Lightweight Causal Cluster Consistency

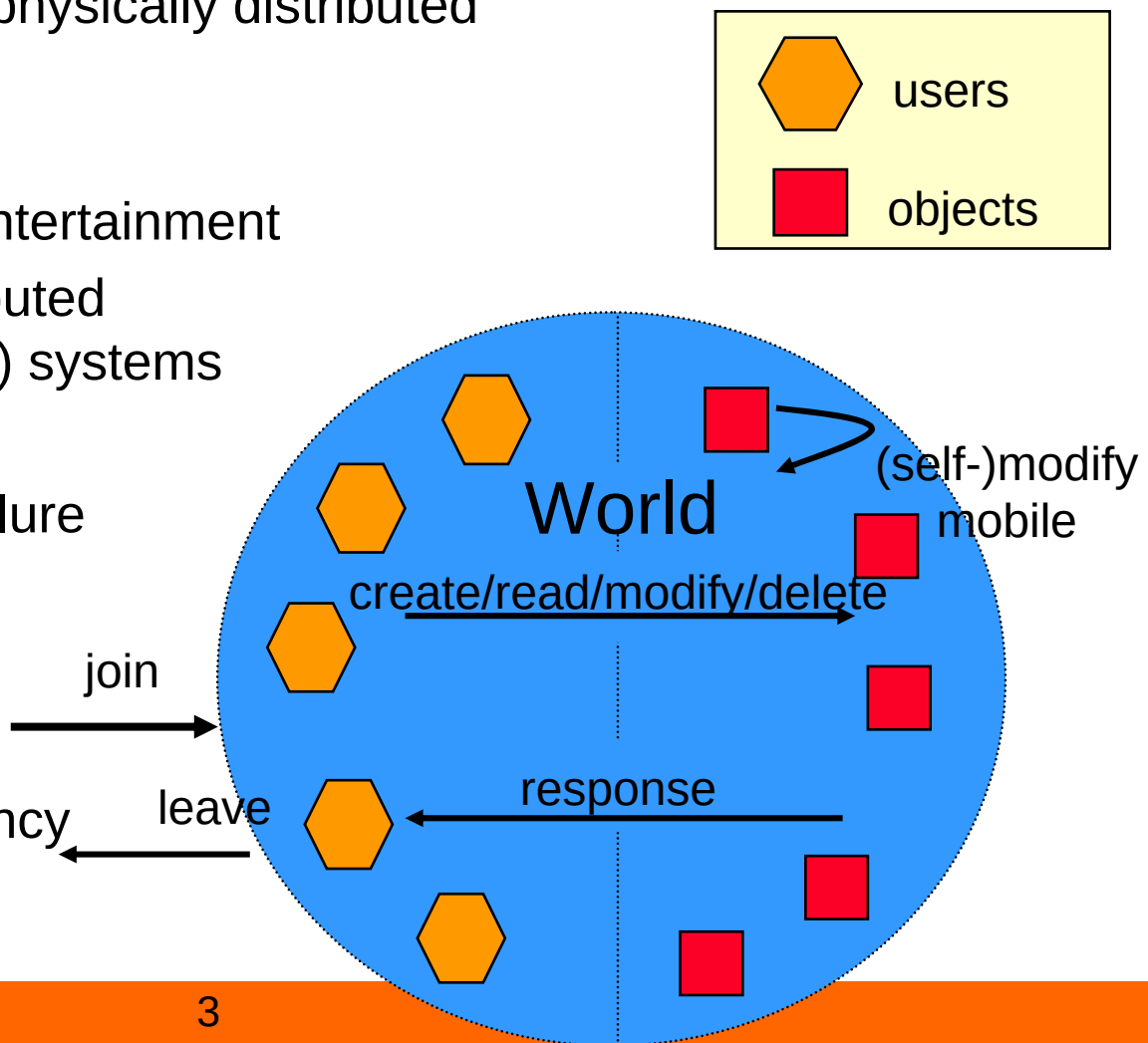
Boris Koldehofe, Anders Gidenstam,
Marina Papatriantafilou, and Philippas Tsigas

Outline

- Introduction
  - Collaborative environments
  - Problem definition
- Causal Cluster Consistency
- Protocol implementing Causal Cluster Consistency
  - Framework
  - Cluster Management
  - Dissemination and Causal delivery
  - Recovery
- Results
- Conclusion and Future Work

Collaborative Environments

- Possible applications with physically distributed "users":
  - Conferencing, CVEs
  - Simulation, Training, Entertainment
  - Administration of distributed (e.g. telecom, transport) systems
- Decentralised solution
  - Avoid single point of failure
  - Share the load evenly
  - Scalability
- Trade-off
  - Overhead vs. Consistency

