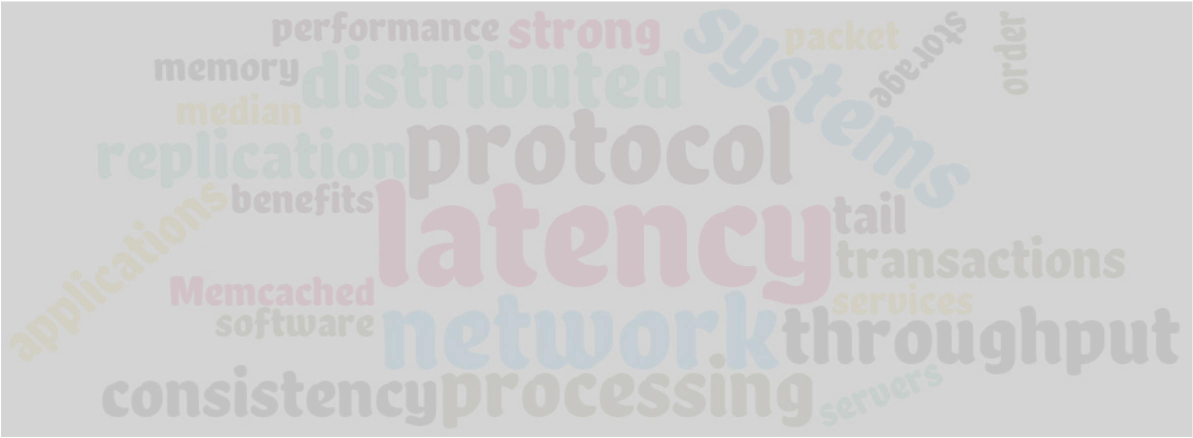


Topology Abstraction and Overlays

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this topic

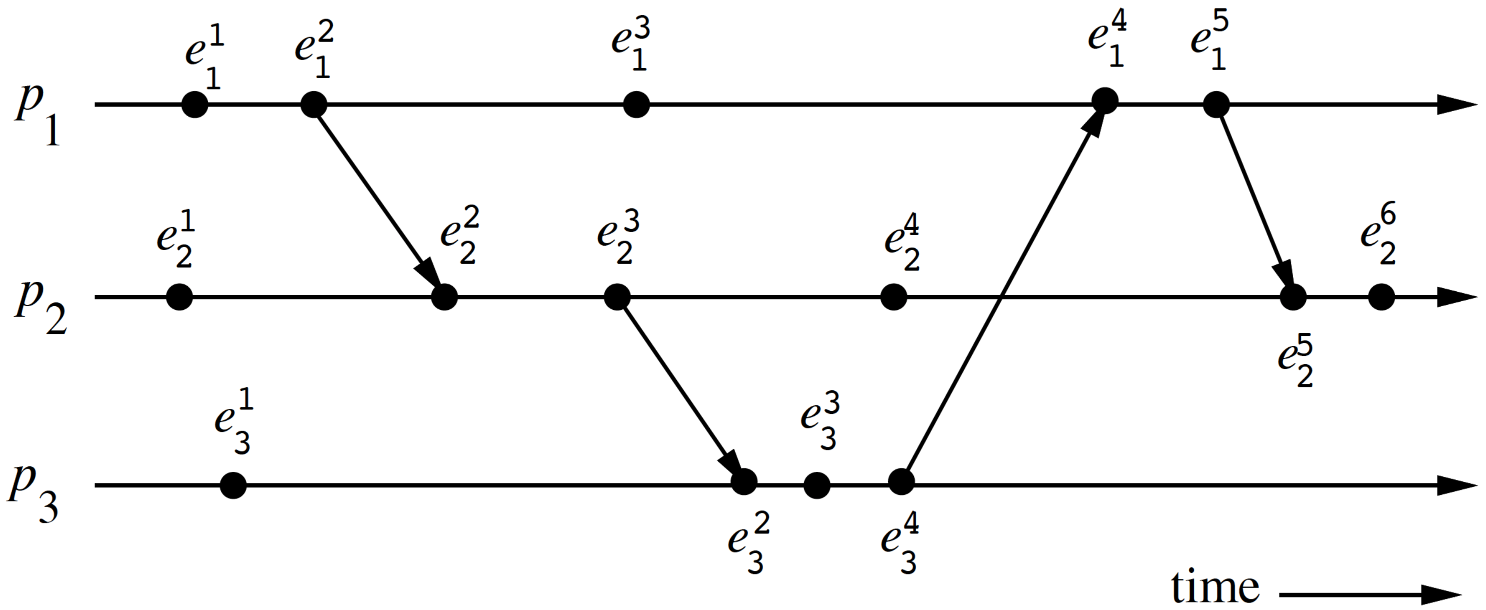
This course covers the essential aspects of **Various Interconnection Networks in Distributed Systems and its related concepts**

