

Maintainer-Based Replication in Unstructured P2P Systems

[Christof Leng et al, CoNEXT 2008]

Presenter: Shahin Kamali

s3kamali@cs.uwaterloo.ca

David Cheriton School of Computer Science

University of Waterloo

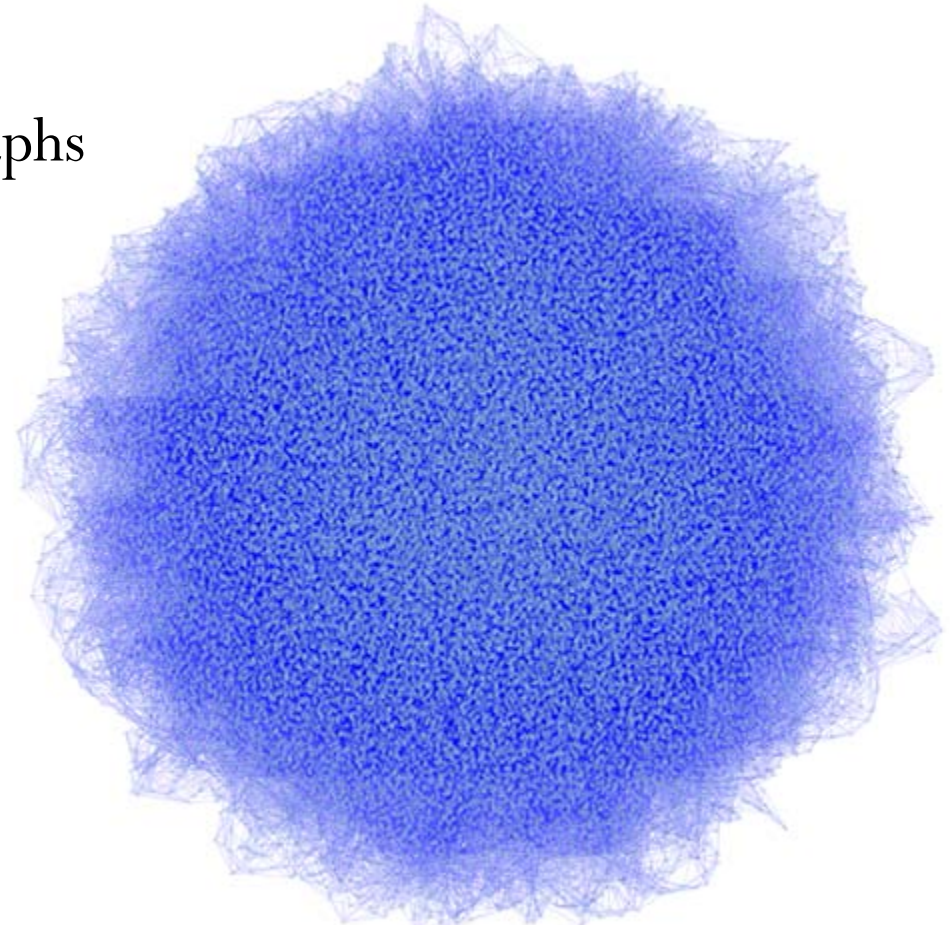
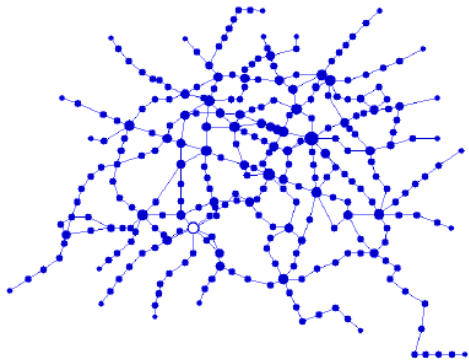


Road Map

- Unstructured P2P systems
 - Challenges and tools
- Problem statement
 - Maintainer based P2P systems
- Design
 - Maintaining (and adjusting) replica density
 - Controlling junk
- Extensions to the model
- Simulation results
- Conclusion and feature works

Unstructured P2P Systems

- In large network structure is ignored (unstructured P2P)
- Nodes are almost 'blind'
- Modelled by complete graphs
 - Mathematical analysis

