Paper Review

Yusu Weng

CholeSharad, AndyAlizadeh,FingerhutMohammad, ChuangShang-Tse, KeslassyIsaac, OrdaAriel, & EdsallTom. (2017). dRMT: Disaggregated Programmable Switching. SIGCOMM '17 Proceedings of the Conference of the ACM Special Interest Group on Data Communication.

1 Background

Today, the predominant architecture for programmable switches is the Reconfigurable Match-Action Table (RMT) architecture. RMT uses a pipeline consisting of several programmable match-action stages to match and modify specific header. However, RMT has two key flaws as a result of its pipelined architecture.

The first drawback of RMT is its poor resource utilization because of the fixed allocation of memory in each stage. For instance, if a table is too large that must be spread over multiple stages, then the match/action units will be wasted.

Second, a hard-wired pipeline must obey a fixed order of match/action, which will cause a waste of match or action unit if matches and actions are imbalanced in the program.

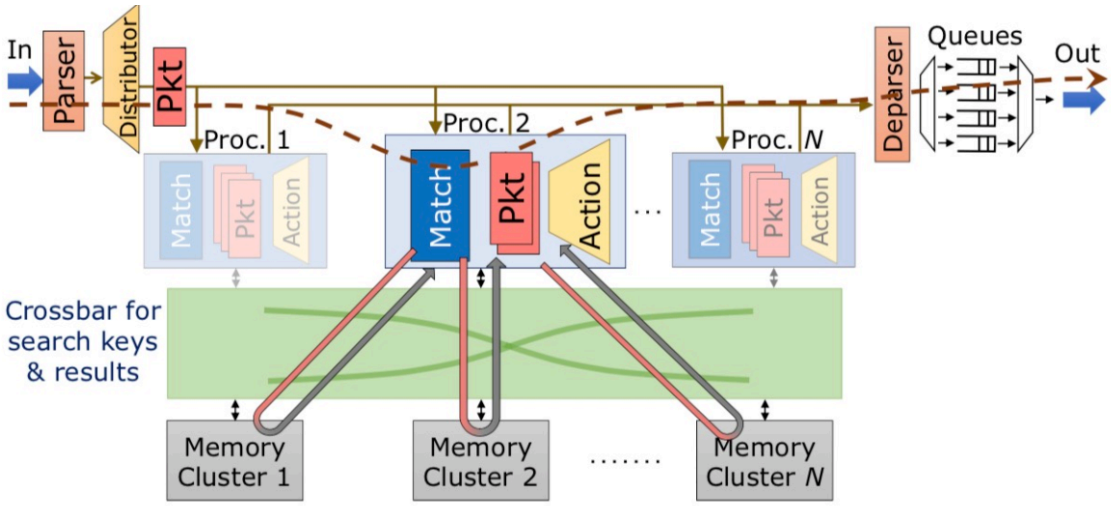
In this paper, the author proposed an improved architecture called dRMT (disaggregated RMT) to solve both problems.

2 Summary

The dRMT's key idea is to disaggregate hardware resources i.e., memory and compute of a programmable switch.

Memory Disaggregation

As mentioned previously, RMT limits each stage can only access its local memory. By contrast, dRMT adds a crossbar in the architecture to let the processor can access all memory clusters.



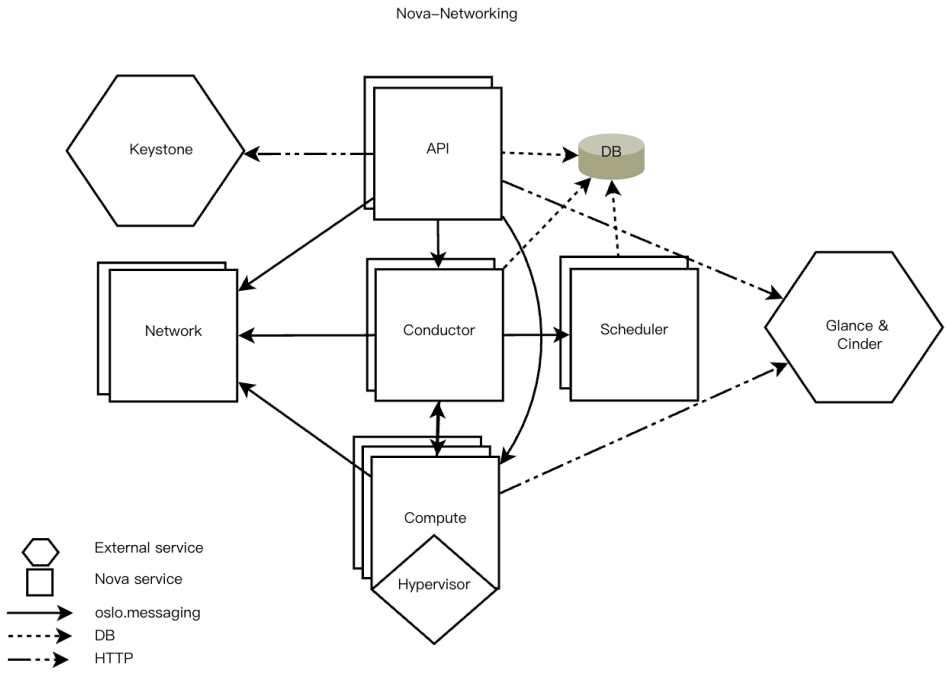
Compute Disaggregation

In RMT architecture, operations must obey the sequence of match-then-action. However, dRMT utilizes processors to replace stages to achieve compute disaggregation. Matches and actions are allowed to interleave execution in the processor in respect of dependencies and resources constraints.