RECAP

What did you learn so far?

- → Challenges in Message Passing systems
- → Distributed Sorting
- → Space-Time Diagram
- → Partial Ordering / Total Ordering
- → Causal Ordering - Precedence Relations
- → Concurrent Events
- → Local Clocks and Vector Clocks
- → Distributed Snapshots
- → Termination Detection using Dist. Snapshots
- → Leader Election Problem in Rings

A State-Time diagram – An Example



- **For Process P'_1:**
  - Second event is a message send event
  - First and Third events are internal events
  - Fourth event is a message receive event

Topics to focus on …

- Leader Election in Distributed Systems

- Topology Abstraction and Overlays

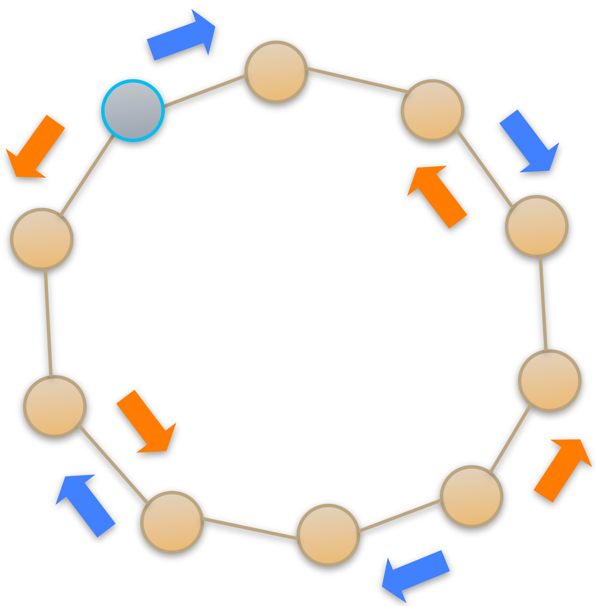
- Message Ordering
- Group Communication
- Distributed Mutual Exclusion
- Deadlock Detection
- Check pointing and rollback recovery

For Mid Semester 2

Topology Abstraction and Overlays in Distributed Systems

Ring Networks

- In an oriented ring, processes have a consistent notion of left and right



Clockwise (right)

Anti-clockwise (left)

- For example, if messages are forwarded on right channel, they will cycle clockwise around the ring

Why Study Rings?

- Simple starting point, easy to analyze
- Abstraction of a token ring
- Lower bounds and impossibility results for ring topology also apply to arbitrary topologies

Interconnection Topologies

- Various Interconnection Networks
  - Abstraction of the overall networks
  - Message Propagation
  - Distributed Processing
  - Computation Complexity
- Overlays
  - Sampling the underlying network topology

Basic Terminologies

- System Model: Undirected (weighted) graph G = (V, E), where n = |V|
  - Model the underlying topology in such a way that the pattern of message passing / communication could be efficiently handled
  - Easy to maintain and apply logics on the abstraction of the underlying topology

Physical Topology

- Physical topology
  - Nodes: network nodes, routers, all end hosts (whether participating or not)
  - Edges: all LAN, WAN links, direct edges between end hosts

