A gossip dissemination protocol for a P2P Publish/Subscribe system

Master Thesis Defense

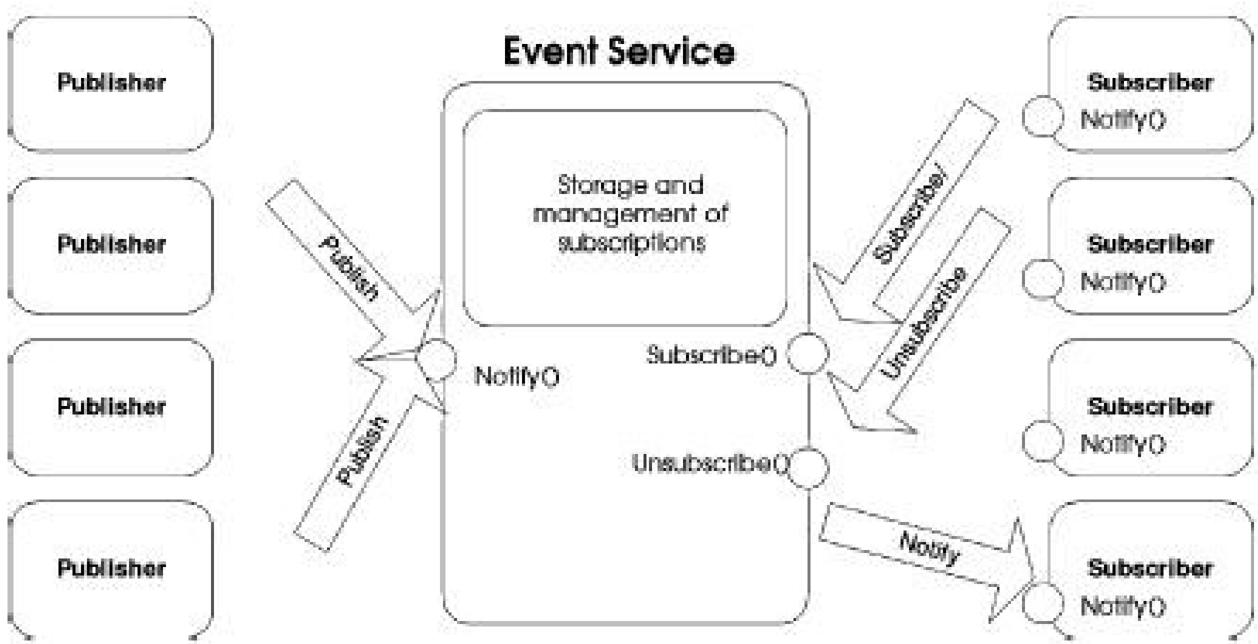
Panagiotis Hasapis (chasapis@aueb.gr)
Advisor: George Xylomenos (xgeorge@aueb.gr)

Main Problem

- The Internet as we know has changed
- No more clients and servers
  - Today we have *producers* and *consumers*
  - Some people are both
  - Social relationships are formed between them
- Besides the relationships, nowadays:
  - We care about categories, not data
  - We care about categories, not their producers
  - Social graphs shape and help knowledge sharing
- No more 1-1, we now have many-many
- Publish/Subscribe architecture fits

A Publish/Subscribe issue

- Many-many can be described as Publish/Subscribe
- Publish/Subscribe model fits better here than client/server



Types of Publish/Subscribe

- *Topic-based*
  - *Each element characterized by title*
- Content-based
  - Each element several fields
  - Query language to traverse content
- Type-based
  - Like topics with the use of ontologies

Related Work

- Pub/Sub P2P
  - "Publish/Subscribe in a mobile environment", Hector Garcia-Molina et al, Wireless Networks, Springer, 2004
- Gossip and semantics
  - "Epidemic-style management of semantic overlays for content based searching", Spyros Voulgaris et al, Computer Science Lectures, Springer, 2005
- Tribler
  - "TRIBLER: A social-based peer-to-peer system", Pouwelse et all, Concurrency and Computation, CiteSeer, 2008