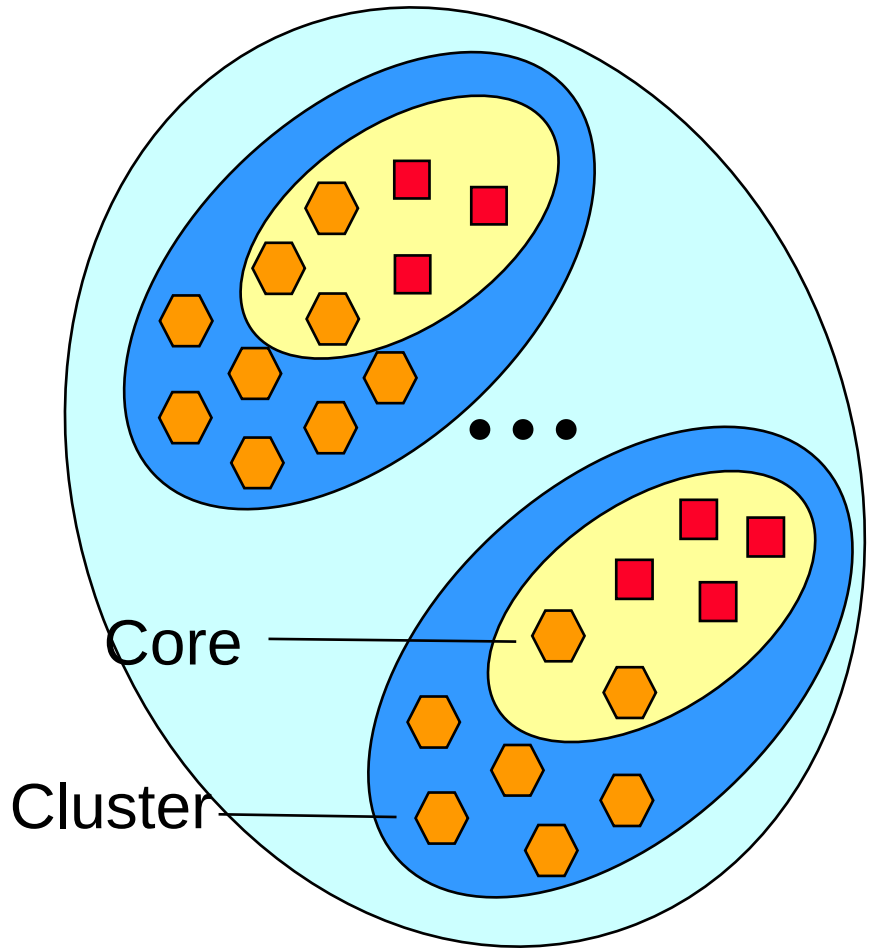


Defining the problem

- Goal: Support large Collaborative Environments
  - Provide Consistency (order of updates matter)
  - Scalable communication media
- Focus: Group communication
  - Propagate events (updates) to all interested processes
  - Ordered event delivery
    - Causal order
- Opportunities
  - Delivery with high probability is enough
  - Limited per-user domain of interest
    - Nobody is interested in changing everything at once
  - Events have lifetimes/deadlines
  - Often more observers than updaters

Example: Collaborative Environments

- World
  - Consists of Clusters
    - Consists of Objects
  - Clusters represent interest
  - Only few updaters per cluster
    - Forming the Core



Causal Cluster Consistency

- n constant known by all processes
- Given a set of clusters $C_1, \ldots, C_m$
  - Cluster corresponding to region of interest
- Processes can join and leave any cluster C_i
  - A process in C_i

