Epidemic Algorithms

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

What is the Problem?

What is the Problem?

- Application-level broadcast/multicast

What is the Problem?

- Application-level broadcast/multicast
  - Database replication
  - Video streaming
  - RSS feeds
  - ...

Possible Solutions

- Flooding
  - Robust, but inefficient $(O(n^2))$

Possible Solutions

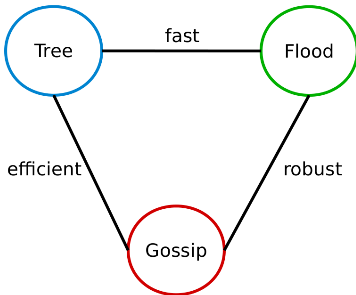
- Flooding
  - Robust, but inefficient ($O(n^2)$)
- Tree
  - Efficient ($O(n)$), but fragile

Possible Solutions

- Flooding
  - Robust, but inefficient ($O(n^2)$)
- Tree
  - Efficient ($O(n)$), but fragile
- Gossip
  - Efficient ($O(nlogn)$) and robust, but has relative high latency

Possible Solutions

- Flooding
  - Robust, but inefficient ($O(n^2)$)
- Tree
  - Efficient ($O(n)$), but fragile
- Gossip
  - Efficient ($O(nlogn)$) and robust, but has relative high latency



Epidemic/Gossip Algorithms

Introduction

- Epidemiology studies the spread of a disease or infection in terms of populations of infected/uninfected individuals and their rates of change.

Introduction

- Epidemiology studies the spread of a disease or infection in terms of populations of infected/uninfected individuals and their rates of change.

- Nodes infect each other trough messages.

- Total number of messages is less than $O(n^2)$.

- No node is overloaded.

Introduction

- Epidemiology studies the spread of a disease or infection in terms of populations of infected/uninfected individuals and their rates of change.

- Nodes infect each other trough messages.

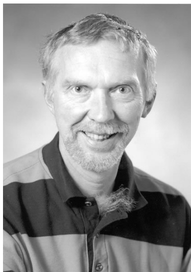
- Total number of messages is less than $O(n^2)$.

- No node is overloaded.

- But
  - No deterministic guarantee on reliability.
  - Only probabilistic ones.

History of the Epidemic/Gossip Paradigm

- First defined by Alan Demers et al. (1987)
- 90s: gossip applied to the information dissemination problem
- 00s: gossip beyond dissemination
- 2006: Workshop on the future of gossip (Leiden, the Netherlands)



History of the Epidemic/Gossip Paradigm (1987)

- Database replicated at thousands of nodes.

History of the Epidemic/Gossip Paradigm (1987)

- Database replicated at thousands of nodes.
- Heterogeneous and unreliable network.

History of the Epidemic/Gossip Paradigm (1987)

- Database replicated at thousands of nodes.
- Heterogeneous and unreliable network.
- Independent updates to single elements of the DB are injected at multiple nodes.

History of the Epidemic/Gossip Paradigm (1987)

- Database replicated at thousands of nodes.
- Heterogeneous and unreliable network.
- Independent updates to single elements of the DB are injected at multiple nodes.
- Updates must propagate to all nodes or be supplanted by later updates of the same element.

History of the Epidemic/Gossip Paradigm (1987)

- Database replicated at thousands of nodes.
- Heterogeneous and unreliable network.
- Independent updates to single elements of the DB are injected at multiple nodes.
- Updates must propagate to all nodes or be supplanted by later updates of the same element.
- Replicas become consistent after no more new updates.

History of the Epidemic/Gossip Paradigm (Today)

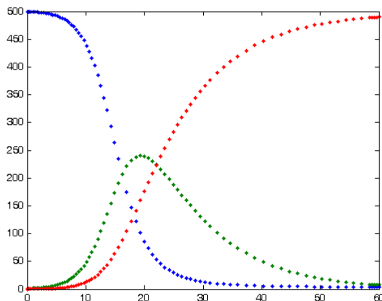
- Amazon uses a gossip protocol to quickly spread information throughout the S3 system.
- Amazon's Dynamo uses a gossip-based failure detection service.
- The basic information exchange in BitTorrent is based on gossip.

SIR Model (1/2)

- Kermack and McKendrick, 1927
- An individual p can be:
  - Susceptible: if p is not yet infected by the disease.
  - Infective: if p is infected and capable to spread the disease.
  - Removed: if p has been infected and has recovered from the disease.