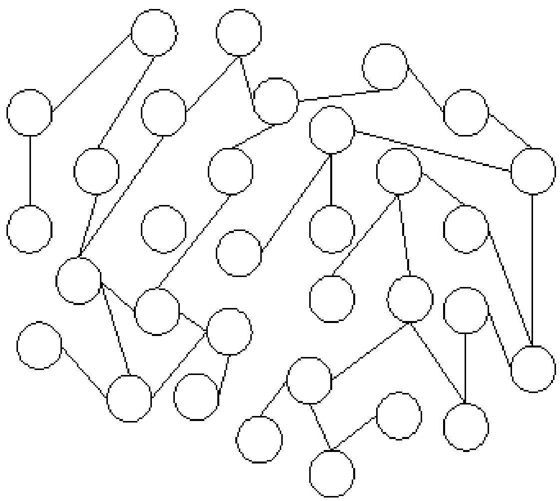
Problems

- Research in Pub/Sub P2P lacks in realism:
  - Connections between producers/consumers are random graphs
  - Uniform distribution function in object demand
    - Assumes everyone wants everything with same probability
- No scenarios in which we can use Pub/Sub

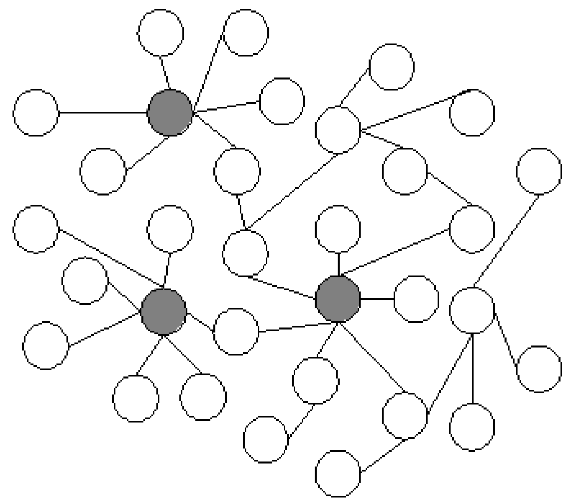
Social Networks

- How can we describe a social graph?
- It is a graph which is:
  - connected
  - undirected
  - not a tree
  - some nodes (30%) with many friends, rest (70%) with few

Social Networks



(a) Random network



(b) Scale-free network

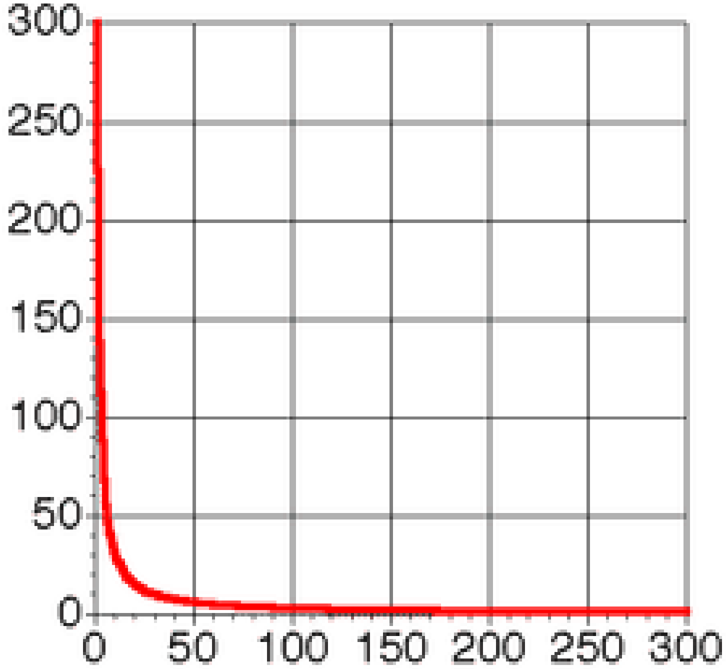
Uniform probability function

Power law distribution: $P(k) = k^{-\gamma}$, $2<\gamma<3$

Pros of social networks

- Hubs serve the whole network
- Many nodes with few connections
- Small probability for network to be disconnected
  - Few hubs -> smaller probability
- Even so, easier to recreate a scalable network