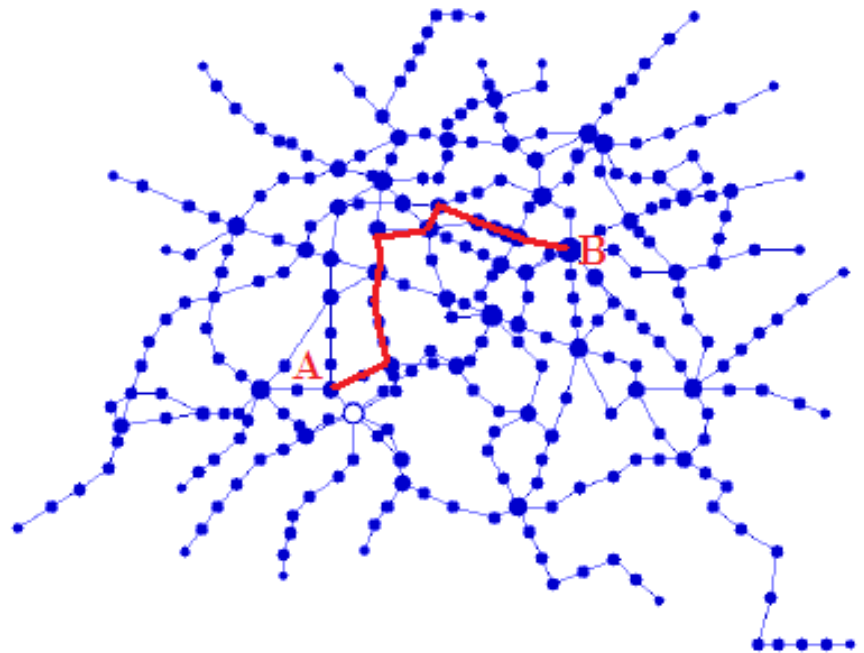


P2P Modeling (Challenges)

- Peers join and leave very frequently
 - Median life time of a peer is 60 minutes [Gnutella by Saroiu]
- High possibility of failure
 - Peer failure
 - Network transient failure
- Limited understanding of the environment
 - No idea on network statistics (network size, diameter, etc..)

P2P Modeling (Tools)

- **Ping mechanism**
 - Detects peer failure
 - Test weather a particular peer is accessible across the network
 - Peers have unique addresses (IPs)
- Not reliable
 - Network transient failure
- Pretty expensive



P2P Modeling (Tools)

- **Gossip mechanism** → Accumulate data (network statistics)
 - Each person has an independent piece of a gossip (peers and data)
 - They use telephone to exchange data (no parallelism in informing neighbours)
 - If everybody has phone number of every-body (complete graph modeling), data is accumulated in exponential time
- Unstructured network with a million nodes
 - 5 minutes time for data accumulation

P2P Modeling (Tools)

- Random walks
 - Allows choosing a random node of the network
 - Provides an IP address
- Push mechanism
 - Connections are initiated by the source to send data into destinations
- Pull mechanism
 - Connections are initiated by the destinations to received data from sources

Replication in P2P networks

- *Recall* → blind search
 - Bad query response, high traffic, etc
 - Use replication ...
- Example:
 - A P2P network with one million nodes
 - Replicate an object on 1000 nodes (replication degree)
- *Recall* → peer leave frequently (voluntarily or by failure)
 - After a while, most of these 1000 nodes are gone !
 - They should be replaced ...

Replication in P2P networks

- *Extreme condition* → After a while, all peers are gone and new peers have replaced them
 - Old peers leave and old replicas stay !
 - Is it realistic always?
- Sometimes replicated objects have ties with specific peers
 - Ex: List of service provided by a specific host (peer)
 - If the host leaves, replicas of the list should be removed

Replication maintain strategies

- Collective replication
 - Replicas remain in the system no matter which peers are up
 - All peers are responsible to achieve replication degree (# of replicas)
 - This approach is intensively studied
- Maintainer based replication
 - Each replicated object is assigned a maintainer
 - Maintainer decides on the replication degree (number of replicas)
 - When maintainer of an object leaves, the object's replicas should be removed
 - A novel approach !