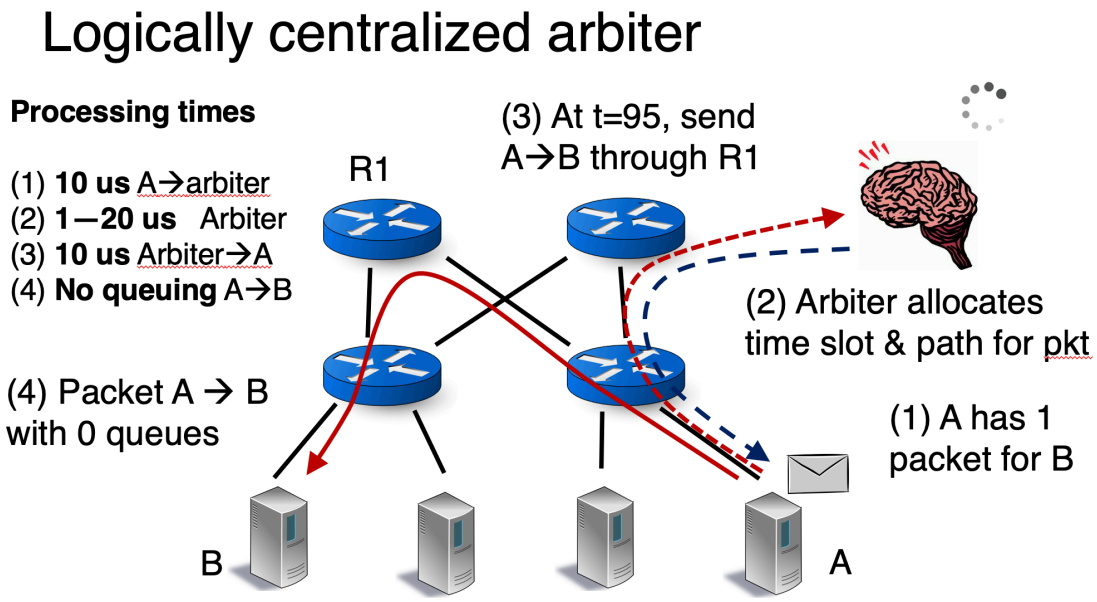
3 Contributions

dRMT mainly focuses on improving resources utilization of RMT architecture. The significant reason leads to poor resources utilization in RMT is due to its fixed allocation of memory and computing units. dRMT tries to solve this problem by memory disaggregation and compute disaggregation. Here I think the "disaggregation" can be understood as collecting all resources together then allocate them according to program's needs. In memory disaggregation, all memory clusters are gathered together and can be accessed by any processors by a crossbar. Similarly, in compute disaggregation, processors are collected and allocated to each incoming header. The key insight is to make the resources flexible to meet the changing demands and needs.

Such resources allocation insight can be applied on many other fields. For instance, in cloud computing, the central server can achieve an elastic distribution of network and memory resources among virtual machines. A concrete example can be found in Openstack architecture. Nova is the OpenStack project that provides a way to provision compute instances (aka virtual servers). Nova-scheduler can automatically allocate computing resources based on its schedule algorithm.



Another example can be found in previous lecture when talking about Centralized Arbitration for Data C**enters. In Fas**tPass project, The Arbiter gains the control of all links and allocate them by time slot and path for the packet.



4 Future Work

It seems dRMT is a totally improved version of RMT and it's hard to find the flaw of it, so I will just propose some potential improvement that can be done in future.

First, the new architecture compared to RMT architecture may bring extra cost on hardware. For example, we can see that the match latency in dRMT is 22 cycles which is higher than 18 cycles

in RMT because of adding the crossbar. This may make a difference when the number of match stages scaling up. Also, dRMT chip requires additional chip area compared to RMT chip. These hardware cost can be improved in future work.

Second, the compute scheduling in dRMT is to manage a processor to handle all matches and actions in one header. A possible attempt is to decouple the processor from the header. I cannot say that this change can improve the performance with certainty, but such scheduling may be helpful in some special circumstance. Consider two packets belonging the same connection that flow in the dRMT switch as the order they were sent, if these two packets are handled in parallel but the latter one has a longer header. Because each header is individually handled by a single processor, as a result, the second packet will reach the end of pipeline earlier than the first one and lead to an out-of-order flow. If all processors can participate in this process, then packets can reach the egress according to the order they flow in.