are very different than in older systems!

DERECHO STARTS WITH A BASIC FAST MULTICAST

What would be the best way to do an RDMA multicast with reliability similar to N side-by-side TCP connections?

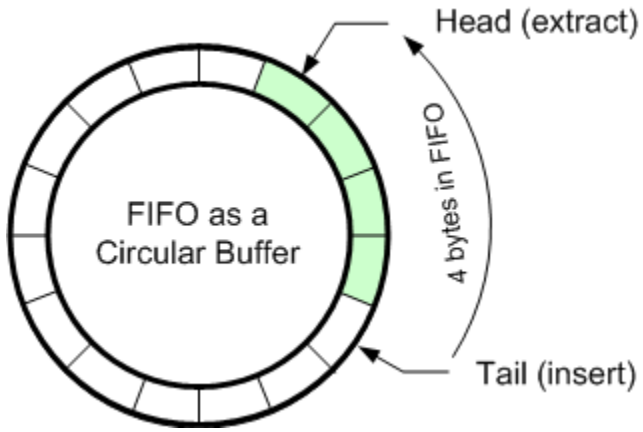
- We could just have N side-by-side reliable unicast RDMA connections
- We could use one-sided RDMA and have N "round-robin ring buffers". The sender could do a lock-free buffer "put" and the receiver, a lock-free "get".
- We could do a tree to disseminate the data using RDMA unicast

IT TURNS OUT THAT THERE ISN'T ONE ANSWER!

The problem with doing N side-by-side RDMA connections is that with reliable RDMA Unicast (or with TCP!) the sender and receiver need to agree on the size of the object being sent.

- The receiver will need to have a suitable memory buffer posted for the incoming DMA transfer.
- So this means the sender must tell the receiver the buffer size first, then wait for the receiver to post the buffer: an RPC interaction.
- Plus, the solution turns out to scale poorly if you do it this way.

WITH SMALL MESSAGES, USE N RING BUFFERS



One-sided RDMA writes into ring buffers work well for smaller messages (maybe up to 1KB).

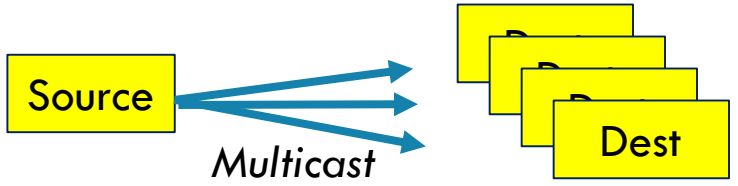
- Inside Derecho this is called SMC.
- There needs to be one ring *per* destination, each with enough memory for R messages. The memory is allocated and posted in advance
- Lock-free updates to the counters of messages in the buffer and free slots are easy to implement.
- The round-robin buffer soaks up any mismatch in speed between the sender and receiver.

WITH LARGE MESSAGES, THOUGH…

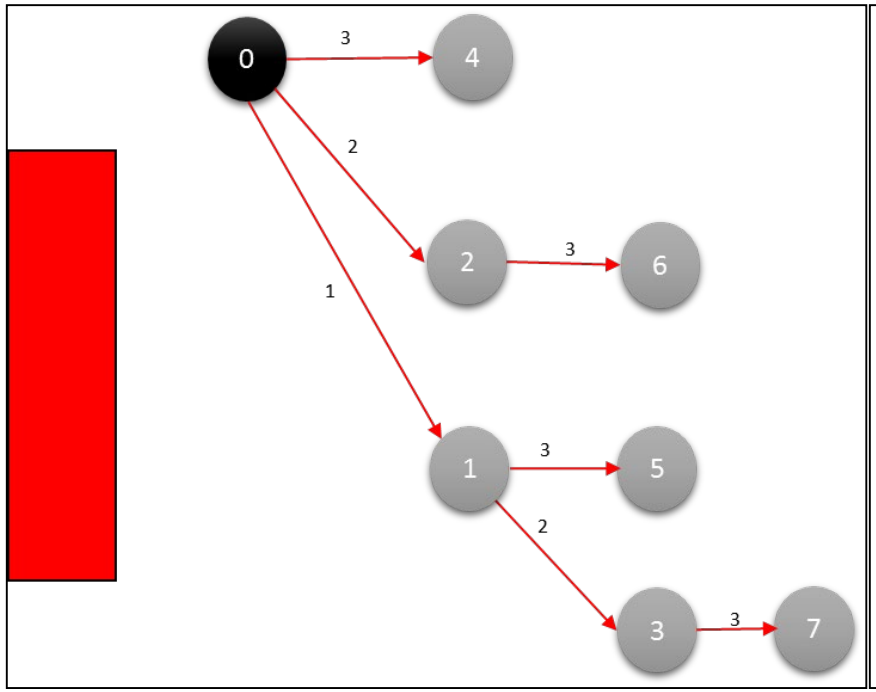
Here we need something fancier.