Demand distribution

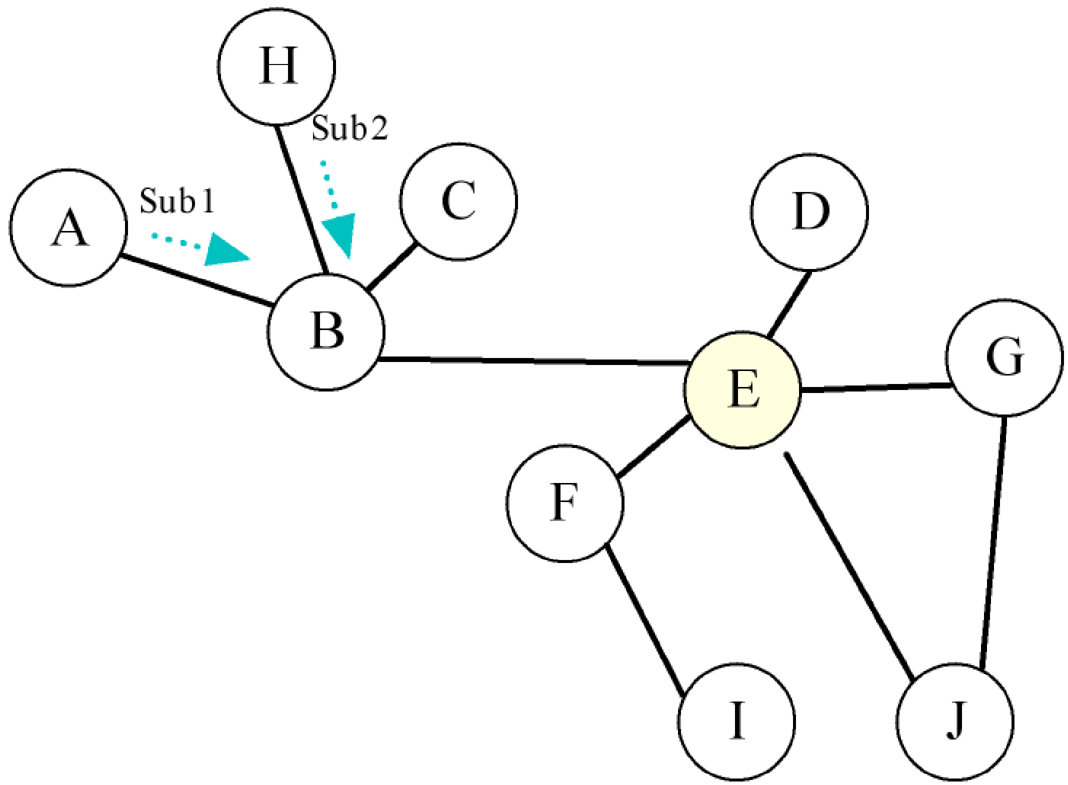


- All things equally demanded?
- Of course not !!
- Zipf distribution is needed
- Both for subscribing distribution and publishing distribution
- Power law $P_i = 1/i^a$, a ~ 1

Our idea

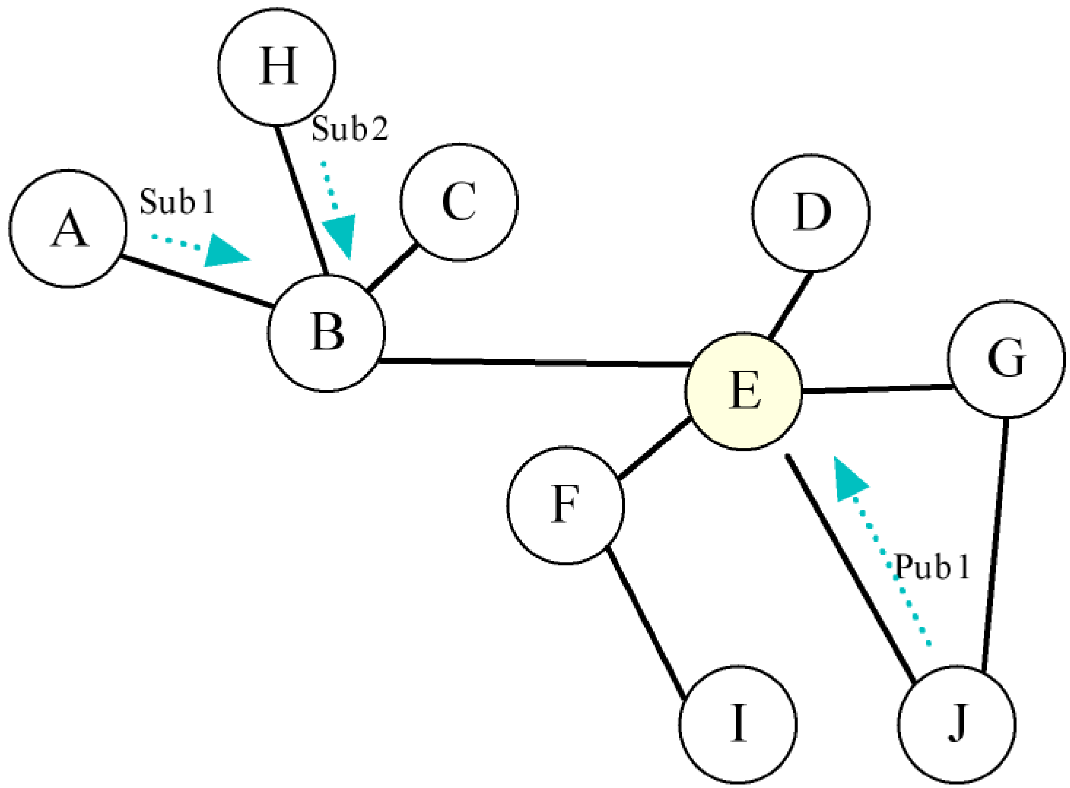
- A social overlay IP network to be used
- Realistic demand distribution
- Topic-based Publish/Subscribe
- Users = peers in unstructured p2p system
- Peers will exchange advertisements asynch
- Gossiping will disseminate advertisements
  - Both publishing and subscribing
  - No data transaction