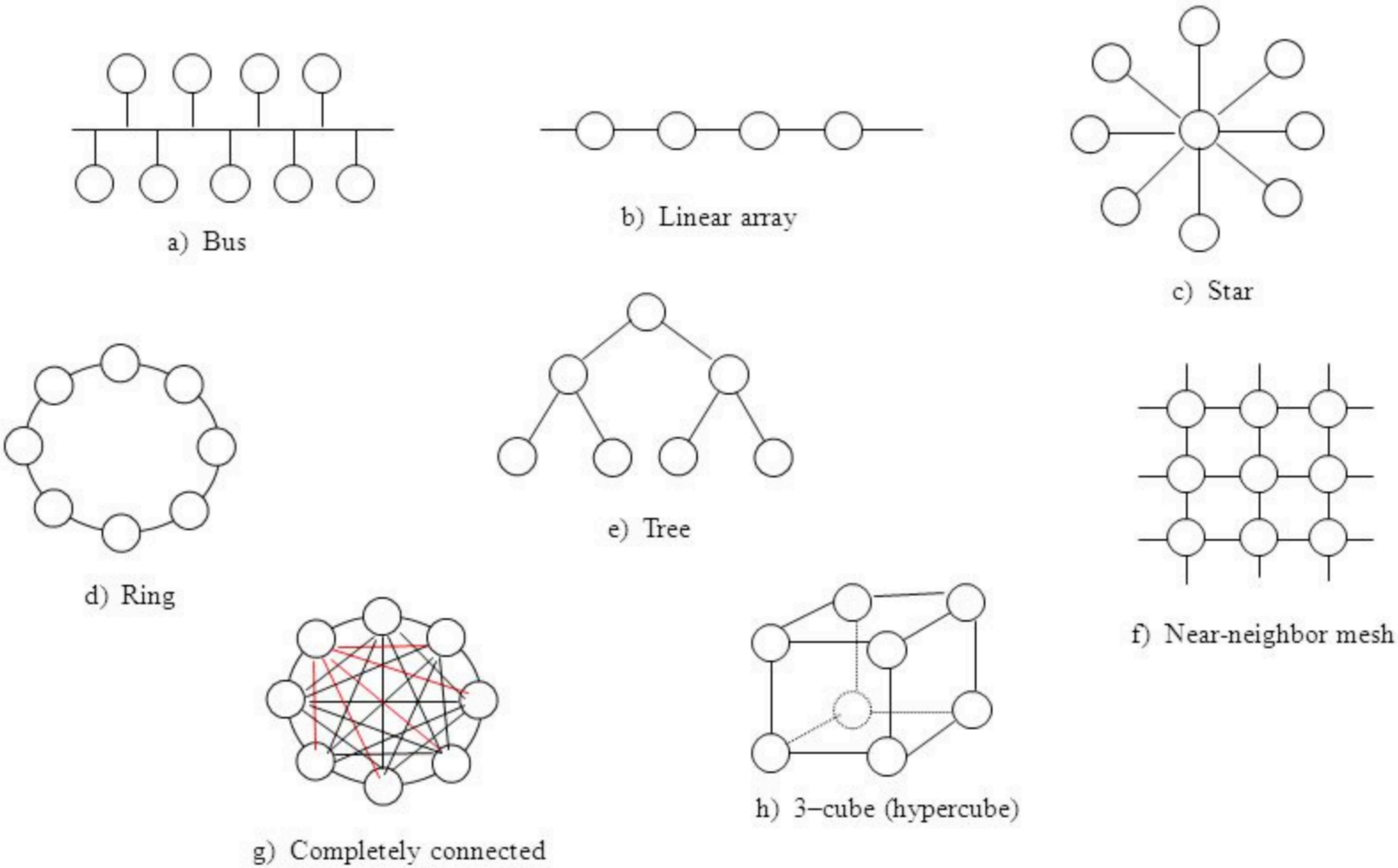
Logical Topology

- Logical topology (application context)
  - Nodes: end hosts where application executes
  - Edges: logical channels among these nodes
  - Fully connected or any subgraph – partial system view, needs multi-hop paths

