Lecture 2 - High performance communications

Gidon Rosalki
2026-04-14

Notice: If you find any mistakes, please open an issue in the github repository

1. Part 0: Reintroduction

This was originally just a course in networking, and has since had more stuff added in. For those of us that expect a course that handles lots of switching, and the like, this will not be that. Prior knowledge of TCP/IP is pretty much sufficient. We will remind TCP, and even discuss in what ways it is insufficient for our needs, and on methods that are better for us, such as RDMA. How this is implemented efficiently, and from there we will move on to high performance computing, and how this is implemented on super computers.

2. Part 1: Networks, Properties and Metrics

In order to compare between standard networking, and high performance networking, we need to measure the performance, and so we need to establish *what* we are measuring:

- Latency: AKA delay
  - This is the time it takes for a message to travel across the network.
  - Also consider the RTT (Round Trip Time), which is the time for a message to go from A to B, and back to A (this is much easier to test)
- Bandwidth (BW)
  - The rate of passing messages (volume per unit time)
  - For an analogy, consider the bandwidth to be the width of a road (the number of cars that can pass through per second), and the latency to be for how long they drive
  - Most laptops / computers have bandwidths of a gigabit, servers tend to run at around 10 gigabit, newer servers and higher performance servers are starting to run at around 800 gigabit, or even higher
- Loss / drop rate
  - This is the rate at which messages are sent, but not received (how many packets are lost per second)
  - Occurs for example when switches drop packets, due to network overload
  - It depends on the properties of the traffic, and may be measured as a percentage of messages over time
  - This is often not visible to application when a lower layer applies the flow control
- Jitter
  - Variation in delay of received packets
  - A common example of something that increases jitter is cache misses, since then you have to access slower memory segments
  - It can be said that the packet that is most delayed is the most interesting, since if we have a million processors that are working together, they will all be stuck waiting for the last packet, so if one arrives particularly late, then we want to know *why*

When discussing networks, there are many other parameters to discuss, such as

- The network topology
  - The shape of the network connections as a weighted graph of entities
- Bisection bandwidth
  - The minimal bandwidth between two halves of the network (the halves are selected to find the minimal BB)
- Routing protocol dynamics
  - The way messages are routed across the network

The different topologies become particularly important, since supercomputers are no longer single site affairs. They require more energy than can be provided in a single location, so are split across an entire

country, or even across many computers. At this point we run into the speed of light limit, and discover that for our purposes, it is really rather slow. Consider two computers that are kilometres apart: The light could take order of milliseconds to arrive, which is very slow in high performance computing.

The routing protocol becomes very important when we consider trying to send messages across the network. It's possible that my network can handle sending at a specific rate, but because the protocol chose to send 2 messages down a single link, that did not quite have the bandwidth to handle it, my entire network can experience a significant slowdown. Remember, in parallel computing, the slowest sets the rate of work. So: the routing protocol is important to ensure that the overall network speed is as fast as possible.

2.1. Bounding protocol

Consider, a switch can connect n_1 computers, and we have n_2 switches. In order to connect all the switches, we connect them with 1 more top level switch. The problem with this is that for computers to communicate across the switch boundary, they are now limited by the speed connecting to the top level switch. We require that the bandwidth between the computers to their switches, and the bandwidth between the switches, and the top level switch(es) to be equal, in order to enable non blocking communication.
Now, if we have n_2 switches, and n_2 top level switches, then we have a "simple" solution, of ECMP, and simply choosing the link with the largest available bandwidth at a given time. This is not just used thanks to the limitations of TCP, the requirement of ordered messages, and how TCP is actually a very slow protocol, which massively increases bandwidth usage when messages are dropped. There are other methods to resolve this problem, we will return to them later in the course.

2.2. Application Properties

We need to consider the traffic composition: The different message types, and sizes. When we consider this, we can use a more appropriate protocol such that we increase the speed on an application basis.

3. How we measure

We would love to be able to send a message from A to B, and then know how long it took. This does not work, since the time on the computers is not synchronised well enough. What we do, is send many messages, and have B respond immediately, take the average RTT, and divide it in two.
We will note that the first few messages are almost always much slower, there will likely be many cache misses, DNS, IP, and more, and on top of that, lots of these protocols use slow start.
To handle this, we will send x iterations of "warmup". Once the warmup has finished, we will send n messages, and measure those.

We will not simply do that in the first exercise, we will actually be measuring the bandwidth / throughput. The bandwidth is the speed at which the cable can transmit, so for gigabit, 10^9 bits may pass through the cable every second. The *throughput* measures how many of these bits are **useful** to us. This is a measurement of the protocol, and the cable, rather than just the cable.

The Ethernet protocol can at minimum send 64B messages, and no smaller than that. This is related to historical reasons of ALOHA, and CDMA/CD. When transmitting messages over the network, we cannot know that there was a collision until the end. If the packet is too small, then we could well miss that there was a collision at all. Despite protocols moving on, since this history underpins it all, we still send packets of at least 64B in size, which was large enough to ensure that we did not miss a collision in the original versions of Ethernet.
We will however note that processes are not fast enough to only use packets of 64B in size, and we generally use packets that are *much* larger. They cannot be too much larger, since then packet drops become a significant data loss.