Our idea



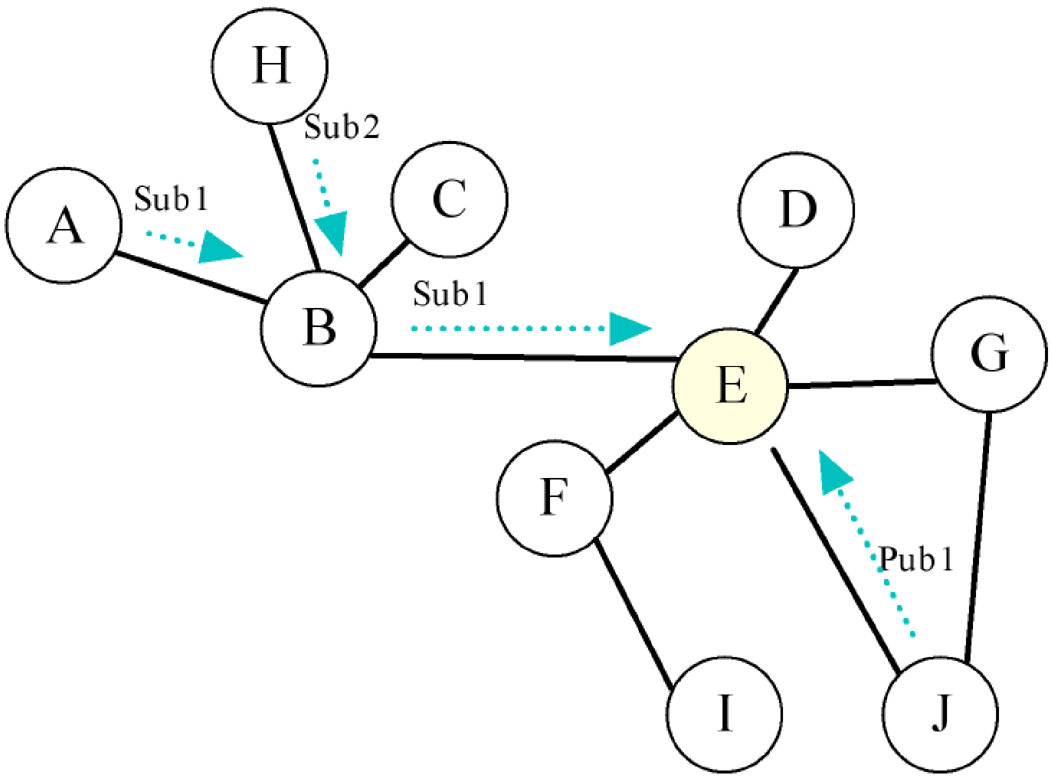
Intermediate peers will match pubs and subs of two unreachable producer and consumer