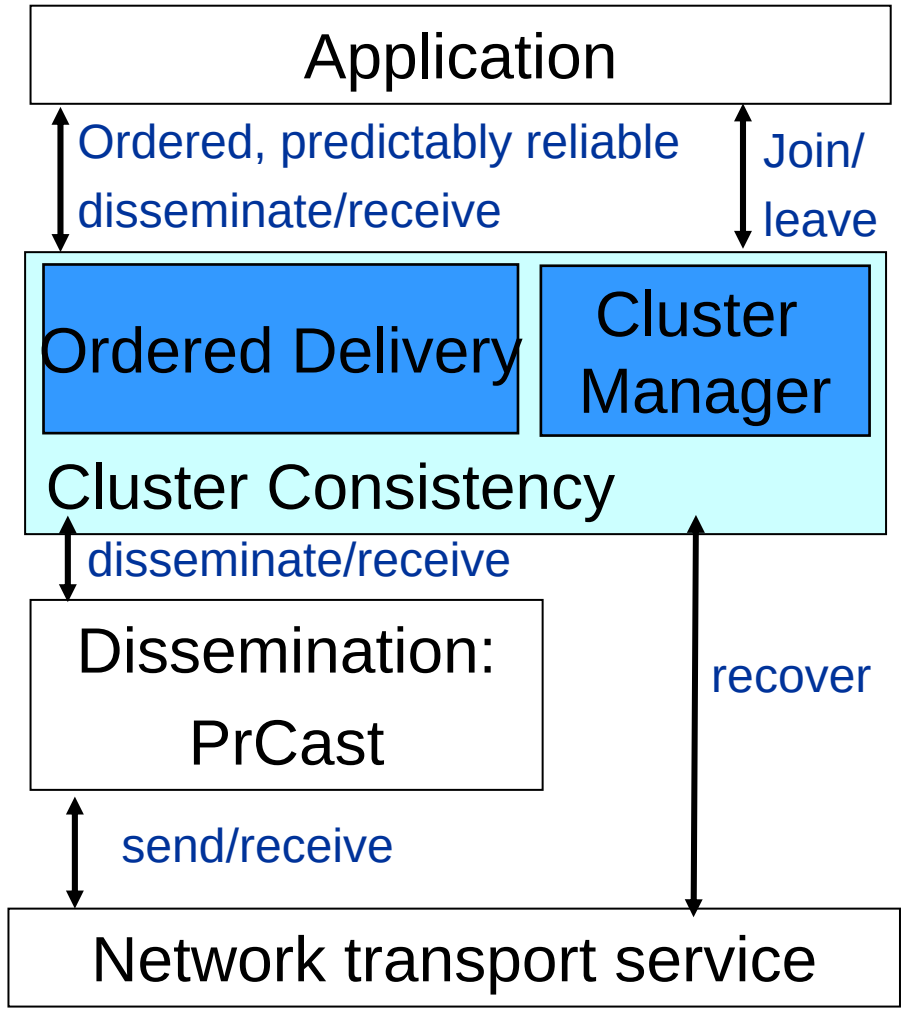
    , receives events disseminated in C_i w.h.p.
  - events can be observed in optimistic causal order
- A dynamic non-empty subset forms the core of C_i
  - at most n processes inside a core
  - Only those processes create new events

Outline

- Introduction
  - Collaborative environments
  - Problem definition
- Causal Cluster Consistency
- Protocol implementing Causal Cluster Consistency
  - Framework
  - Cluster Management
  - Dissemination and Causal delivery
  - Recovery
- Results
- Conclusion and Future Work

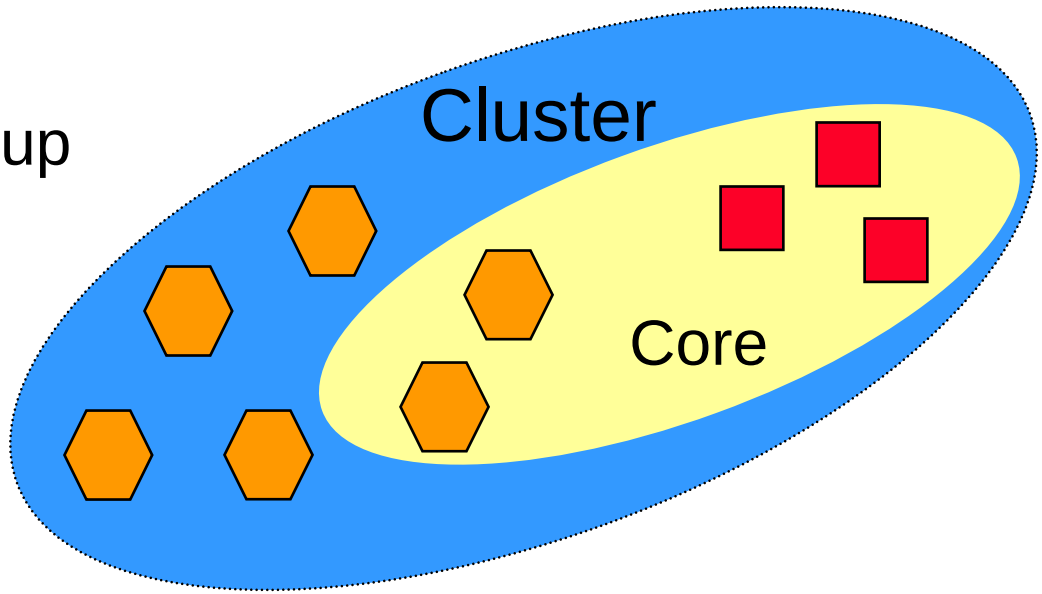
Overview: A Layered approach

- Point-2-point communication layer
- Dissemination layer
  - Gossip protocol
  - Reader membership
- Causal layer
  - Cluster Manager
    - Controls concurrent updates
  - Causal delivery
  - Recovery