SIR Model (2/2)

- Initially, a single individual is infective.
- Individuals get in touch with each other, spreading the disease.
- Susceptible individuals are turned into infective ones.
- Eventually, infective individuals will become removed.



From Epidemiology to Distributed Systems

- The idea
  - Disease spread quickly and robustly.
  - Our goal is to spread an update as fast and as reliable as possible.
  - Can we apply these ideas to distributed systems?

From Epidemiology to Distributed Systems

- The idea
  - Disease spread quickly and robustly.
  - Our goal is to spread an update as fast and as reliable as possible.
  - Can we apply these ideas to distributed systems?

- SIR Model for database replication:
  - Susceptible: if p has not yet received an update.
  - Infective: if p has not yet received an update.
  - Removed: if p has the update but is no longer willing to share it.

Two Styles of Epidemic Protocols

- Anti-entropy
- Rumor mongering

Anti-Entropy

- Each node p periodically contacts a random partner q selected from the current population.

Anti-Entropy

- Each node p periodically contacts a random partner q selected from the current population.

- Then, p and q engage in an information exchange protocol, where updates known to p but not to q are transferred from p to q (push), or vice-versa (pull), or in both direction (push-pull).

Rumor Mongering

- Nodes are initially ignorant.

Rumor Mongering

- Nodes are initially ignorant.
- When an update is learned by a node, it becomes a hot rumor.

Rumor Mongering

- Nodes are initially ignorant.
- When an update is learned by a node, it becomes a hot rumor.
- While a node holds a hot rumor, it periodically chooses a random node from the current population and sends (pushes) the rumor to it.

Rumor Mongering

- Nodes are initially ignorant.
- When an update is learned by a node, it becomes a hot rumor.
- While a node holds a hot rumor, it periodically chooses a random node from the current population and sends (pushes) the rumor to it.
- Eventually, a node will lose interest in spreading the rumor.

Epidemic Algorithms Applications

- Aggregation
- Peer sampling (Cyclon)
- Topology management (Tman)
- ...

Aggregation

Aggregation

- Aggregation provides a summary of some global system property.

Aggregation

- Aggregation provides a summary of some global system property.
- It allows local access to global information.

Aggregation

- Aggregation provides a summary of some global system property.
- It allows local access to global information.
- Examples of aggregation functions:
  - The average load of nodes in a cluster.
  - The sum of free space in a distributed storage.
  - The total number of nodes in a P2P system.

Aggregation Generic Framework (1/3)

- Executed by all processes:

```
repeat evert t time units:
  q = selectRandomPeer() // Select a random neighbor
  send <p, pullRequest, Sp> to q
```

Aggregation Generic Framework (2/3)

- Executed by all processes:

```
upon receive<p, pullRequest, Sp> do:
  send <q, pullResponse, Sq> to p
  Sq = update(Sp, Sq)
```

Aggregation Generic Framework (2/3)

- Executed by all processes:

```
upon receive<p, pullRequest, Sp> do:
  send <q, pullResponse, Sq> to p
  Sq = update(Sp, Sq)
```

- `update` function:
  - avg: return `(Sp + Sq) / 2`
  - max: return `max(Sp, Sq)`

Aggregation Generic Framework (3/3)

- Executed by all processes:

```
upon receive<q, pullResponse, Sq> do:
  Sp = update(Sp, Sq)
```

Aggregation Generic Framework (3/3)

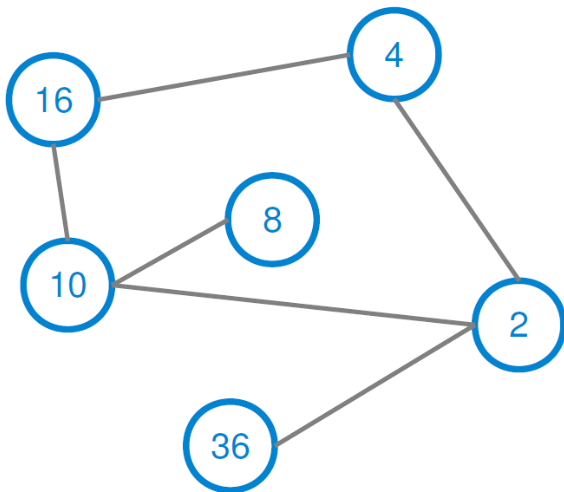
- Executed by all processes:

```
upon receive<q, pullResponse, Sq> do:
  Sp = update(Sp, Sq)
```