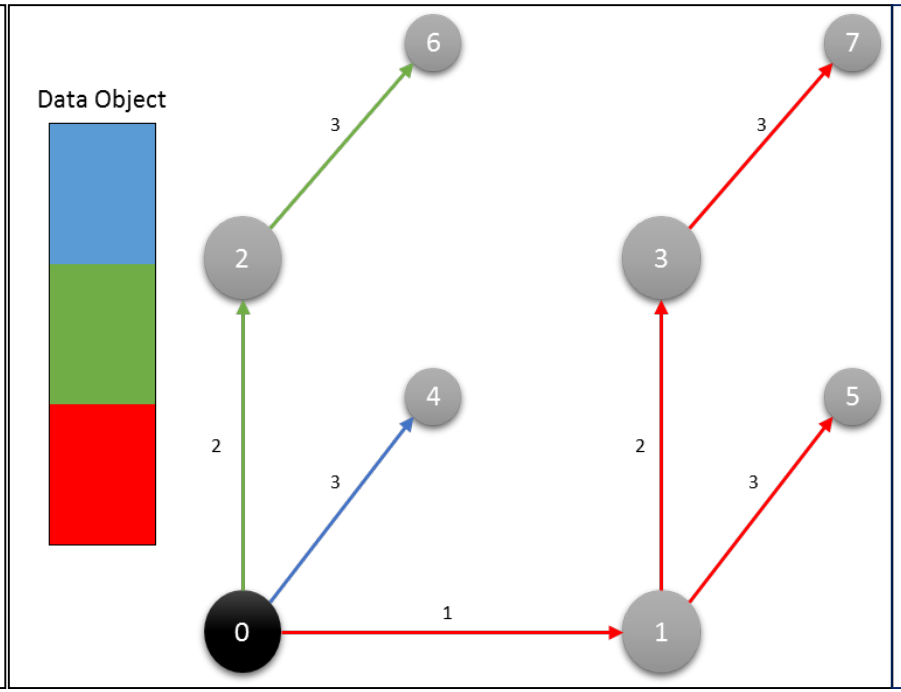
We don't want to do N RDMA unicast writes for a big object: inefficient.

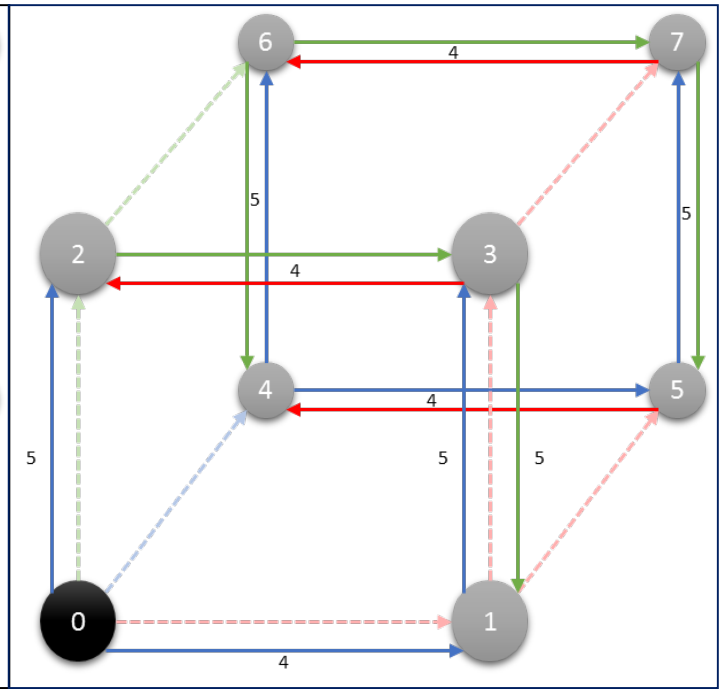
So we need a tree.