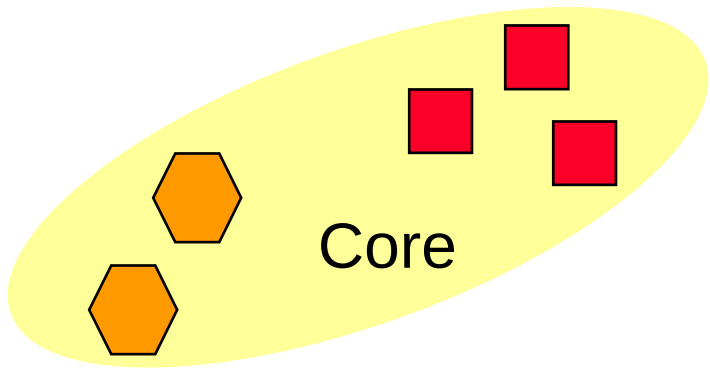
Cluster Management

- Each cluster corresponds to a process group
- Interested processes join
- Readers – everyone
  - Join the process group
- Updaters
  - At most n at a time
  - Core of the cluster



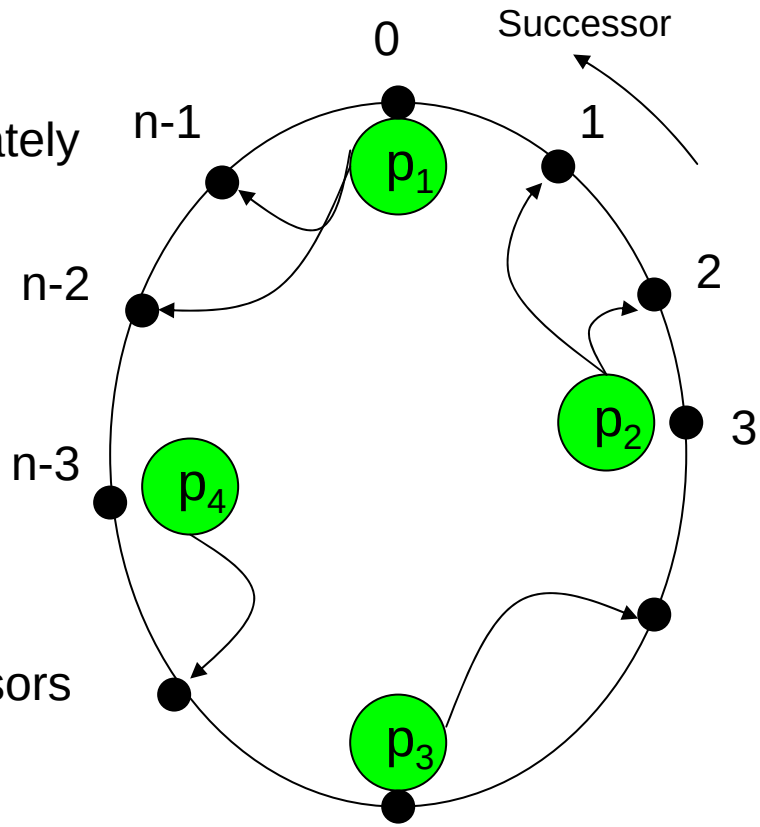
Managing the Core

- Assign unique identity for each process
  - Ids 5 {0, …, n-1}
- Two processes never own the same id
  - Even in the occurrence of failures
    - Stop failures
    - Communication failures
- Reclaim tickets



Cluster Management Algorithm

- Inspired by DHT
  - Ids form a cycle (max n)
  - Each process manage the entries immediately before it.
- Contact any coordinator to join
  - Notify successor if given an entry
  - Notify all about the new coord.
- Failure detection
  - Heartbeats
    - Send to 2k + 1 closest successors
    - Receive from 2k + 1 closest predecessors
    - If < k + 1 received, stop