- `update` function:
  - avg: return `(Sp + Sq) / 2`
  - max: return `max(Sp, Sq)`

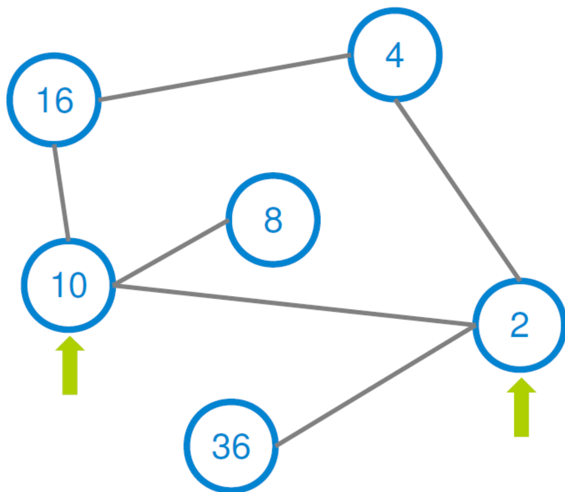
Aggregation Example (1/5)

- Taking the average of the numbers in the nodes.



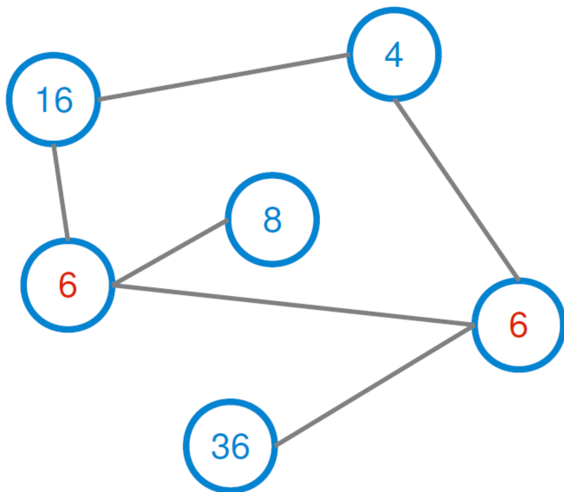
Aggregation Example (2/5)

- Taking the average of the numbers in the nodes.



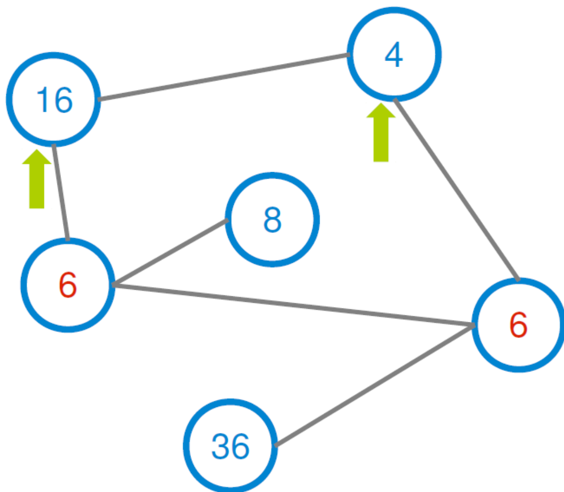
Aggregation Example (3/5)

- Taking the average of the numbers in the nodes.



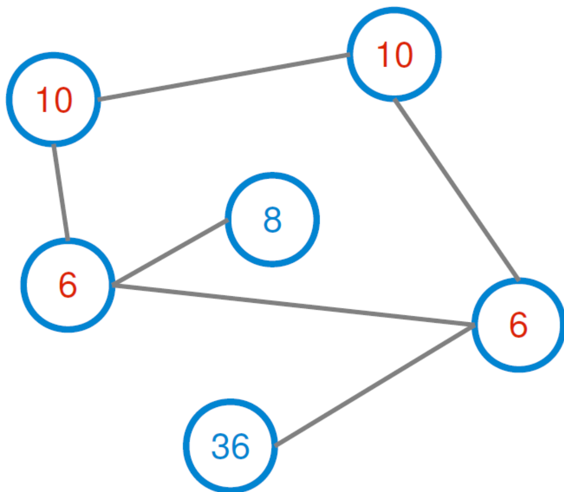
Aggregation Example (4/5)

- Taking the average of the numbers in the nodes.