RDMC: MULTICAST ON RDMA







Binomial Tree ***Binomial Pipeline*** ***Final Step***

KEY IDEA… AND LIMITATION…

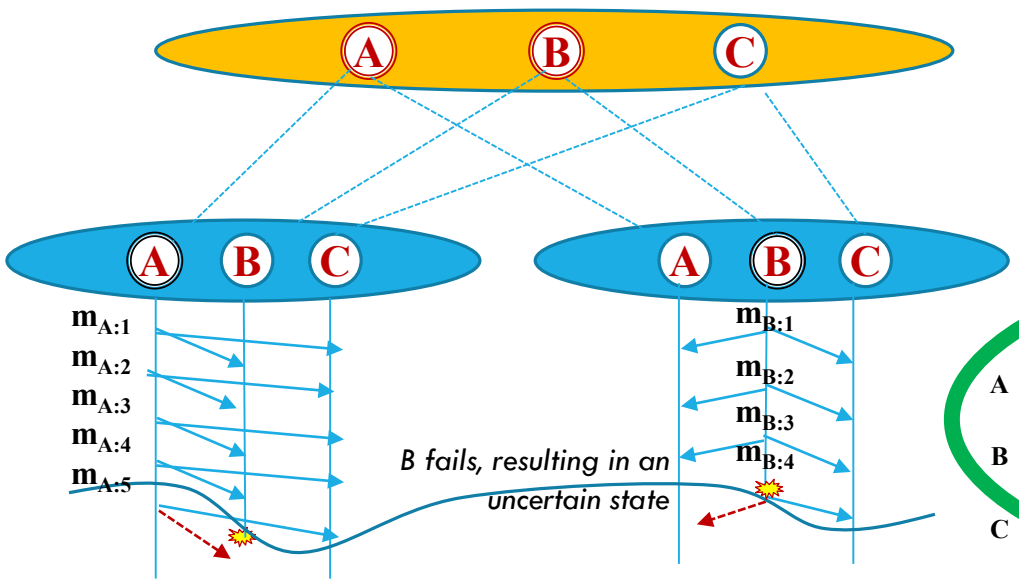
RDMA is good at large, steady streams. RDMC is optimized for that case, and works best as a pipeline. SMC is tuned for streams of small messages.

But protocols like Paxos also need some amount of back-and forth

SMC and RDMC aren't matched to "2 phase" kinds of interactions.

IMPLEMENTATION: DERECHO = RDMC/SMC + SST

Derecho group with members {A, B, C}
in which C is receive-only



$V_3 = \{ \underline{A}, \underline{B}, C \}$

Current view, showing senders A and B
C is a "receive-only" member

	Suspected			Proposal	nCommit	Acked	nReceived			Wedged
A	F	T	F	4: -B	3	4	5	3	0	T
B	F	F	F	3	3	3	4	4	0	F
C	F	F	F	3	3	3	5	4	0	F

DERECHO'S SHARED STATE TABLE

Derecho uses the SST for back-and-forth sharing of data:

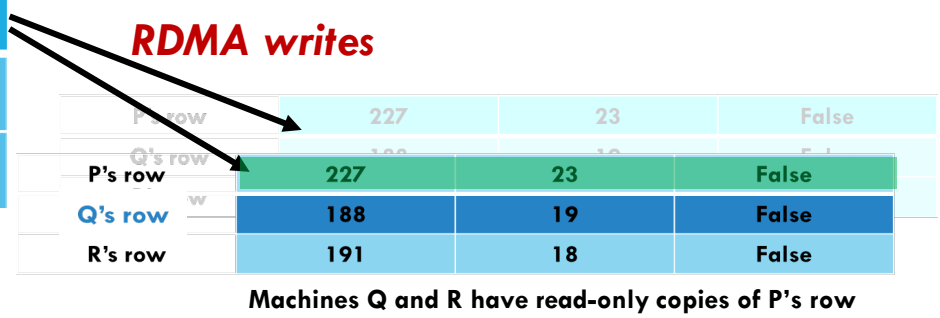
- Instead of messages, SST lets programs talk through shared memory
- Each row is a "struct" in C++. Derecho developers define the format.
- Each machine can write to its own row, "push" to other machines
- Each has a read-only replica of the rows of others