Our idea



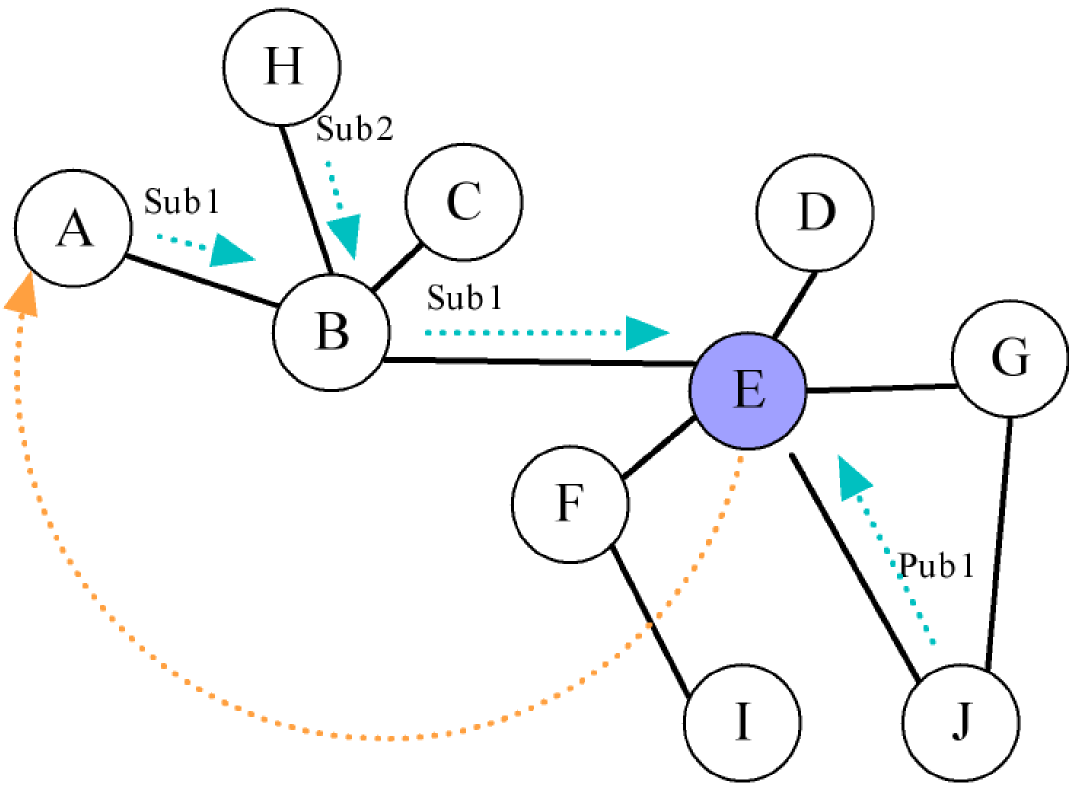
Intermediate peers will match pubs and subs of two unreachable producer and consumer

Our idea



Intermediate peers will match pubs and subs of two unreachable producer and consumer

Our idea



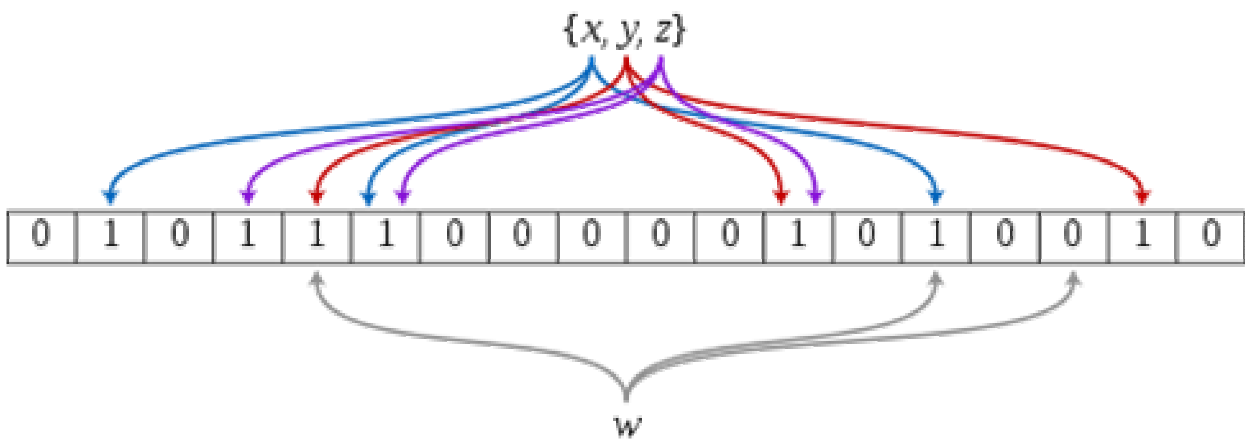
Intermediate peers will match pubs and subs of two unreachable producer and consumer

What else we need

- We consider objects to be pre-classified
  - Each object with a topic (R_{id}) or taxonomy
- Local load balancing
  - Unmatched advertisements will fade out
  - Its own will not be released
- Minimize traffic
  - UDP network programming
- Scalability
  - Use of Bloom filters
  - Add random peers in each node (as in Tribler)
- Minimize duplicate messages
  - Use of Bloom filters

Minimize duplicate messages

- k hash functions for bloom filter
- Answers only whether an element is in filter or not
- Will store people that already seen this particular advertisement



Advertisement Message

Origin Address	Originating peer's address
Message Type	Publication or Subscription
Bloom bit array	The character array of the filter
Bit array size	The size of the array (in bytes)
Resource ID	The topic of the message
Hops	Hop Count
Object's name	The advertised objects name