Aggregation Example (5/5)

- Taking the average of the numbers in the nodes.



Network Size Estimation

- Any ideas?

Network Size Estimation

- Any ideas?
- All nodes set their states to 0.

Network Size Estimation

- Any ideas?
- All nodes set their states to 0.
- The initiator sets its state to 1 and starts gossiping for the average.

Network Size Estimation

- Any ideas?
- All nodes set their states to 0.
- The initiator sets its state to 1 and starts gossiping for the average.
- Eventually all nodes converge to the $avg = \frac{1}{N}$.

Peer Sampling Service

Epidemic Protocols

- In a epidemic (gossip) protocol, each node in the system periodically exchanges information with a subset of nodes.

Epidemic Protocols

- In a epidemic (gossip) protocol, each node in the system periodically exchanges information with a subset of nodes.

- The choice of this subset is crucial.

Epidemic Protocols

- In a epidemic (gossip) protocol, each node in the system periodically exchanges information with a subset of nodes.

- The choice of this subset is crucial.

- Ideally, the nodes should be selected following a uniform random sample of all nodes currently in the system.

Achieving a Uniform Random Sample

- Each node may be assumed to know every other node in the system.

Achieving a Uniform Random Sample

- Each node may be assumed to know every other node in the system.
- Providing each node with a complete membership table is unrealistic in a large scale dynamic system.

Peer Sampling

- An alternative solution.

Peer Sampling

- An alternative solution.

- Every node maintains a relatively small local membership table that provides a partial view on the complete set of nodes.

Peer Sampling

- An alternative solution.
- Every node maintains a relatively small local membership table that provides a partial view on the complete set of nodes.
- Periodically refreshes the table using a gossiping procedure.

Peer Sampling Generic Framework (1/4)

- Executed by all processes:

```
repeat evert t time units:
  q = selectPeer()
  buf = ((myAddress, 0))
  view.permute()
  move oldest H items to the end of view
  buf.append(view.head(c/2-1))
  send <p, psRequest, buf> to q
```

Peer Sampling Generic Framework (2/4)

- Executed by all processes:

```
upon receive<p, psRequest, bufp> do:
  buf = ((myAddress, 0))
  view.permute()
  move oldest H items to the end of view
  buf.append(view.head(c/2-1))
  send <q, psResponse, buf> to p
  view.select(c, H, S, bufp)
  view.increaseAge()
```

Peer Sampling Generic Framework (3/4)

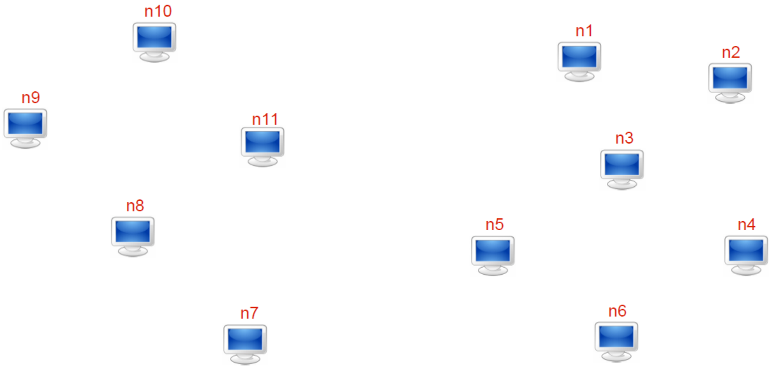
- Executed by all processes:

```
upon receive<q, psResponse, bufq> do:
  view.select(c, H, S, bufq)
  view.increaseAge()
```