P's row	227	16	True
Q's row	188	19	False
R's row	191	18	False

SST PUSH OPERATION

P updates its row, then SST issues a series of one-sided RDMA writes. These copy the changes to other machines

P's row	227	23	False
Q's row	188	19	False
R's row	191	18	False



Machines Q and R have read-only copies of P's row

… the transfers occur "silently" and Q's row is updated to match P's new version. The actual transfer is via DMA, low address in memory first, reliable, fifo, etc. Q rereads the data to see that it changed

SST PROGRAMMING

Because the SST is lock-free, values can change "under your feet".

But this is also good, in the sense that threads don't disrupt one-another.

It motivates us to program the SST in an unusual way

SST PROGRAMMING

SST programming: via a kind of "predicates"

if (*some condition holds*) { … *trigger this code* … }

- We made a choice: we create rows of "monotonic" values that change in one direction, like a counter (it only gets bigger)
- We define "aggregating" operators that compute things like *min*. If the underlying values only get larger, *min* only gets larger too

STABLE AND MONOTONIC PREDICATES

A deduction (a predicate) is *stable* if, once it becomes true, it remains true

- Suppose **counter** is a column in the SST, and is monotonic
- ***min*(counter) = *v*** is not stable in the SST: if the counters grow, *min* grows
- … but ***min*(counter) ≥ *v***, in contrast, remains true once it becomes true

STABLE AND MONOTONIC PREDICATES

A deduction (a predicate) is *stable* if, once it becomes true, it remains true

Some stable predicates are also monotonic, in this sense:

- If **Pred(min(c))** holds, then ∀ **v < c, Pred(v)** holds.

Monotonic predicates allow receive-side batching of actions, like delivery of messages 0..min(c)

if (… messages 0..min(c) are stable) { deliver(0, min(c)) }

THIS IS NOT AN OBVIOUS WAY TO PROGRAM!

Notice how ***the hardware forced us to program differently:***

- The hardware is very fast, but only if used in a certain way
- To use it in that way, at that speed, we couldn't do "normal" things, like sending messages and waiting for acknowledgements, or votes
- So we had to invent this new shared table abstraction, and had to rewrite the standard Paxos protocols in a totally new way

… NOT UNUSUAL WITH NEW HARDWARE!

New hardware often results in ideas like Derecho

Specialty hardware can be extremely fast, but often requires that you use it in some very unfamiliar way.

If we just run the old style of algorithm on the new hardware, but in the old way, we wouldn't benefit

2-PHASE COMMIT "VIA" SST

P writes something, like "I propose to change the view".

Q and R echo the data, as a way to say "ok"

When all have the identical data in their rows, we consider that the operation has committed.

In fact the SST can carry many kinds of information: values that change (ideally, in one direction: *monotonic*), messages, even multicasts.

A LONG SEQUENCE OF 2PC ACTIONS

For a *single* 2PC, SST wouldn't be much faster

But Derecho has to do 2PC on millions of Paxos atomic multicasts/second

At those speeds, the SST monotonicity pays off

- Senders ask for votes on *all messages up to n*. N is a counter.
- Receivers vote *ok up through k*, and this allows monotonic reasoning.

INITIAL CONCLUSION

We managed to make Derecho very fast, but to do so:

- Had to come up with a way to move bytes at crazy speed.
- Then had to come up with a "control plane" that can run separate from the data plane.
- Turned out it needed a lot of 2PC kinds of mechanism. To get those to be fast we invented this whole way to program the SST.
- A lot of work, but the payoff was extreme speed.

ZERO COPY REQUIREMENT

Moreover, to get the full possible speed of Derecho, you need to write code that won't involve any copying (even using memcpy, or even automated copying done in the programming language runtime).

Copying is slower than RDMA!