Problem statement

- Create a maintainer-based replication model that:
 - Keeps the replication degree at the desired level (1000)
 - Dynamically adjusts the replication degree (1000→2000)
 - Eliminates replicas of offline maintainers
- *Recall* → difficulties
 - Peers join and leave very frequently
 - Peer failure
 - Network transient failure

Terminology

- d_i : replication degree of object i (number of replicas)
- n : size of network (number of peers)
- p_i : replication density \rightarrow percentage of peers that should have a replica of object I
 - $P_i = d_i / n$ ex: $1000 / 1000000 = 1 / 1000$
- To achieve good performance,
 - system is highly replicated
 - $d \approx 1 / \sqrt{n}$ (decided by maintainer)

Problem statement

- A solution should have strategies for
 - When maintainer joins (very beginning)
 - A new object comes to system
 - When peers leave (voluntarily or by failure)
 - May have a replica
 - When peers join
 - May need to get a replica
 - When maintainer adjusts the replication degree (1000→2000)
 - When maintainer leaves
 - When maintainer fails

A probabilistic approach

- The expected value of the replication degree is $d_i(1000)$
 - may be more or less
- The probability a peer has a replica of object i is p_i
 - To achieve desired replication degree fix expected density p_i
 - $p_i = d_i / n$ ($1000 / 100000 = 1\%$)

When maintainer joins

- *Recall* → the goal is to
 - Keeps the replication degree at the desired level ($d=1000$)
 - Dynamically adjusts the replication degree ($1000 \rightarrow 2000$)
 - Eliminates replicas of offline maintainers
- New maintainer sends d replicas into the network
 - *Recall* → random walks (Choose d random peers)
 - *Recall* → Push mechanism (push replicas to these peers)
 - Also push the address of maintainer

What we covered?

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure)
 - When peers join
 - When maintainer adjusts the replication degree
 - When maintainer leaves
 - When maintainer fails

When a peer leaves

- Possible solution:
 - Inform maintainers of their leave (to replicate on another peer)
- Difficulties when peer fails
 - No time for informing others of a failure
 - Network transient failure
- When a peer leaves (by failure or voluntarily)
 - Do nothing !
 - Expected density p_i does not change

What we covered?

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure) ✓
 - When peers join
 - When maintainer adjusts the replication degree
 - When maintainer leaves
 - When maintainer fails

Maintaining replica density

- When a peer joins
 - Object o should have a replica with probability p_o
 - Pull replicas from k random maintainers (out of m)

$$\mathbf{P}(\text{maintainer of } o \text{ not pulled}) = \left(1 - \frac{1}{m}\right)^k = 1 - p$$


$$k = \ln(1-p) / \ln(1-1/m)$$

- For another object p' , we may have a different k'
 - Ex: $m = 1000$, $p = 1/100$ gives $k \approx 10$
 $p' = 5/100$ gives $k \approx 51$
 - Send 51 pull request with index i
 - Maintainer of object with $k=10$ ignore request with $i > 10$

Maintaining replica density

- When a peer joins

```
// updated by underlying system:  
m = sum(1) // sum over all maintainers  
n = sum(1) // sum over all peers  
maxp = max(p) // max over all densities
```



Through Gossiping

```
f(p): return ceil(ln(1-p)/ln(1-1/m))
```

```
Upon joining network: // by joining peer  
x = f(maxp);  
for i in [0, x):  
    send pull(i, myAddress) to random maintainer;
```

```
Upon receiving pull(i, addr): // by maintainer  
for obj in objects_maintained:  
    if i < f(obj.get_p()):  
        send(obj) to addr
```

Maintaining replica density

- For any sequence of peer join and crash/leave
 - Replication distribution for an object with density p converges to the binomial distribution $B_{n,p}$ [theoretically proved]

- *Recall* → the goal is to
 - Keeps the replication degree at the desired level ($d=1000$)
 - Dynamically adjusts the replication degree ($1000 \rightarrow 2000$)
 - Eliminates replicas of offline maintainers

What we covered?