At peer initialization, its cache will store only its own sub messages (and pub, if exist)

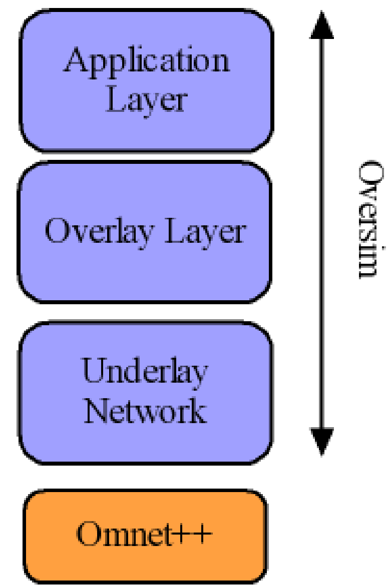
The algorithm

```
GossipTell()
{
  while(true)
  {
    sleep(gossip interval)
    delete old unmatched Rid's

    msg :=selectRandom() //either pub or sub

   select up to maxUsers that you have not
          send it previously

    add selected nodes to bloom in msg
    udp_send msg to each selected node
  }
}
```

```
//event triggered method, as soon as a
//gossip message received
GossipHear()
{
  get msg from lower layer

  if (already seen any Name in msg)
    discard message and exit function

  for each interested party
      inform party and keep statistics

  store msg – update datastructs accordingly
}
```

Simulation Platform

- Use of Oversim on top of Omnet++
- Omnet++ is the main simulator
- Oversim provides framework
  - Especially good for P2P apps
  - Tier architecture
  - Scripts for results processing

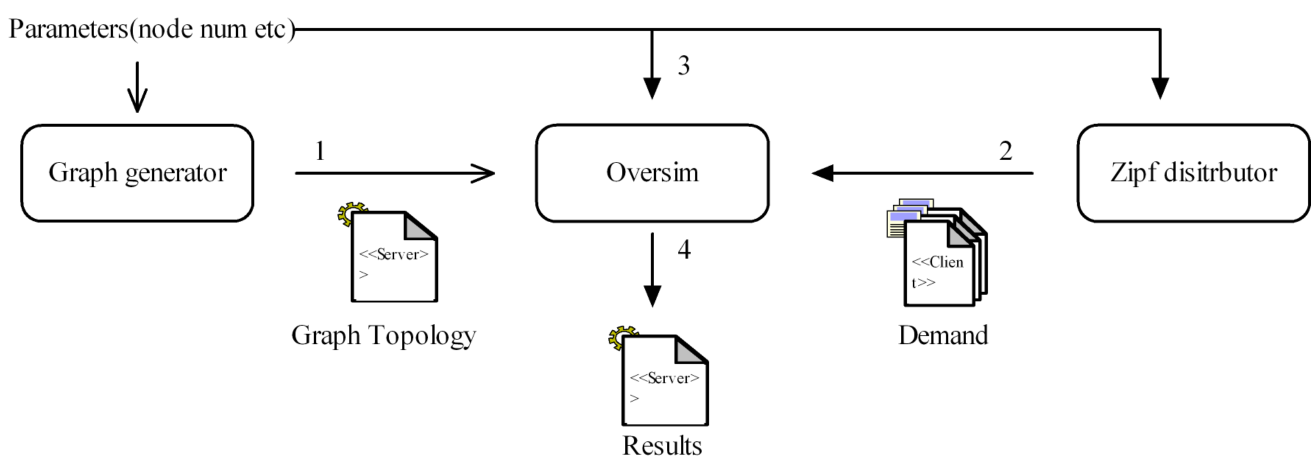


Implementation

- Needed a dummy overlay tier
- Substituted their overlay with ours
- Implemented over UDP
- Algorithm implemented on higher tier
  - Much faster this way
  - Simpler to implement

Experiments methodology

- Assumed constant uniform packet loss = 0.05 %
- Made tests for 200, 400, 600 peers
- Assumed a Zipf distribution of Pub/Sub topics
- Graph generator is NGCE 2.0
- All users are subscribers, only 30% publishers