Our first exercise will be to measure the throughput between computers in the lab, and measure the throughput for different packet sizes. We will then turn this into a graph, of measured throughput, against packet size.

So, how do we measure throughput? Very similar to what we did regarding the latency. Instead of 1 packet, and its RTT, we send many (say 1000), and at the end, the second computer states that it received. This will enable us to know how long it took for all this data to arrive, and we may compute the throughput of the network.

4. Acceleration by Offloading

Offloading is getting someone / something else to do the work. One might offload TCP, where we move IP and TCP processing to the NIC (Network Interface Card). The main justification for communication offloading breaks down to

- Reduction of host CPU cycles for protocol header processing, checksumming
- Fewer CPU interrupts
- Fewer bytes copied over the memory bus
- Potential to offload expensive features such as encryption

This brings us to a new metric to measure! **CPU utilisation**. This is the proportion of time that the CPU is used for actual work, since time spent on communication may be considered *wasted*. We bought the processor to waste its time multiplying matrices for AI, or perhaps to analyse data, or all manner of useful things. Not generate packet headers. So it becomes a rather good idea to offload the work of packet understanding to the NIC.

However, transport offloading may not be enough, since the CPU must still perform a system call, which is incredibly expensive, with the context switching, memory copying, and so on. A solution to this is called *U-net*. This is a virtual network interface that allows applications to send and receive messages without operating system intervention. This moves all buffer management and packet processing to user space, making this a "zero-copy" protocol, since we have avoided the memory copy step. Efforts of U-net eventually resulted in the *Virtual Interface Architecture* (VIA), which in turn led to the implementation of various high performance networking stacks such as Infiniband, iWARP, RoCE. These are commonly referred to as RDMA (Remote Direct Memory Access) stacks. Infiniband would seem to be the one which succeeded the most, at least in the world of consumers.

4.1. OS bypass & RDMA

The basic idea is to cut out the middle man. The user level application may directly access the NIC, without going via the OS. In the literature, a zero-copy is an operation that does not use any additional memory.

4.1.1. RDMA standards

- RDMA is traditionally used for Infiniband networks
  - Infiniband has a number of vendors, Intel, Qlogic, Mellanox
  - It is used extensively in HPC (supercomputers)
  - Expensive, and requires specialised hardware (physical network and NIC)
  - Standard is 100GB/s

We all recall the layered model for networking. This is a wonderful system, in a general case. The power of RDMA and Infiniband is that it was designed for specific traffic, in specific locations (inside a datacentre), but not for across the country. Problem is, getting rid of Ethernet is difficult, especially since it standardised to everywhere, so there was a desire for RDMA over Ethernet, such that they get most of the improvement, without the expense of the specialised hardware

- RoCE: RDMA over Ethernet (instead of Infiniband)
  - RDMA over Converged Ethernet
  - Still requires specialised hardware, but cheaper since it is only specialised NICs
  - 40Gb/s (maybe up to 60Gb/s)
  - RoCE seems to scales worse than Infiniband
- iWARP
  - RDMA over TCP, again reducing the amount of specialised hardware
  - Once again, cheaper, only needs specialised NICs
  - This has mostly not worked, Gil thinks that there are no companies left that use it

- There are lots of arguments over which protocol is preferable. Like everything, I suspect it depends on additional context.

4.2. Data Transfer Model - Work queues

If we're running RDMA over things like Ethernet, and TCP, we need to discuss how we avoid the OS. To do this, we will discuss Work Queues.

- Work Request: work items that the hardware should perform
- Work Completion: When a WR is completed, it may create a Work Completion which provides information about the ended WR (Type, opcode, etc.)
- Work Queue (WQ): A queue which contains WRs.
  - Scheduled for completion by the HR, ordering guaranteed within a single queue, no guarantee about ordering across queues
  - Adding WR to WQ is called "posting to WQ"
  - Every WR posted is considered "outstanding" undtil it ends with Work Completion. While outstanding:
    - One cannot know if it was scheduled by HW or not
    - Send buffers cannot be reused / freed
    - Return buffer content cannot be determined
- Send Queue (SQ)
  - A WQ that handles sending messages
  - Every entry is a Send Request (SR), it specifies:
    - How data is used
    - What memeory buffers to use (to send or receive data, depending on the opcode)
    - How much data is sent
    - More attributes
  - Adding an SR to an SQ is called "posting an SR"
  - SR may end with a Work Completion
- Receive Queue (RQ)
  - A WQ that handles incoming messages
  - Every entry is called a Receive Request (RR), which specifies memory buffers to be used
  - Adding an RR to the RQ is called "posting an RR"
  - Always ends with work completion
  - May send data as response, depends on opcode
- Queue Pair (QP)
  - An object which unifies both SQs, and RQs
  - Every queue is independent
  - Every QP is associated with a Partition Key (P_Key)

To discuss this further, let us define:

- Requester: The active side, posts Send Requests
- Responder: Post Receive Requests (before data is received)

4.3. RDMA Opcodes

4.3.1. Atomic

Similar to the atomic codes in C/PP, but from RDMA. We have atomic to allow us to have only one process access memory at once. We have the compare and swap operation, to compare against a value, and swap it base doff the condition. Recall OS.