Why we don’t need to specify the load balancing algorithm in the Standard.

Paul Congdon
Sundar Subramaniam
Hewlett-Packard Company

Load Balancing Algorithms

Load Balancing Algorithms

- Considerations
- Some Algorithms
- Example of a bad algorithm
- The Receive Algorithm
- Mixing Algorithms and Topologies
- What Needs to be Specified
- Conclusions

Considerations

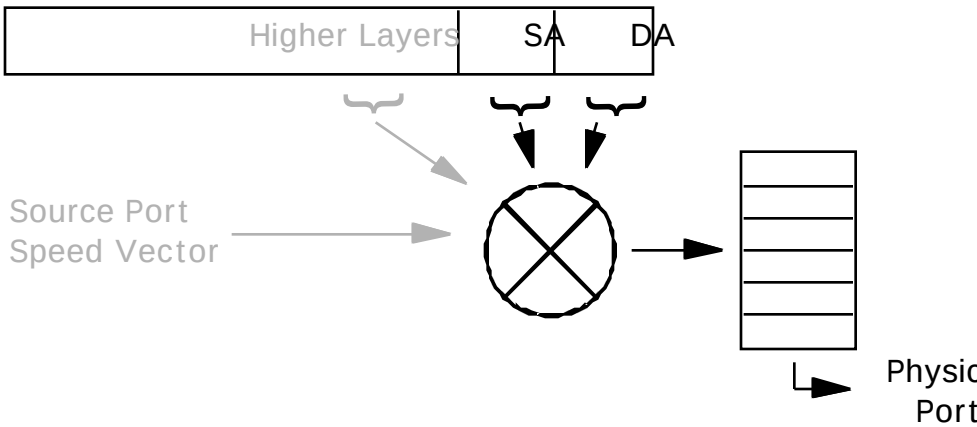
- Frame ordering must be preserved for a particular SA-DA pair (of same priority). NOTE: This can be controlled by the sender.
- A single algorithm for receiving frames on the aggregation possible - all frames come from a “logical” port. NOTE: Multiple receive algorithms may require negotiation.
- Frame duplication can not occur.
- Fragmentation and re-assembly across the aggregation does not scale without a hardware assist.
- Link aggregations with mixed speed and MAC type can be made to work, but with additional complexity.

Some Algorithms for Sending

Load balancing algorithms must define the function

F(x1, x2, x3,....) = physical port

Note : Result must always be the same for a given “flow”

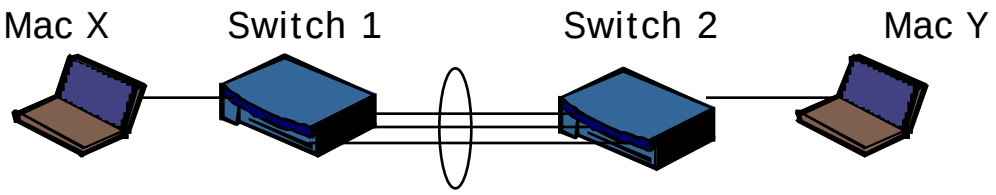


Some possibilities are:

- F(SA)
- F(DA)
- F(SA, DA)
- F(SA, DA, SrcPort)
- F(Level3, Level4 information)
- Conditional functions
  e.g. if Multicast traffic use F(DA), else use F(SA, DA).

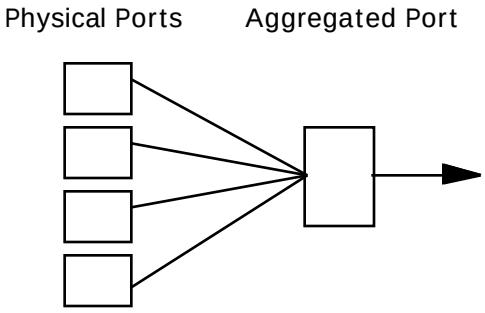
An Example of a Bad Algorithm

- F(FDB(DA)) = physical port
- Use the switch's forwarding database to distribute addresses across the aggregation as they are learned. NOTE: in this case the FDB still references physical ports.
- What happens when FDB(DA) fails? - Use a pre-defined flood link
- Example
    1. Switch 1 has learned X, but not Y. Switch 2 knows both
    2. X sends to Y, and Switch 1 uses pre-defined flood link
    3. Y sends back to X via known path in Switch
    4. Switch 2 learns Y, applies algorithm and assigns Y to link
    5. Next frame from X to Y travels over link 3 (potentially passing previously flooded frames



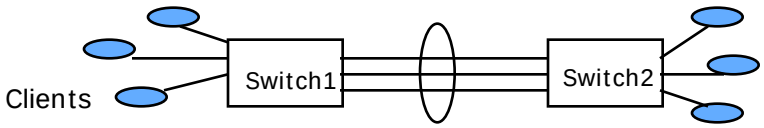
A Single Receive Algorithm

- All frames are received on the aggregated link are handled as though they came from a single port for:
  - Switch Learning
  - Higher Layer Functions
- Order is not “made worse” by the receiver, and “flows” remain in order from the sender’s perspective.

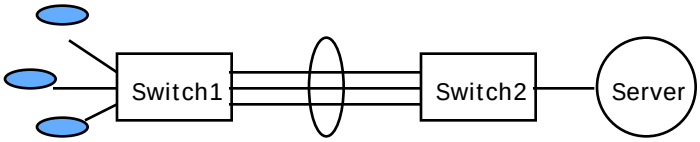


Mixing Sender Algorithms

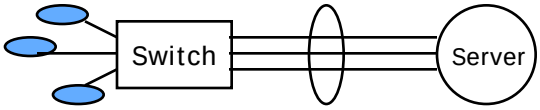
Some combinations are more optimal - But order is preserved!



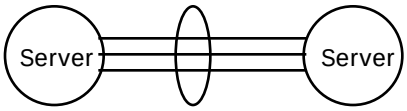
Many-to-Many



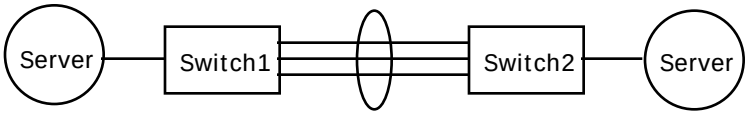
Many-to-one



Many-to-one (direct)



One-to-one (direct)



One-to-one
(indirect)

Many Possibilities

Switch1 - F(SA) or F(DA)
Switch2 - F(SA) or F(DA)

Switch1 - F(SA)
Switch2 - F(DA)

Switch - F(SA)
Server - F(DA)

Server - F(Layer3, Layer4)
Server - F(Layer3, Layer4)

Server - F(Layer3, Layer4)
Switch1 - F(SA,DA)
Switch2 - F(SA,DA)
Server - F(Layer3, Layer4)

Don’t Specify the Algorithm

Only is basic requirements

Requirements

- ’Frame order must be preserved within a “flow”
- ’Basic flow is an SA/DA pair, however...
- ’Higher layer flows can supersede (at least at the originator?)

Why we shouldn’t standardize the algorithm

- Inter-operability is not an issue - devices implementing different algorithms can inter-operate.
- Would take a lot of time to decide which is the best - delays the standard.
- Optimal algorithm is often topology specific.
- Leave room for vendors to enhance and optimize.

Conclusions

- Load Balancing Algorithms should be deterministic,
  at least for a particular “flow”.
- There are many choices for good algorithms - Some are better for some topologies.
- Mixing good algorithms always works, given a common receive algorithm!
- We don’t need to standardize the algorithm, only its requirements.