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure) ✓
 - When peers join ✓
 - When maintainer adjusts the replication degree
 - When maintainer leaves
 - When maintainer fails

Adjusting the replication degree

- When maintainer wants to **increase** density from p to q
 - Push x replicas to the network where
 - m is the number of replicas
- When maintainer wants to **decrease** density

$$x = \frac{\ln(1-q) - \ln(1-p)}{\ln(1 - \frac{1}{m})}$$

- Each maintainer has a list of peers with a replica of the object
- Use ping to detect these peers are still alive
- Maintainer randomly chooses the peers that should remove the replica

Maintaining replica density

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure) ✓
 - When peers join ✓
 - When maintainer adjusts the replication degree ✓
 - When maintainer leaves
 - When maintainer fails

Maintainers: Delete on Leave

- *Recall* → maintainers have a list of peers with replicas
- When a maintainers is leaving voluntarily
 - Asks the peers to remove associated replicas
 - It may not succeed through network failures
 - Junk would be created
 - Still ok (junks would be removed periodically)

What we covered?

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure) ✓
 - When peers join ✓
 - When maintainer adjusts the replication degree ✓
 - When maintainer leaves ✓
 - When maintainer fails

Controlling junk

- When maintainer fails
 - The associated replicas should be removed
 - *Recall* → valid objects versus junk
- Peers can use ping to check if maintainers are alive
 - Unreliable: network failure could cause the peer to incorrectly conclude that the maintainer is down
 - This false deduction will never be corrected
 - Long lived objects lose replicas in long time

Controlling junk

- Flush all replicas from long-lived peers
 - Peers re-execute the replica loading algorithm
- Is it feasible?
 - Objects are relatively small (ex. service lists)
 - Most peers are not long-lived comparing with maintainers

Controlling junk

- When to flush?
- Each peer flushes when its junk level is more than a threshold
 - Estimates the number of replicas it is supposed to have

$$D = \sum_o p_o$$

- Knows the number of replicas it has
 - Estimates the junk percentage !
- Flushing is cheap !
- Comparing with an imaginary optimum strategy

$$\text{Percentage of wasted transfers} \approx \frac{c}{2/g - 2}$$

percentage of
crushing maintainers

Expected percentage
of valid replicas

What we covered?

- A solution should have strategies for
 - When maintainer joins (very beginning) ✓
 - A new object comes to system
 - When peers leave (voluntarily or by failure) ✓
 - When peers join ✓
 - When maintainer adjusts the replication degree ✓
 - When maintainer leaves ✓
 - When maintainer fails ✓

Some extensions

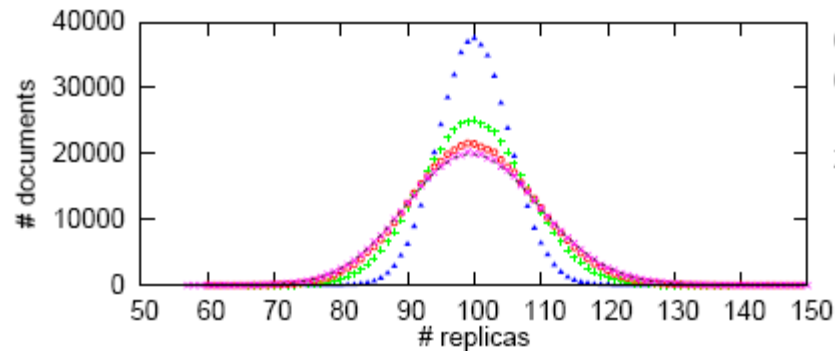
- Update in place
 - Maintainer uses its list to refresh all replicas with an update
 - The list is updated by ping
 - it may not contain some replicas due to network failure
 - temporarily inconsistent replicas (before flushing)
- Heterogeneity
 - Peers are associated different capacity
 - Just a couple of modifications ...
- Clients behind a NAT
 - Can establish connection (push)
 - Other nodes can not directly connect to them
 - Random walks can find clients to pull → Clients can be maintainers

Simulations...

- How important is simulation for judging this system?
- Assumptions
 - When a node's lifetime expires, it is replaced by a new node
 - Nodes have median lifetime of 60 minutes
 - distinguishes between peers and maintainers
 - half a million peers and half a million maintainers
 - each maintainer provides one object with density $p = 0.02\%$