Superimposed Topology

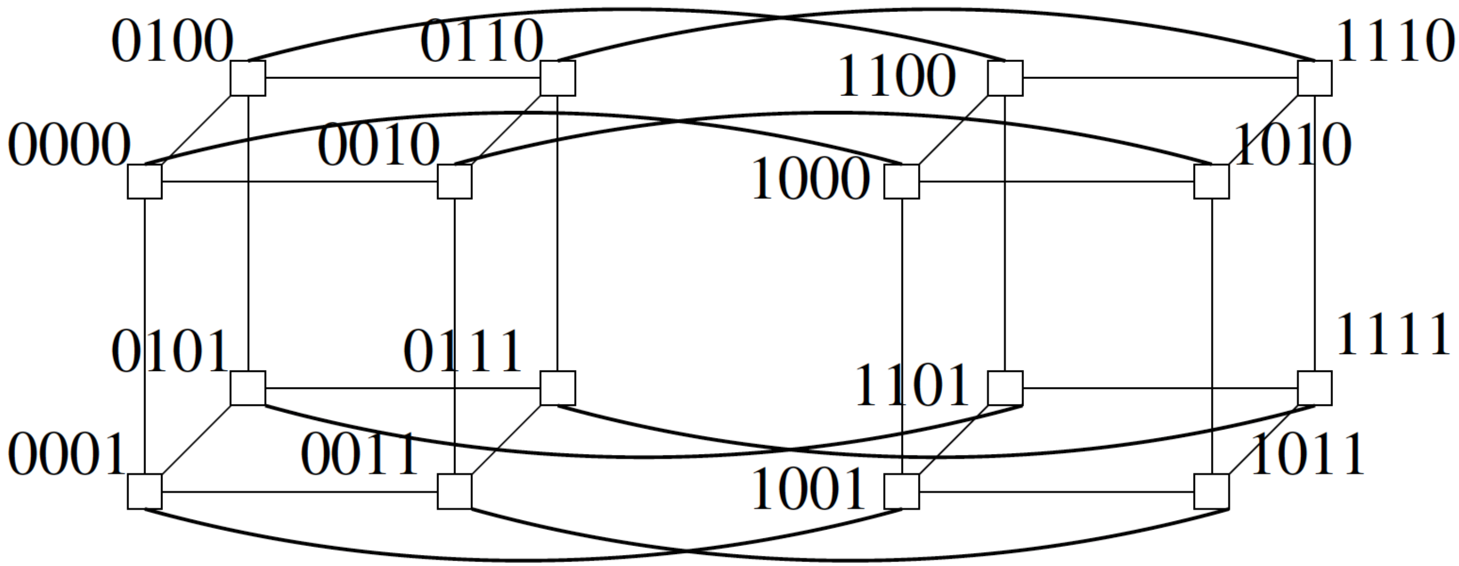
- Superimposed topology (also called as "topology overlay"):
  - superimposed on logical topology
  - Goal: efficient information gathering, distribution, or search (as in P2P overlays)
  - Examples: ring, tree, mesh, hypercube

Interconnection Topologies



Interconnection Topologies (contd)

- Hypercube of Dimension 4



- k-ary d-cudes (generalized version)

Basic Concepts

- Application execution vs. control algorithm execution, each with own events
- Control algorithm:
  - for monitoring and auxiliary functions, e.g., reaching consensus, global state detection (deadlock, termination etc.), check pointing and
  - superimposed on application execution, but does not interfere
  - its send, receive, internal events are transparent to application execution

Classifications

- Centralized and distributed algorithms
  - Centralized: asymmetric roles; client-server configuration; processing and bandwidth bottleneck; point of failure
  - Distributed: more balanced roles of nodes, difficult to design perfectly distributed algorithms (e.g., snapshot algorithms, tree-based algorithms)
- Symmetric and asymmetric algorithms