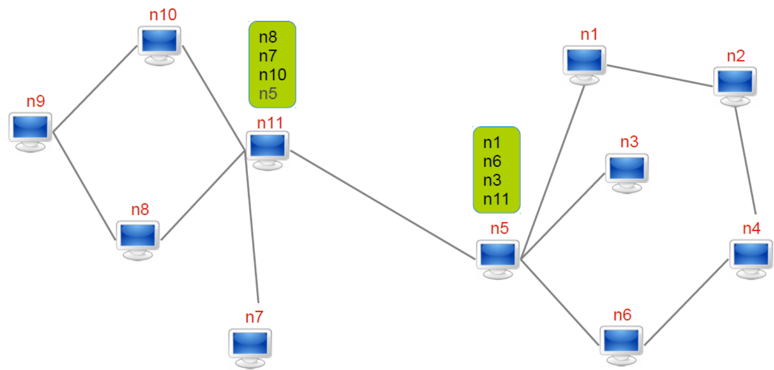
Peer Sampling Generic Framework (4/4)

```
method view.select(c, H, S, bufp)
  view.append(bufp)
  view.removeDuplicates()
  view.removeOldItems(min(H, view.size-c))
  view.removeHead(min(S, view.size-c))
  view.removeAtRandom(view.size-c)
```

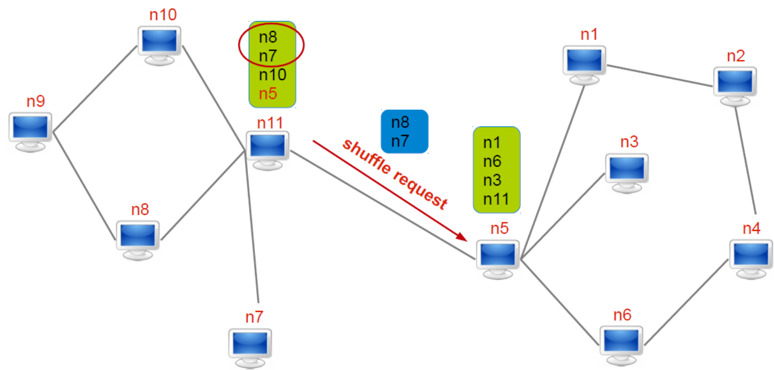
Gossip-based Peer Sampling (1/7)



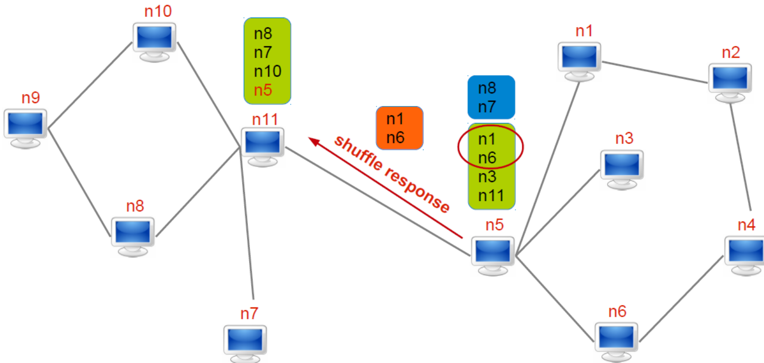
Gossip-based Peer Sampling (2/7)



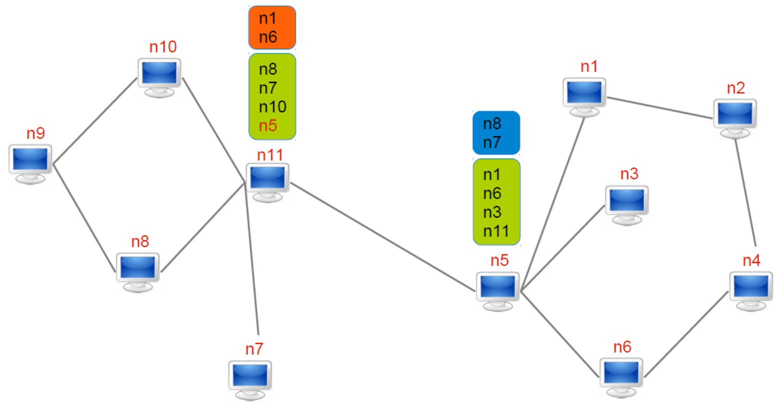
Gossip-based Peer Sampling (3/7)