Simulation results

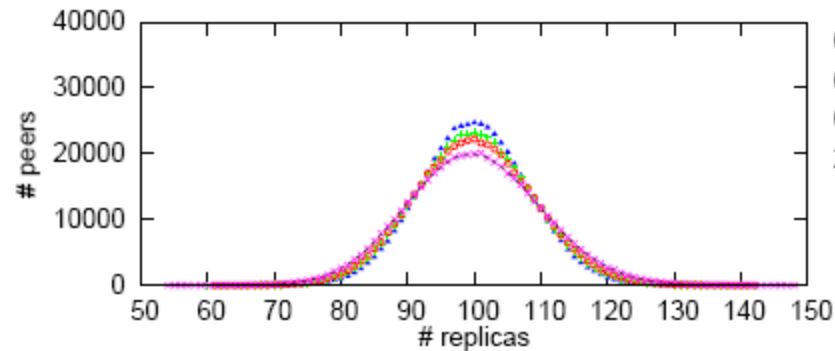
- Convergence under theoretical conditions:



Object replication

Maintainer
always online

Peers join
and leave



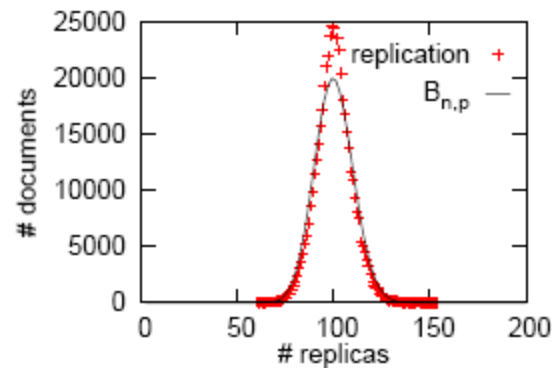
Peer load

Maintainer
leave cleanly

Peers are
always online

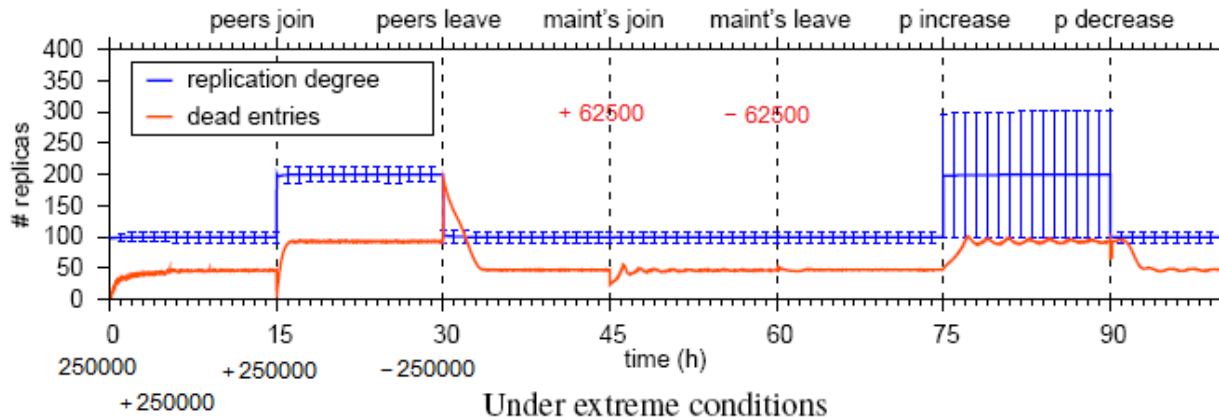
Simulation results

- Replication in steady-state and under massive peer, maintainer, and density changes



Goodness = 80%
 Gossip = 5 min
 Objects = 500000
 Maintainers = 125000
 Maintainers fail ratio = 50%