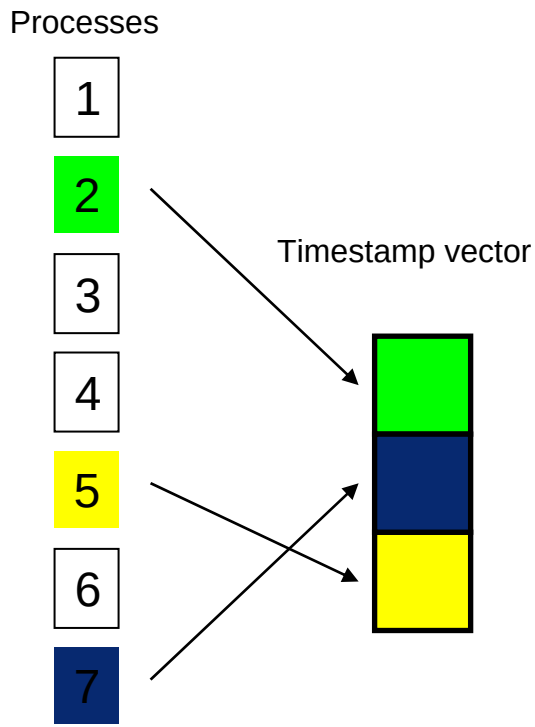


PrCast

- Gossip based protocol
  - Epidemic style dissemination
  - Good scalability and fault-tolerance
  - no ordering of events provided
- Use dissemination scheme providing delivery guarantee w.h.p.
  - W.h.p. = with probability $O(1-n^{-k})$, $k>1$.
  - Only a small number of processes is not receiving an event
    - , only few messages require recovery

Causally ordered delivery

- Vector timestamps
  - For each event in cluster
  - #simultaneous updaters limited => bounded number of vector entries in timestamps
  - ID of the cluster manager corresponds to entry in the vector clock
  - Can detect missing dependencies
  - Deliver in causal order
    - Skip events not recovered in time



Recovery

- Some events may not be delivered by PrCast
- Can detect these events with the help of the vector timestamp
  - Queue of delayed events
  - Queue of missing event ids
- A delayed event is delivered latest after a lifetime
  - ‰ Exp(time to disseminate + time to recover)
- Recovery of missing events if a delayed event has a lifetime
  - ¿ Exp(time to disseminate)

Recovery Schemes

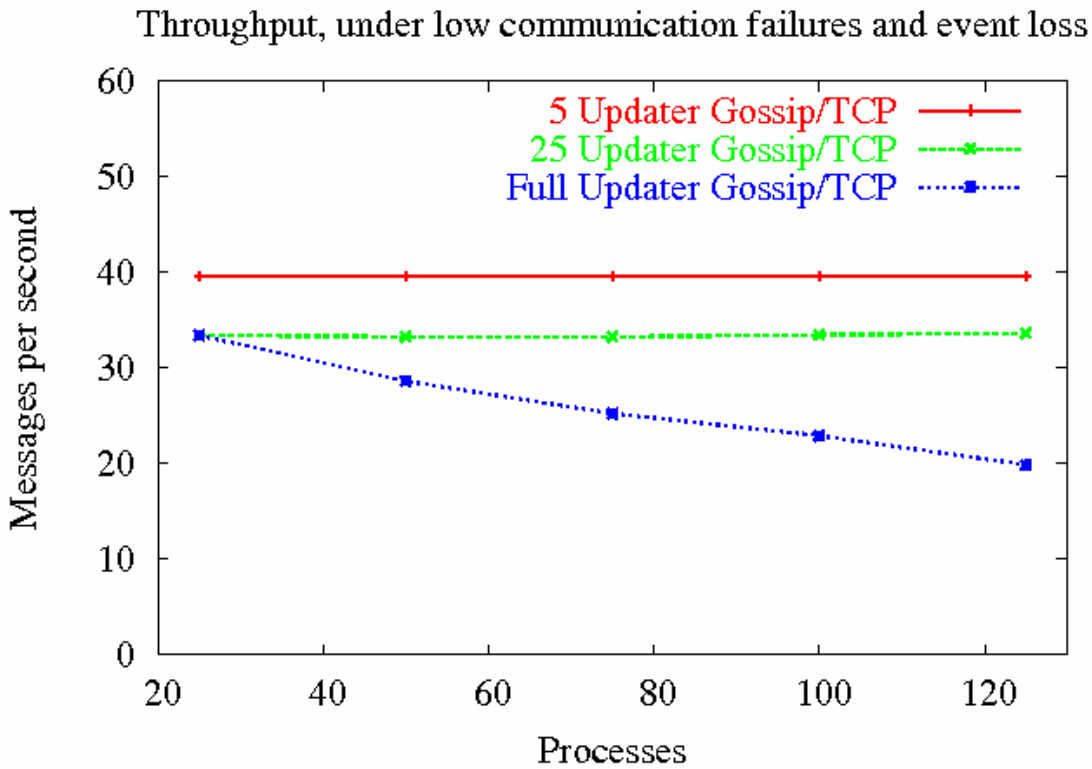
- Recover from source
  - \+ Only small buffer size needed
    - Sender buffers only own events
  - \+ Only one message per recovery
  - – Source may fail before recovery starts
  - – Too many processes may contact the source