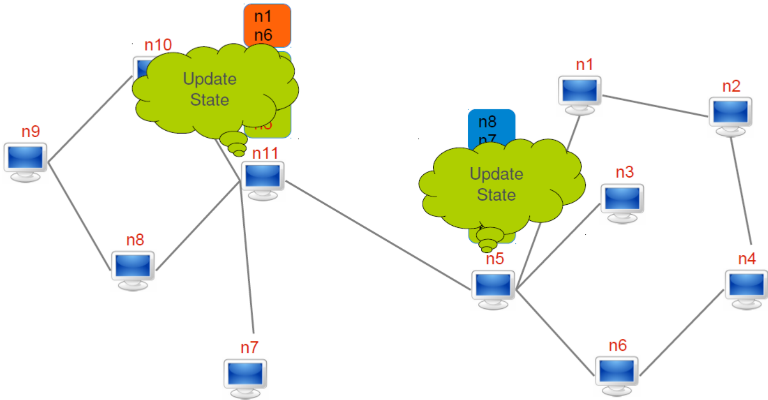
Gossip-based Peer Sampling (4/7)



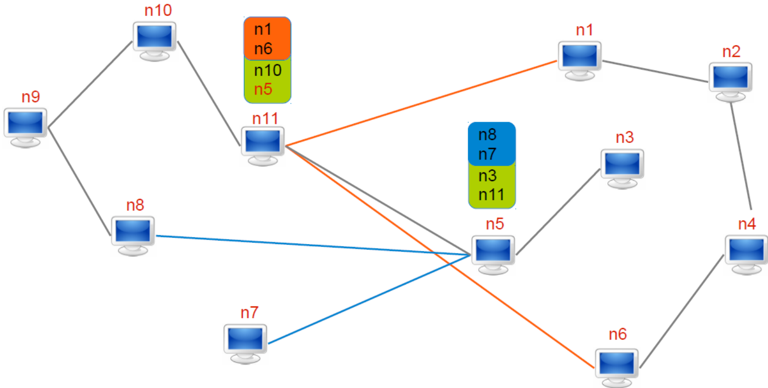
Gossip-based Peer Sampling (5/7)



Gossip-based Peer Sampling (6/7)

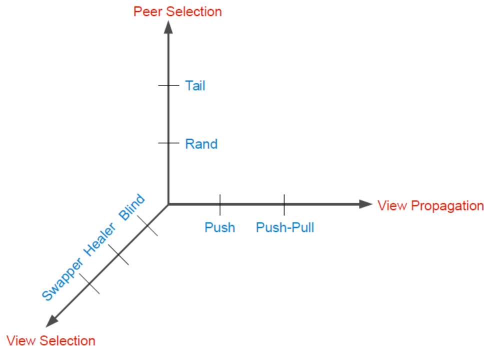


Gossip-based Peer Sampling (7/7)



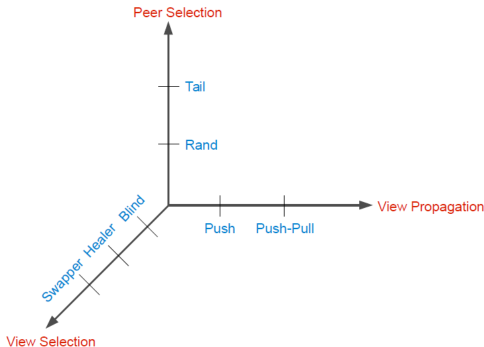
Peer Sampling Design Space

- Peer Selection
  - Rand: uniform random
  - Tail: highest age
- View Propagation
  - Push
  - Push-Pull
- View Selection
  - Blind: $H = 0, S = 0$
  - Healer: $H = c/2$
  - Swapper: $H = 0, S = c/2$

