TSN: Lecture 7
Circuit & Packet Switching II

Topics Covered

- Blocking
- Sorting
- Merging Networks
- Effect of packet size on switching fabrics

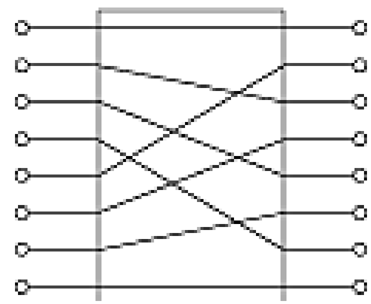
Blocking

- Can avoid with a buffered banyan switch
  - but this is too expensive; how much buffer at each element?
  - hard to achieve zero loss even with buffers
- Instead, can check if path is available before sending packet
  - three-phase scheme
  - send requests
  - inform winners
  - send packets
- Or, use several banyan fabrics in parallel
  - intentionally misroute and tag one of a colliding pair
  - divert tagged packets to a second banyan, and so on to k stages
  - expensive (e.g., 32x32 switch with 9 banyans can achive 10^{-9} loss)
  - can reorder packets
  - output buffers have to run k times faster than input

Sorting

- Or we can avoid blocking by choosing order in which packets appear at input ports
- If we can
  - present packets at inputs sorted by output
  - similar to TSI
  - remove duplicates
  - remove gaps
  - precede banyan with a perfect shuffle stage
  - then no internal blocking
- For example

  [X, 011, 010, X, 011, X, X, X] -(sort)-> [010, 011, 011, X, X, X, X, X] -(remove dups)-> [010, 011, X, X, X, X, X, X] -(shuffle)-> [010, X, 011, X, X, X, X, X]
- Need sort, shuffle, and trap networks



Shuffle Exchange

This input when presented to Banyan Network is non-blocking

Sorting

- Build sorters from merge networks
- Assume we can merge two sorted lists to make a larger sorted list
  - Called Batcher Network
  - Needs $\lceil \log N \rceil \lceil \log N{+}1/2 \rceil$ stages
- Sort pairwise, merge, recurse
- Divide list of *N* elements into pairs and sort each pair (gives *N/2* lists)
- Merge pair wise to form *N/4* and recurse to form *N/8* etc to form one fully sorted list
- All we need is way to sort two elements and a way to merge sorted lists

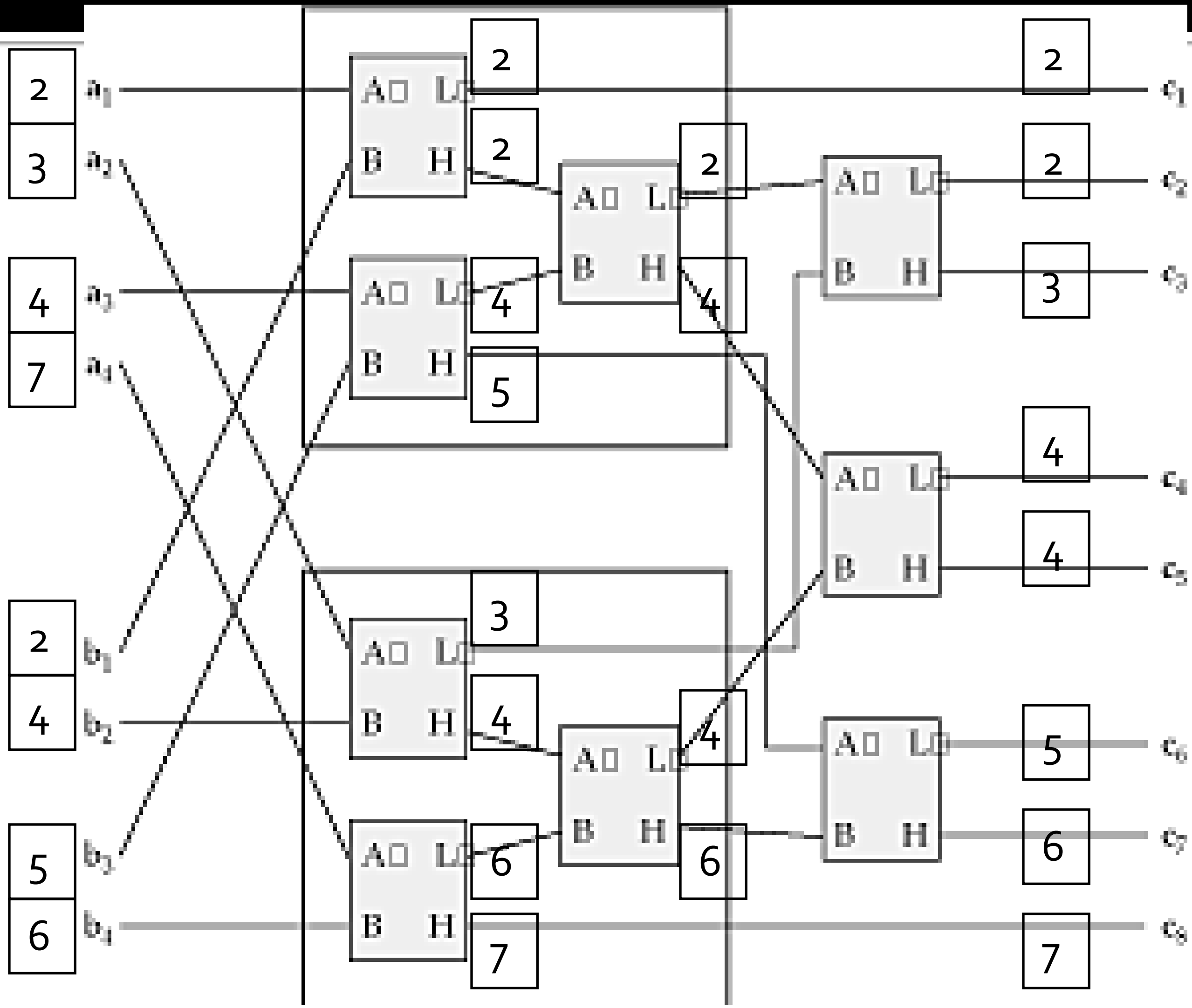
Sorting (Example)