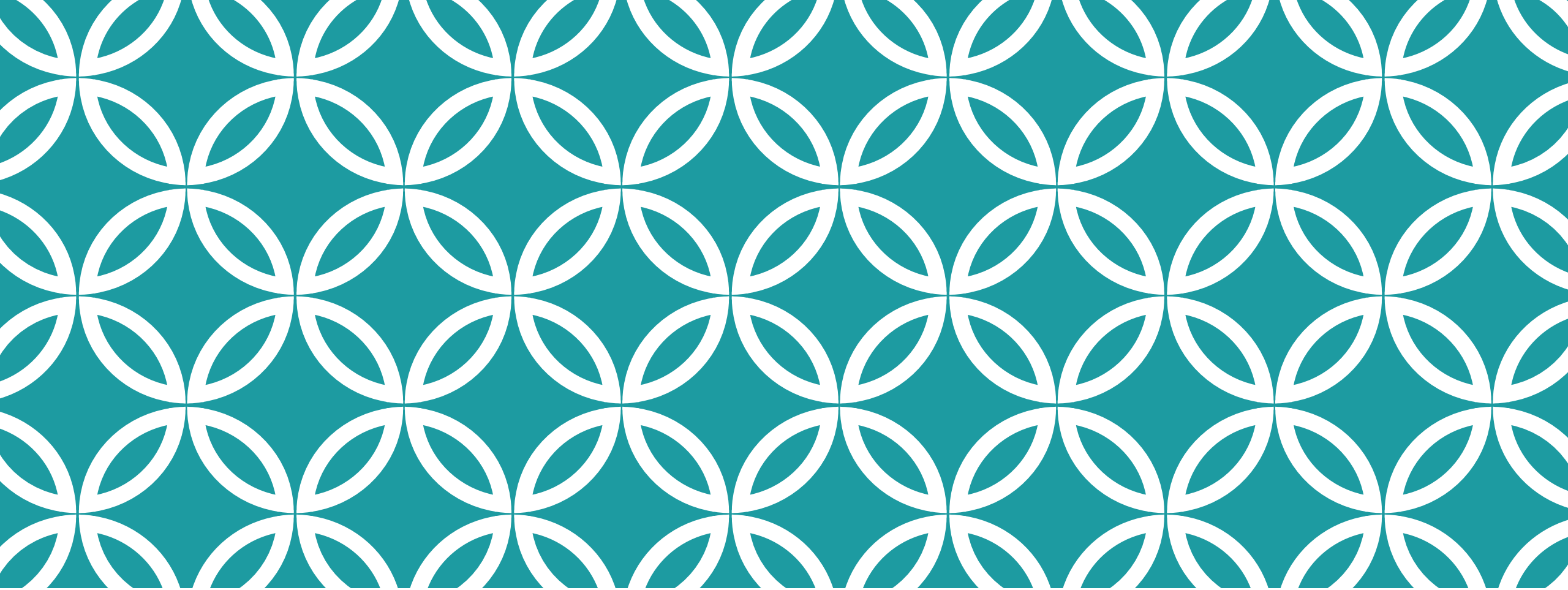
This is quite tricky: Your application needs to use RDMA "everywhere" for large objects where performance will be critical.

OTHER ACCELERATORS: SIMILAR STORIES

GPU units require entire special programming languages (CUDA) and a really peculiar programming style (move object into GPU, load program, press “run”, move results back into CPU memory)

FPGA accelerators have to be coded in a gate-level language, like the ones used for VLSI chip design.

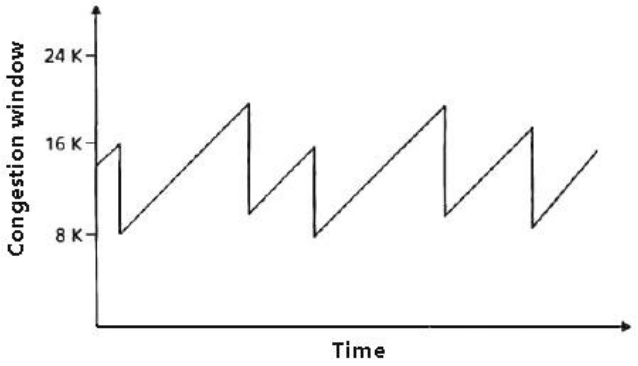
Network interfaces can be programmed, but the run a very strange kind of code focused on moving messages (P4, but it isn’t yet a standard).



NOW WE KNOW ALL ABOUT DERECHO.

Should everyone switch to it in all their edge systems?