Important Concepts

- **Anonymous algorithm:**
  - Process ids are not used to make any execution (run-time) decisions
  - Structurally elegant but hard to design, or impossible, (anonymous leader election is impossible)
- **Uniform algorithm:**
  - Cannot use n, the number of processes, as a parameter
  - Allows scalability; process leave/join is easy and only neighbors need to be aware of logical topology changes
- **Adaptive algorithm:**
  - Let k ($\leq n$) be the number of processes participating in the context of a problem X when X is being executed. Complexity should be expressible as a function of k, not n
  - For example, Mutual Exclusion is a good example

Deterministic vs Nondeterministic

Deterministic vs. nondeterministic executions

- Nondeterministic exec: contains at least 1 nondeterministic receive; deterministic execution has no nondeterministic receive
  - Nondeterministic receive: can receive a message from any source
  - Deterministic receive: source is specified

Difficult to reason with

- Asynchronous system: re-execution of deterministic program will produce same partial order on events ((used in debugging, unstable predicate detection etc.)
- Asynchronous system: re-execution of nondeterministic program may produce different partial order (unbounded delivery times and unpredictable congestion, variable local CPU scheduling delays)

Synchronous vs Asynchronous

Synchronous:

- ➔ Upper bound on message delay
- ➔ Known bounded drift rate of clock with respect to the real time
- ➔ Known upper bound for process to execute a logical step

Asynchronous: above criteria not satisfied

- ➔ Spectrum of models in which some combo of criteria satisfied
- ➔ Algorithm to solve a problem depends greatly on this model

Distributed systems are inherently asynchronous

Algorithms / Channels

- Wait-free algorithms (for synchronization operations)
  - resilient to n - 1 process failures, i.e., operations of any process must complete in bounded number of steps, irrespective of other processes
  - very robust, but expensive
  - possible to design for mutual exclusion
  - may not always be possible to design, for example, the producer-consumer problem
- Communication channels
  - point-to-point: FIFO, non-FIFO
  - At application layer, FIFO usually provided by network stack

Process failures (Sync + Async syst.)

- Fail-stop: Properly functioning process stops execution. Other processes learn about the failed process (thru some mechanism)
- Crash: Properly functioning process stops execution. Other processes do not learn about the failed process
- Receive omission: Properly functioning process fails by receiving only some of the messages that have been sent to it, or by crashing.
- Send omission: Properly functioning process fails by sending only some of the messages it is supposed to send, or by crashing. Incomparable with receive omission model.
- General omission: Send omission + receive omission
- Byzantine (or malicious) failure: Process may (mis) behave anyhow, including sending fake messages. Authentication facility => If a faulty process claims to have received a message from a correct process, that is verifiable.

Process Failures (contd.)

- Process failures → Timing failures (sync systems):
  - General omission failures, or process violating bounds on time to execute a step
  - More severe than general omission failures

Failure models influence design of algorithms

- Link failures
  - Crash failure: Properly functioning link stops carrying messages
  - Omission failure: Link carries only some of the messages sent on it, not others
  - Byzantine failure: Link exhibits arbitrary behavior, including creating fake and altering messages sent on it
- Link failures → Timing failures (sync systems):
  - messages delivered faster/slower than specified behavior

Complexity Measures

- Each metric specified using
  - lower bound (Omega),
  - upper bound (big O)
  - exact bound(Theta)

Metrics

- Space complexity per node
- System-wide space complexity (= n space complexity per node). E.g., worst case may never occur at all nodes simultaneously!
- Time complexity per node
- System-wide time complexity. Do nodes execute fully concurrently?

Metrics

- Message complexity
  - Number of messages (affects space complexity of message overhead)
  - Size of messages (affects space complexity of message overhead + time component via increased transmission time)
  - Message time complexity: depends on number of messages, size of messages, concurrency in sending and receiving messages
- Other metrics: # send and # receive events; # multicasts, and implementation related metrics?
- (Shared memory systems): size of shared memory; # synchronization operations

Summary

➔ Topology Abstraction and Overlays

- ➔ Various Interconnection Topologies
- ➔ Abstraction - Basic Concepts
- ➔ Interconnection Patterns suitable for message propagation
- ➔ Types of Algorithms and their executions
- ➔ Measures and Metrics

➔ Many more to come up … stay tuned in !!