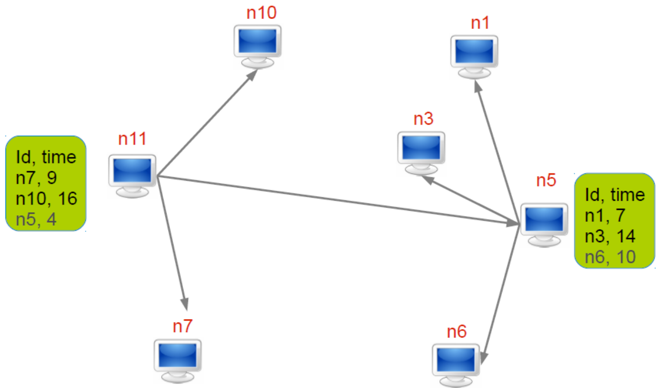


Cyclon as a Peer Sampling Service

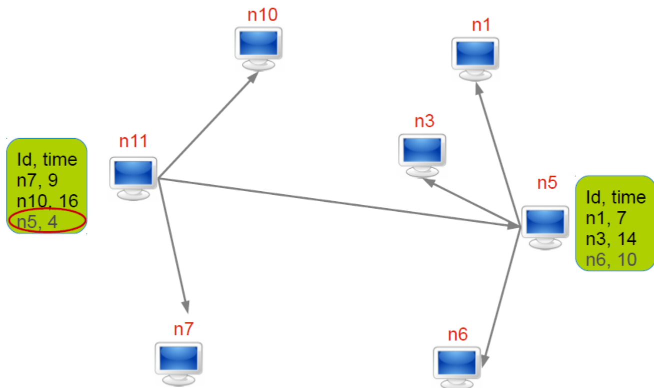
- Peer Selection
  - Rand: uniform random
  - Tail: highest age
- View Propagation
  - Push
  - Push-Pull
- View Selection
  - Blind: $H = 0, S = 0$
  - Healer: $H = c/2$
  - Swapper: $H = 0, S = c/2$



Cyclon (1/5)

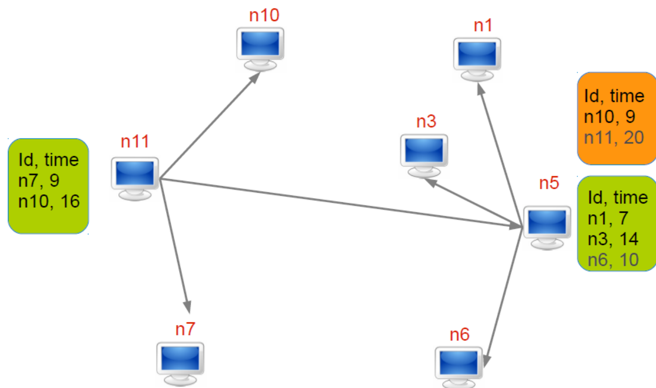


Cyclon (2/5)



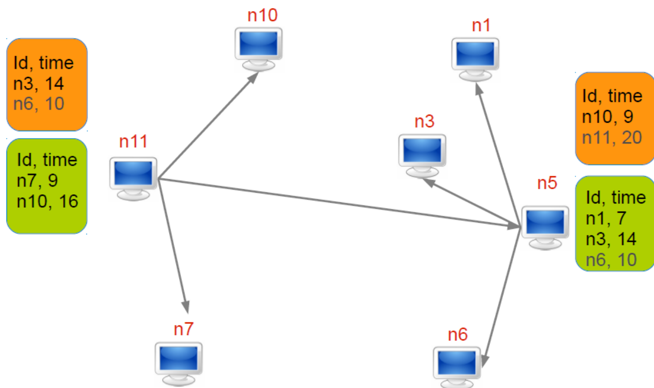
- Pick the oldest node from my view and remove it from the view (tail)

Cyclon (3/5)



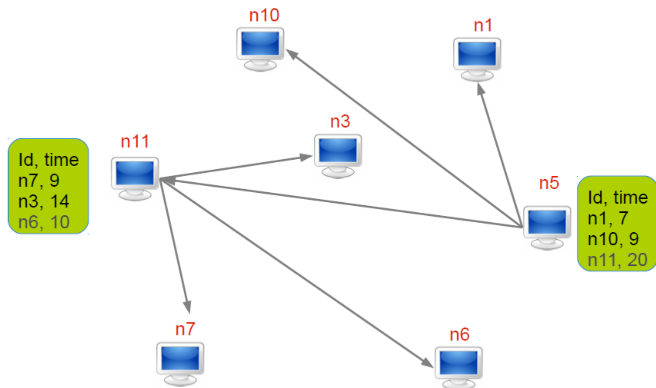
- Exchange some of the nodes in neighbours (push-pull)

Cyclon (4/5)



- Exchange some of the nodes in neighbours (push-pull)

Cyclon (5/5)



- Update the views (swapper)

Topology Management

T-Man

- T-man is a protocol to construct and maintain any topology with the help of a ranking function.

T-Man

- T-man is a protocol to construct and maintain any topology with the help of a ranking function.
- The ranking function orders any set of nodes according to their desirability to be neighbors of a given node.