- Alternatively recover from k peers (chosen at random)
  - Avoids problems above
  - Needs to buffer some of the received events
  - Can evaluate buffer size and k suitable for high probability recovery

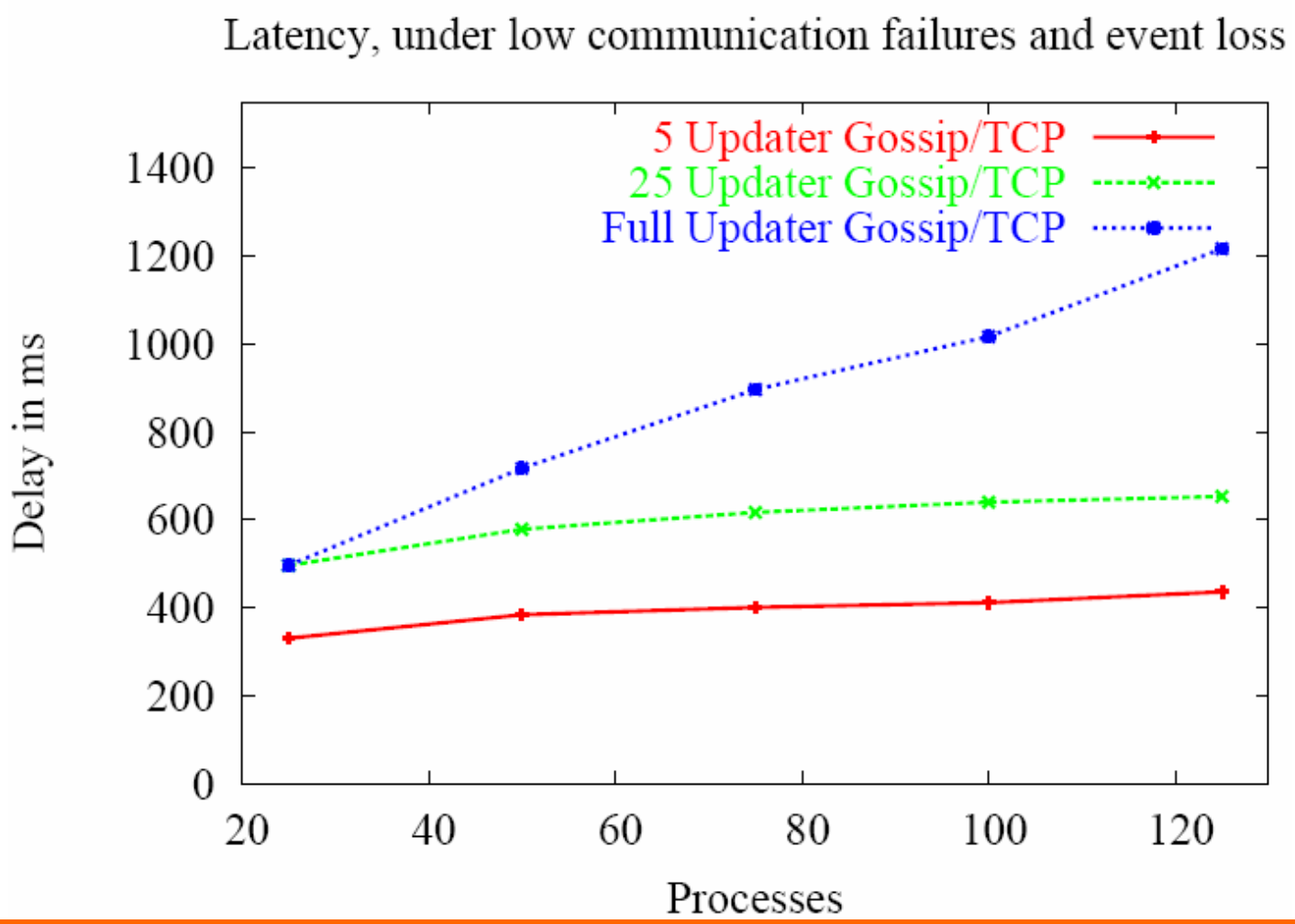
Experimental Evaluation

- Evaluate
  - Scalability
    - effect of limited number of updaters
  - Reliability
    - Measure effect of recovery schemes
- ‘‘Real network‘‘ experiment
  - Used self-implemented group communication framework
  - Test application performing on up to 125 workstations
  - Configured to provide maximum throughput and performing stable