How to reach me?

- ➔ Please leave me an email:
  rajendra [DOT] prasath [AT] iiits [DOT] in

- ➔ Visit my homepage @

  - ➔ http://www.iiits.ac.in/FacPages/index-rajendra.html

  OR

  - ➔ http://rajendra.2power3.com

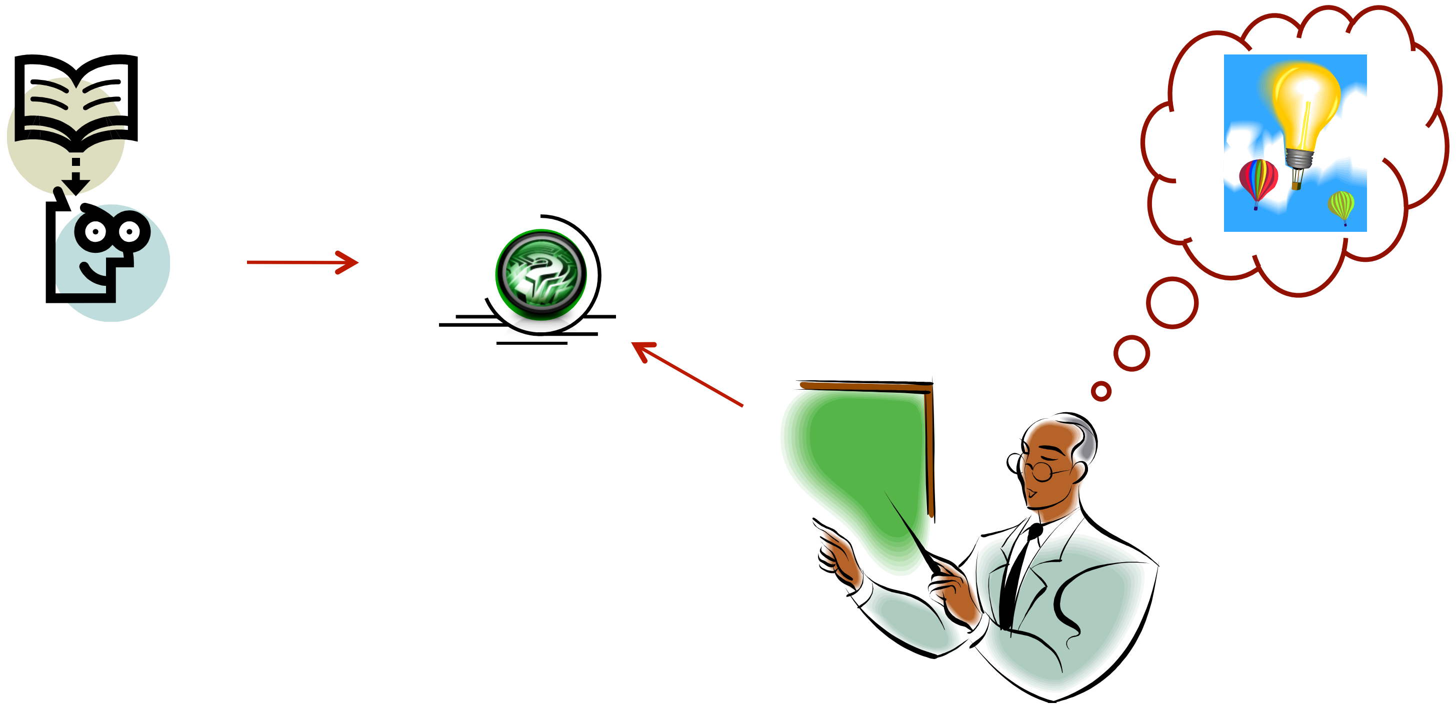
Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Thanks ...



... Questions ???