T-Man Generic Framework (1/3)

- Executed by all processes.

```
repeat evert t time units:
  q = selectPeer()
  myDescriptor = (myAddress, myProfile)
  buf = merge(view, myDescriptor)
  buf = merge(buf, view.rnd)
  send <p, tmanRequest, buf> to q
```

T-Man Generic Framework (1/3)

- Executed by all processes.

```
repeat evert t time units:
  q = selectPeer()
  myDescriptor = (myAddress, myProfile)
  buf = merge(view, myDescriptor)
  buf = merge(buf, view.rnd)
  send <p, tmanRequest, buf> to q
```

- `selectPeer`
  - Sort all nodes in the view based on ranking.
  - Pick randomly one node from the first half.

T-Man Generic Framework (1/3)

- Executed by all processes.

```
repeat evert t time units:
  q = selectPeer()
  myDescriptor = (myAddress, myProfile)
  buf = merge(view, myDescriptor)
  buf = merge(buf, view.rnd)
  send <p, tmanRequest, buf> to q
```

- `selectPeer`
  - Sort all nodes in the view based on ranking.
  - Pick randomly one node from the first half.
- `view.rnd`
  - Provides a random sample of the nodes from the entire network, e.g., using cyclon.

T-Man Generic Framework (2/3)

- Executed by all processes.

```
upon receive<p, psRequest, bufp> do:
  myDescriptor = (myAddress, myProfile)
  buf = merge(view, myDescriptor)
  buf = merge(buf, rnd.view)
  send <q, tmanResponse, buf> to p
  buf = merge(bufp, view)
  view = selectView(buf)
```

T-Man Generic Framework (2/3)

- Executed by all processes.

```
upon receive<p, psRequest, bufp> do:
  myDescriptor = (myAddress, myProfile)
  buf = merge(view, myDescriptor)
  buf = merge(buf, rnd.view)
  send <q, tmanResponse, buf> to p
  buf = merge(bufp, view)
  view = selectView(buf)
```

- `selectView`
  - Sort all nodes in buffer (about double size of the view).
  - Pick out c highest ranked nodes.

T-Man Generic Framework (3/3)

- Executed by all processes.

```
upon receive<q, tmanResponse, bufq> do:
  buf = merge(bufq, view)
  view = selectView(buf)
```

T-Man Generic Framework (3/3)

- Executed by all processes.

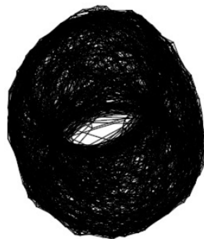
```
upon receive<q, tmanResponse, bufq> do:
  buf = merge(bufq, view)
  view = selectView(buf)
```

- `selectView`
  - Sort all nodes in buffer (about double size of the view).
  - Pick out c highest ranked nodes.

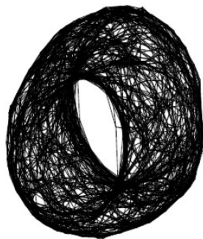
Ranking Function

- Sample ranking functions:
  - Line: $d(a, b) = |a - b|$
  - Ring: $d(a, b) = min(N - |a - b|, |a - b|)$

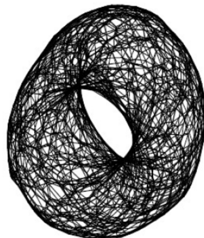
Illustration of T-Man



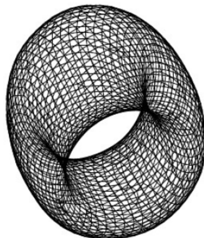
after 3 cycles



after 5 cycles



after 8 cycles



after 15 cycles

Summary

Summary

- Epidemic/Gossip algorithms: anti-entropy and rumor mongering
- Aggregation
- Peer sampling service: cyclon
  - Peer selection
  - View propagation
  - View selection
- Topology management: T-man

References:

- M. Jelasity, and A. Montresor, Epidemic-style proactive aggregation in large overlay networks, ICDCS, 2004.
- M. Jelasity, and O. Babaoglu, T-Man: Gossip-based overlay topology management, Engineering Self-Organising Systems, 2006.
- M. Jelasity et al., Gossip-based peer sampling, TOCS, 2007.
- S. Voulgaris, D. Gavidia, and M. Van Steen, Cyclon: Inexpensive membership management for unstructured P2P overlays, Journal of Network and Systems Management, 2005.
- A. Demers et al., Epidemic algorithms for replicated database maintenance, PODC, 1987.

Questions?

Acknowledgements

Some slides were derived from Alberto Montresor (Trento University).