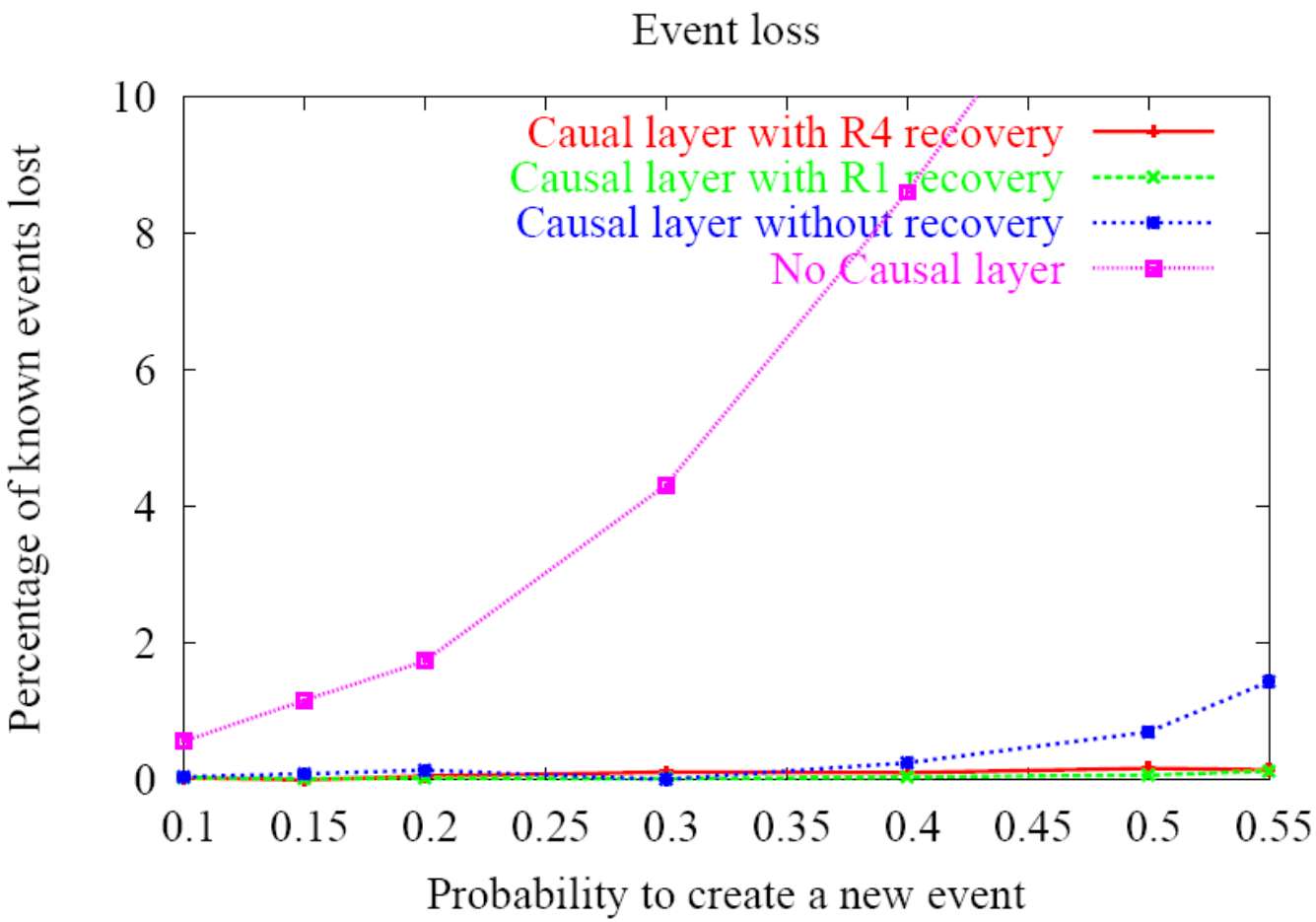
Experiments: Scalability



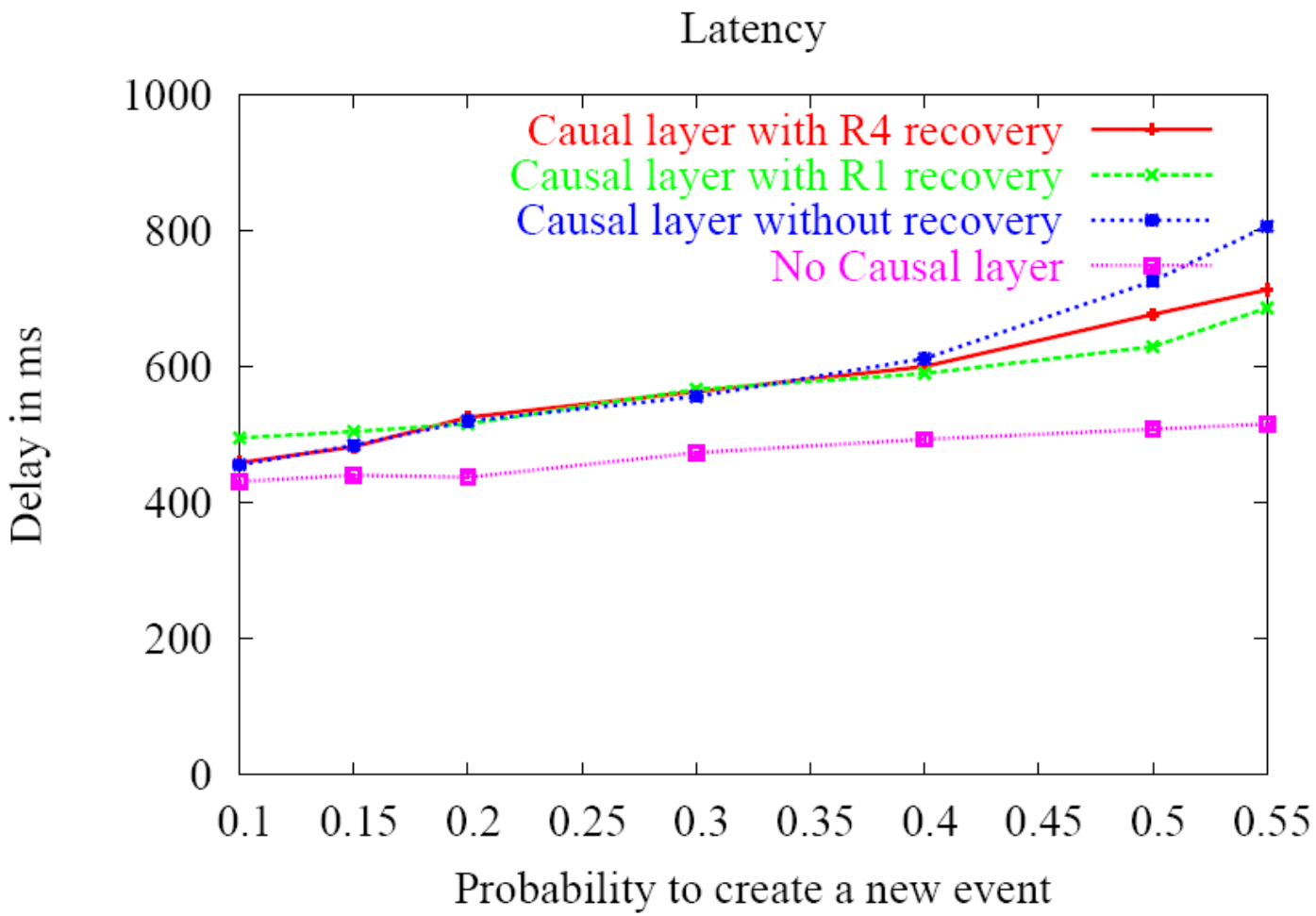
Experiments: Scalability



Experiments: Reliability



Overhead


Latency
Caual layer with R4 recovery
Causal layer with R1 recovery
Causal layer without recovery
No Causal layer
Delay in ms
1000
800
600
400
200
0
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
Probability to create a new event

Results

- Can combine predictable reliable protocols and causal delivery
- The number of concurrent updaters
  - Important for the performance
  - Scalable solutions require a bound on the number of updaters
- Recovery
  - Increases delivery rate for many concurrent events
  - Recovery fails if
    - Only few processes received the event
    - Recovered event arrives late

Conclusions and Future Work

- Causal Cluster Consistency
  - Suitable preserving optimistic causal order relations
    - Interesting for Collaborative Environments
    - Good predictable delivery guarantees
    - Scalability
  - requires a natural clustering of objects
- Recovery
  - Can increase delivery rate
  - Good match with protocols providing delivery w.h.p.
  - Source recovery (R1) vs. decentralised recovery (R4)
    - Here no real difference
    - For larger systems R4 expected to perform better
- Future work
  - Recovery for larger systems
  - Different ordering and time stamping schemes (e.g. plausible clocks)
  - Evaluate effect on dynamic systems

Recovery Success