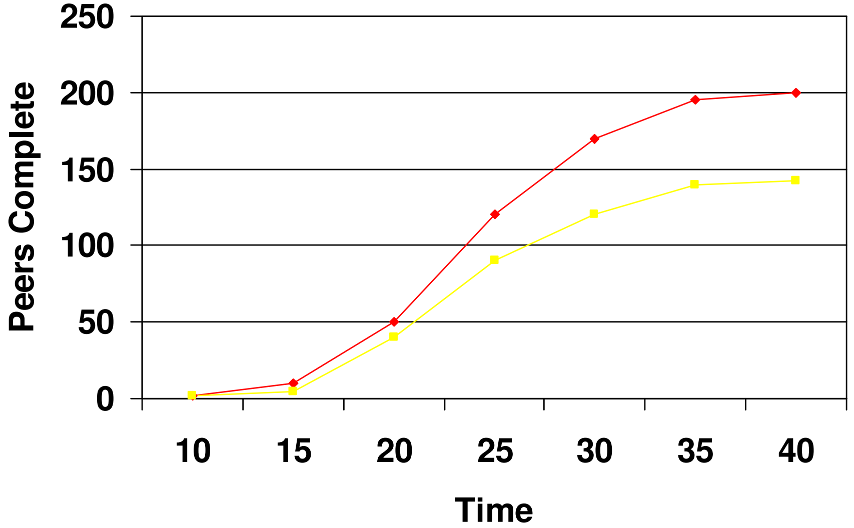


What to measure

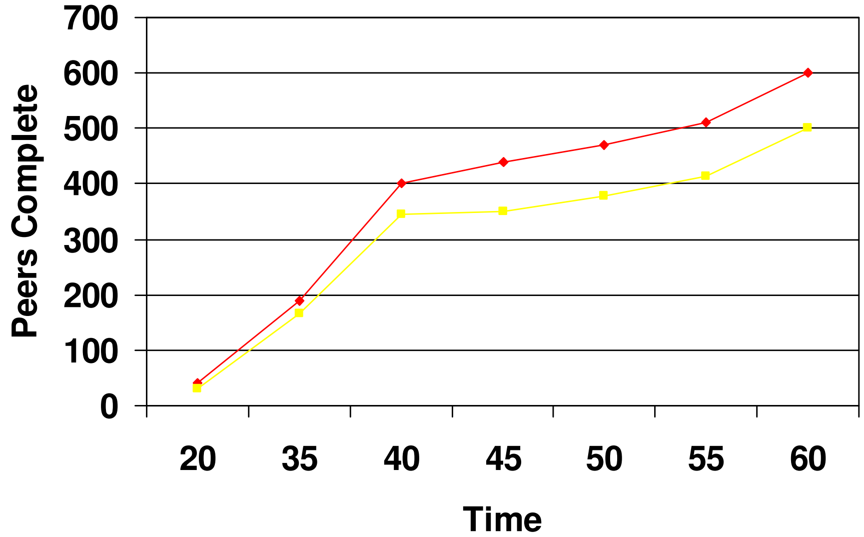
- Two basic measurements
- 1-Peers complete
  - When he has received a match for all his subscriptions
- 2-Who are the ones that make matches
  - Which type of user makes the most matches
- Will compare results with the basic gossip protocol
  - Protocol in which only pub messages are gossiped
  - Until subscribers are reached

Results (Peers complete)