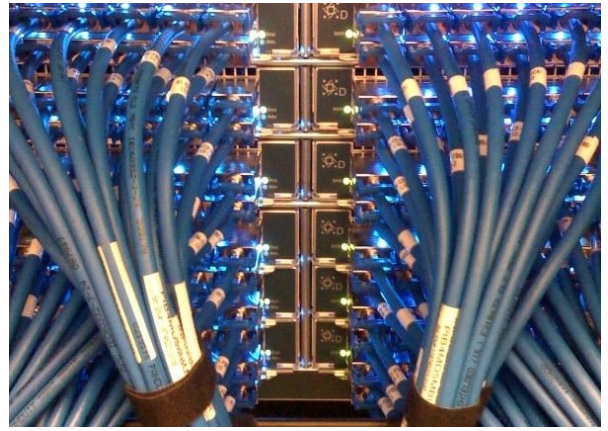
THERE IS A SMALL ISSUE…



Not so fast… RDMA doesn't really work in datacenters! A long history…

- RDMA was invented in connection with a novel networking approach called Infiniband. It competes with optical ethernet
- In ethernet, senders send packets, and packets are dropped if congestion (overload) occurs.
- This causes loss if the packets are part of UDP messages, but TCP retransmits missing chunks.
  - It "backs off" (slows down) if loss occurs
  - Additive increase, multiplicative backoff

INFINIBAND ISN'T ETHERNET



In Infiniband, no device sends unless it has permission to send from the receiving device first

So when a router transmits a packet to another router, for example, the receiver has granted "credit" for the sender to send B bytes.

This is true for every step along the route!

- Hop by hop, no data is moved without assurance of a place to put it
- The optical network layer is so reliable that Infiniband is **lossless!**
  - More precisely: Loss is incredibly rare and usually caused by some form of crash

SUPERCOMPUTERS LOVE INFINIBAND

The "market share" for Infiniband in HPC systems is extremely high

Ethernet is unpopular because those packet drops aren't rare, and this causes erratic performance.

So we now have 15 years of experience with Infiniband with as many as hundreds of thousands of datacenter computers!

RDMA was born in this world: DMA transfer over Infiniband works because hop by hop, no loss ever occurs. Every piece of data is moved reliably at "optical network" speed. Today: 200Gbps (bi-directional) is available

- In contrast, memcpy with a single core is more like 36 Gbps for large transfers that can't leverage the L2 cache.

RDMA ON CONVERGED ETHERNET: RoCE

The idea emerged of running RDMA over optical ethernet, around 2010

Puzzle: RDMA doesn't do retransmits over Infiniband, and a full TCP-style solution wouldn't be nearly as fast

So, how to get RDMA to run in a setting without sender credits?

They introduced a concept of "Priority Pause Frames"

RDMA ON CONVERGED ETHERNET: RoCE

The idea emerged of running RDMA over optical ethernet, around 2010

They introduced a concept of "Priority Pause Frames"

- An overloaded router or switch or NIC sends PPFs to the sender of a flow if it becomes overloaded by incoming data
- The RDMA NIC pauses, then restarts the transfer (the entire transfer) if it receives even a single PPF

… but unfortunately, PPF didn't work very well

- It can generate PPF "storms" and RDMA performance collapse

"BUT DOES ROCE WORK??"

There are many stories of datacenter technologies that didn't work well

They include optical Ethernet multicast (broadcast), early web server technologies, early packet routing solutions, early ways of connecting browsers to web servers, early DDoS attack filters

Often, they disabled entire datacenters when they malfunctioned!

… so datacenter operators are not eager to embrace RoCE yet, because "a little unstable" can mean "my datacenter could be toast"

ETHERNET "BROADCAST STORMS"

A long story, but we'll abbreviate it…

Your friendly local ethernet supports broadcast (all receive everything) but also a more selective "multicast" (it has a form of topics, represented by an integer, and only machines subscribing to the topic receive the messages for that topic).

… but, the integers must be in a fairly small range (12 bits)

IMAGINE A MASSIVE DATACENTER

Suppose it was using this feature

With widespread use, we run low on multicast addresses

And it turns out that even if we fix this (IPv6) the hardware has the same issue for a different reason!

… with a few thousand addresses in use, it *treats multicast as broadcast*

WHAT DOES THIS MEAN? "TREATS ETHERNET MULTICAST AS A BROADCAST"?

In your design, some message was intended for 2 receivers, perhaps even in the same rack (it won't be forwarded, when things work properly).