- Sort the list 5,7,2,3,6,2,4,5 by merging
- Solution:
  - Sort elements two-by-two to get four sorted lists {5,7}, {2,3}, {2,6}, {4,5}
  - Second step is to merge adjacent lists to get four element sorted lists {2,3,5,7}, {2,4,5,6}
  - In the third step, we merge two lists to create a fully sorted list {2,2,3,4,5,5,6,7}
- Sorter is easy to build
  - Use a comparator
- Merging needs a separate network

Merging Networks

- A merging network of size 2N takes two sorted lists of size N as inputs and creates a merged list of size 2N
- Consists of two N-sized merging networks
- One of them merges all the even elements of the two inputs and the other merges all the odd elements
- The outputs of the mergers are handed to a set of 2x2 comparators

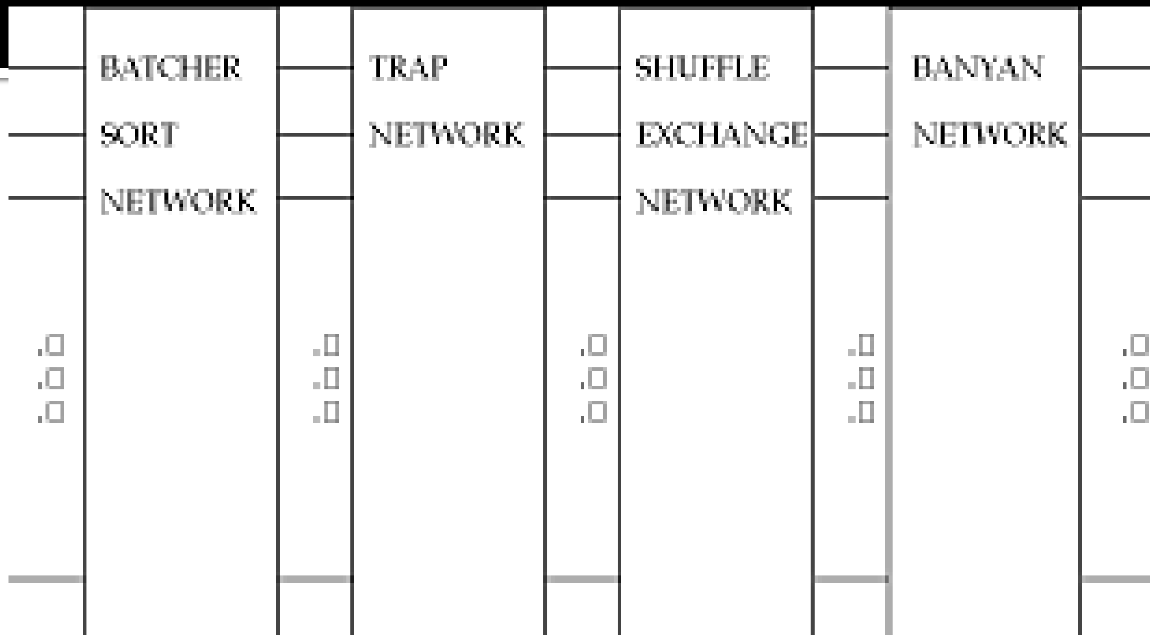
Merging



Merging Example

- Merge the two sorted lists {2,3,4,7} and {2,4,5,6}
- Solution:
  - First stage, we merge even elements from the two lists {2,4} with {2,5}
  - Recursing we need to merge {2} with {2} and {4} with {5} then compare them
  - Results of the two merges are {2,2} and {4,5}
  - Comparing higher element of the first list with lower element of the second list, we determine the merged list is {2,2,4,5}
  - Next merge odd elements {3,7} with {4,6} with result {3,4} and {6,7}
  - Comparing the high and low elements we get merged list {3,4,6,7}
  - Carrying out the comparisons we get {2,2,3,4,4,5,6,7}

Putting it together- Batcher Banyan



- What about trapped duplicates?
  - re-circulate to beginning
  - or run output of trap to multiple banyans (*dilation*)

Effect of packet size on switching fabrics

- A major motivation for small fixed packet size in ATM is ease of building large parallel fabrics
- In general, smaller size => more per-packet overhead, but more preemption points/sec
  - At high speeds, overhead dominates!
- Fixed size packets helps build synchronous switch
  - But we could fragment at entry and reassemble at exit
  - Or build an asynchronous fabric
  - Thus, variable size doesn't hurt too much
- Maybe Internet routers can be almost as cost-effective as ATM switches