Realistic convergence

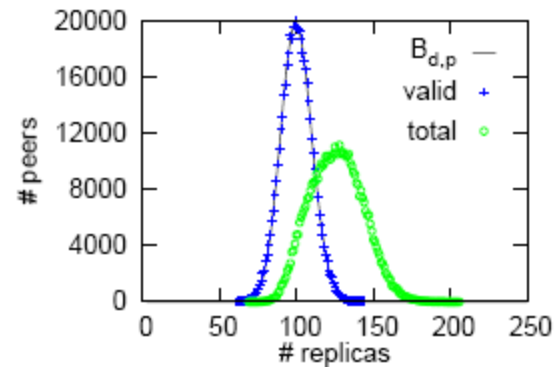


Average replicas per object

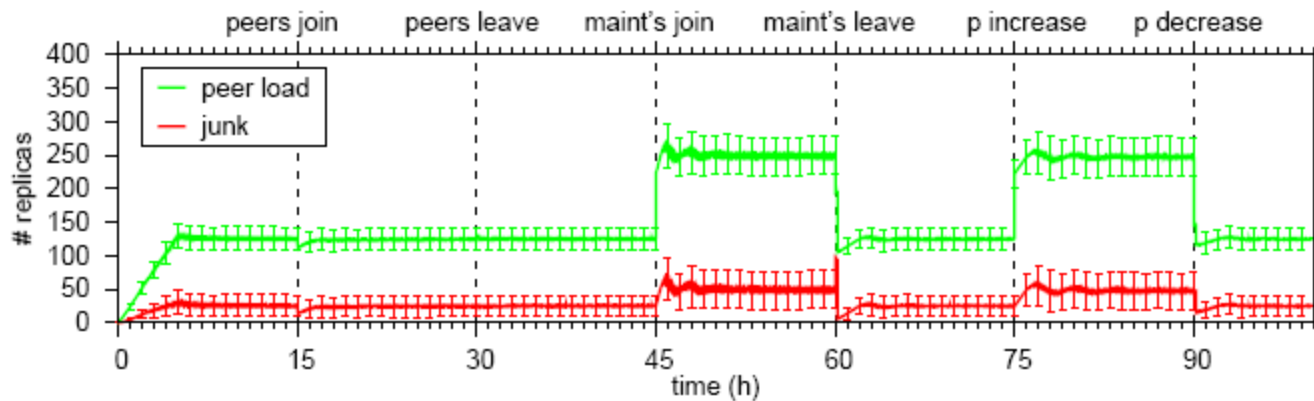
Under extreme conditions

Simulation results

- Peer load in steady-state and under massive peer, maintainer, and density changes



Realistic convergence



(b) Under extreme conditions

Conclusion and future work

- A new replication method is presented in which each replicated object is assigned a *maintainer*
- Effective strategies are presented which maintain the desired replication ratio considering network failure and peers' failure
- No future work is presented !
- A hybrid strategy
 - Maintainers have the option to unloose their replicas
- We may consider the same replication strategy when network is structured and still large

Thank you

Question on this diagram?

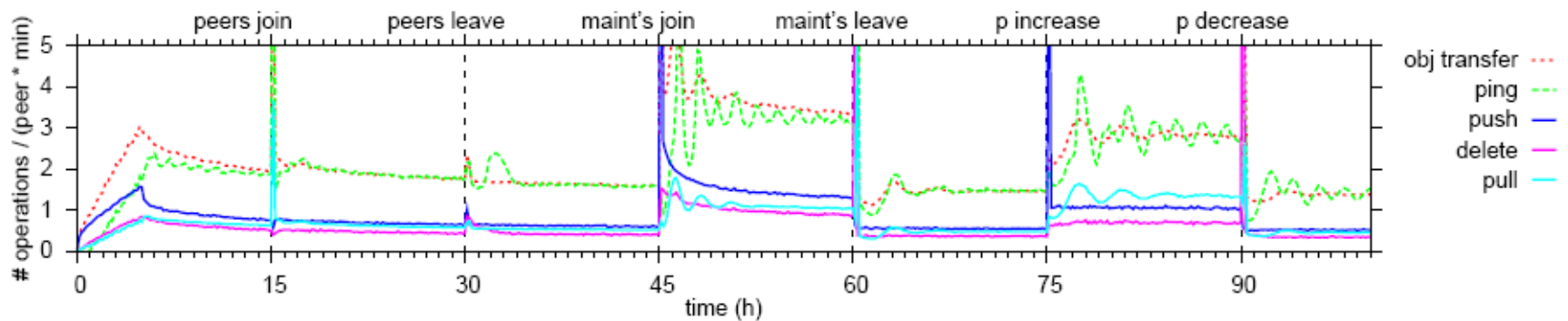


Figure 5: Network operations under extreme conditions

Question on this equation?

$$\begin{aligned} \mathbf{P}(o \text{ loses replica} \mid v \text{ flushes}) &= \mathbf{P}(v \text{ stores } o \mid v \text{ flushes}) \\ &\stackrel{*}{=} \mathbf{P}(v \text{ stores } o) = p \end{aligned}$$