Instead, it gets delivered to *every computer in the whole data center.*

We end up with all-to-all messages: broadcasts, and everyone receives every single one of them!

TRY TO VISUALIZE THE PATTERN

Every machine suddenly receives 50,000 msgs/s

Suppose that we have a half million machines

Some application forms small groups of perhaps 4 machines. There are 10,000 machines running this: 2,500 small groups. The groups have a message traffic of 20 msgs/s (the 4 members send ~5 msgs/s each).

Now suppose that due to this hardware problem, every multicast behaves like a broadcast. How many messages/second are delivered to an average machine in the data center, one not part of the 10,000?

COULD RDMA MULTICAST OR DERECHO TRIGGER SUCH A STORM?

No, not in a literal sense

- The multicast storm was caused by a feature of routing and NICS specific to the way that Ethernet class-D multicast forwarding is done
- In fact the "cause" is that a hardware hash-table fills up and overflows (underlying limitation: "Bloom filter" hash tables are too small)

But in the larger sense, the story is about how a technology used by one subsystem can overwhelm the whole datacenter and disrupt other systems

- So you need to ask: "*Could enabling use of RDMA disrupt the cloud?*"

DERECHO: THE REMAINING PUZZLES

To get the full speed of the technology

- You need to work in C++. But many people prefer Python, Java…
- Programs need to be "zero copy". But most people have no idea if the packages they use do copying
- Your code needs to be nearly lock free and rather pipelined. Few people are used to coding this way

Is it worth it? Derecho is as much as 15,000x faster than other options… but only if you use it properly!

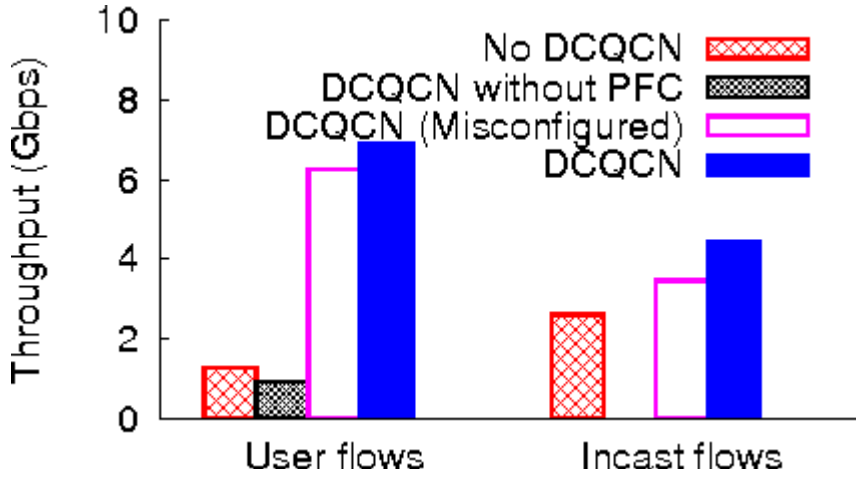
REPRESENTATIVE SAD STORY

Do you recall the first pictures
you saw of massive cloud datacenters,
circa 2004?

Puzzle: What were all those machines
really "doing"?

For years, the answer was: "Most of them are rebooting"

DCQCN



Data Center Quantum Congestion Notification is a new idea from Microsoft and Mellanox that uses end-to-end congestion notification

Basically, these are the same as credits in Infiniband

Experiments on modest datacenter clusters worked well!

- RDMA + DCQCN enabled RDMA to work in normal datacenters!
- A disruptive and transformational development!

Not so fast…

MICROSOFT AZURE SET UP A RED TEAM

Goal: Deploy DCQCN *side by side* with normal TCP/IP in a new Microsoft datacenter, during the burn-in testing period (about six weeks long)

Test aggressively – try and see if they can trigger problems.

- Includes misconfiguration, but not "breaking the logic"
- They want to know: are we getting into trouble here?

Guess what? It didn't work!

ISSUES THAT WERE IDENTIFIED

RDMA and normal TCP/IP interfered with each other.

PPF standard requires "Enterprise VLAN" feature on switches. Azure doesn't use this technology.

But they solved these

- They repurposed another (also unused) feature called "DiffSrv"
- There was a DiffSrv packet format that could be reused for PPFs
- DiffSrv also allowed TCP/IP to run side-by-side with RDMA, isolated

"Jim, it dinna work. Antimatter containment will fail in 3 minutes!"

… IT STILL DIDN'T WORK

They discovered that RDMA + DCQCN + DiffSrv + PPF could cause a new kind of routing / congestion loop

It was triggered by a form of resource exhaustion because the TCP/IP layer and the RDMA layer were sharing buffers inside NICs, switches and routers.

By dividing the resources into pools, this could be solved. But their hardware lacked a way to do that. They solved it by "overprovisioning"

- They bought more memory for the routers and switches than needed
- Then configured to make sure that the memory never gets used up

MORE LIMITATIONS

RDMA only works if the application is working from "pinned" memory pages, and works best if the memory pages are huge.

- Seems to be at odds with virtualization
- Advances in RDMA hardware may help reduce these issues

When an RDMA transfer finishes, the program can definitely access the data. But if you instantly do a DMA transfer to disk, or try to display it on a graphics device, caches and pipelines may need to be flushed first.

- There is no standard way to actually do this.

IN THE END, THEY GOT IT RUNNING!

Today, Azure uses RDMA, but only for system services (for now)

Azure HPC actually does have application-layer RDMA, but only for people who use a package called MPI.

- In fact, MPI doesn't share the devices with normal TCP/IP
- Instead, Azure HPC computers have an entire extra Infiniband network

In the future, Microsoft may expand use of RDMA to allow some trusted subsystems to also use it. Probably it will never be free for general use.

… SO, CAN YOU USE DERECHO ON AZURE?

Nope. It may be possible to run Derecho on Azure HPC soon, and perhaps also on Azure containers.

But virtualization seems to be incompatible with RDMA

And even where Microsoft has RDMA, they don't allow you to access it yet.

Plus, if you had Derecho, would you be able to get the full speed?

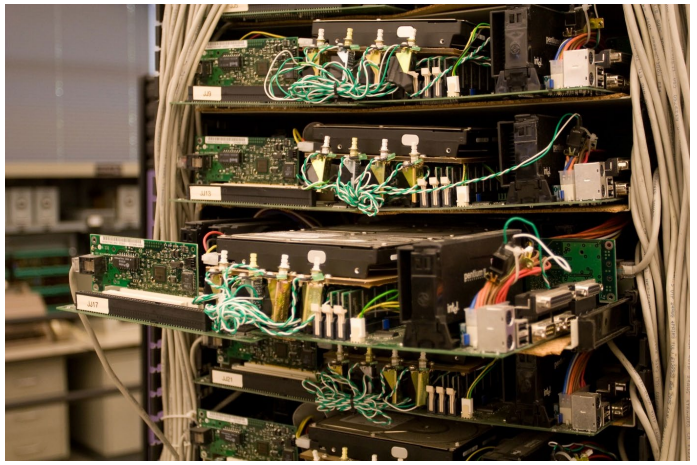
RDMA IS JUST ONE OF MANY "PLAYERS"

This story of RDMA as an accelerator in the cloud is just one of a few

Each technology has many hurdles to overcome!

- Is it *way* faster than not using it?
- Will it save the cloud owner money?
- Is it stable? Really stable?
- How hard will it be to "manage"?
- How hard is it to program?

PROGRAMMABLE COMPONENTS



General purpose cores on NUMA machines

Network Interface Card (NIC) for modern RoCE (RDMA-capable Converged Optical Ethernet). Optical network itself.

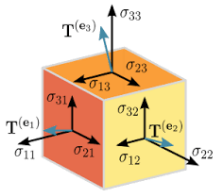
Storage components (SSD)

Network Switches and Routers

Field Programmable Gate Array (FPGA), FPGA clusters, ASICs.

- GPU (graphics accelerator) and GPU clusters.
- TPU (tensor processor unit) and TPU clusters.

Quantum computing hardware



AGAIN, A SIMILAR STORY

Every one of these has amazing potential, but to leverage it can require changing all sorts of things that used to be standard in Linux

As the modern data center evolves, we will run into that issue often.

Yes, we want accelerators. But the pain level is substantial!

ROUGH ROAD AHEAD!

The latest new thing always sounds amazing…

Paradise awaits!

Road to Paradise

The pitch. What could go wrong?

The boss: Paid to say "no"!

100x Speedups have a chance of adoption, if your pain tolerance is high!