For 200 peers

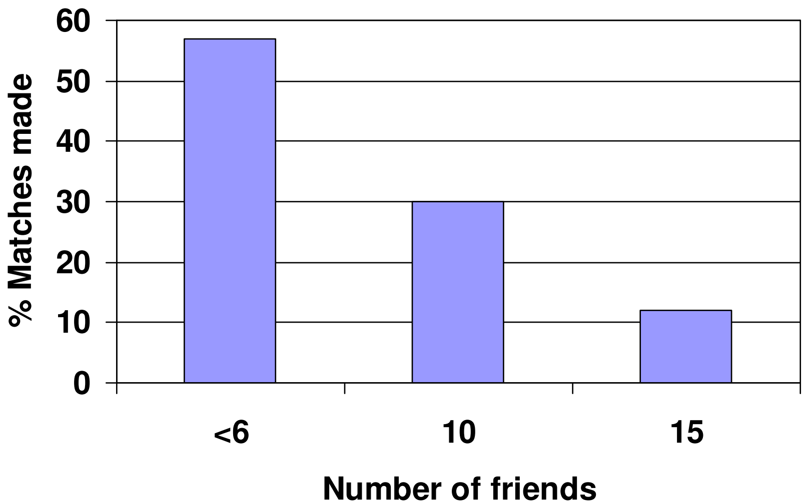


For 600 peers

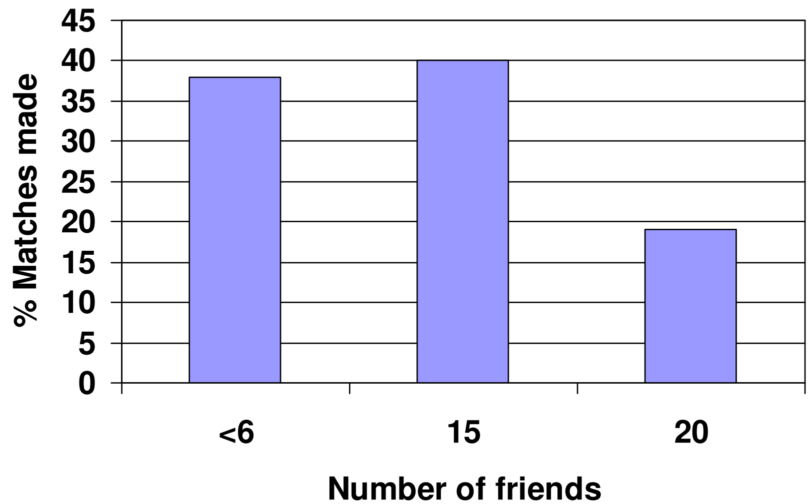


Results (Who made most)

For 200 peers



For 600 peers

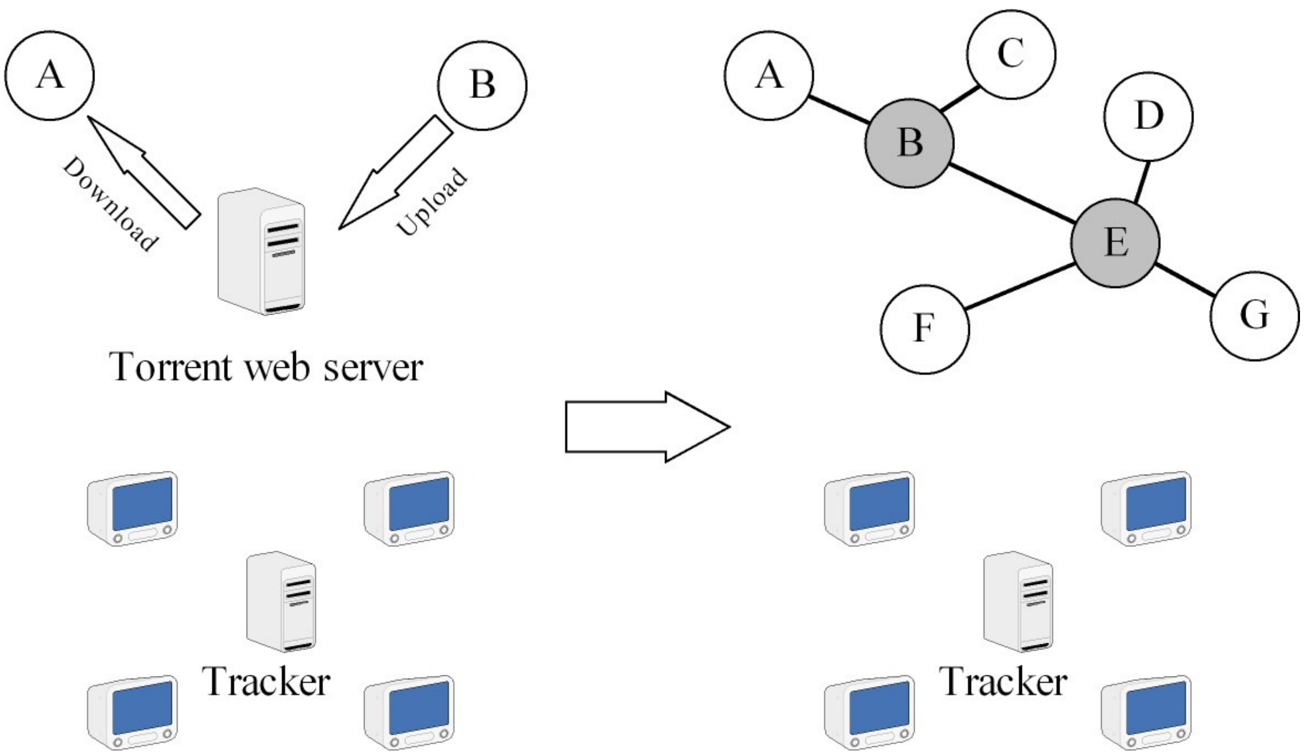


Practical uses

- Not many P2P Publish/Subscribe scenarios
  - Researchers neglected to propose
- Believe in the power of networks
- We propose a use in BitTorrent
  - To eliminate the need of torrent search engines

A BitTorrent Scenario

From this… to this



Future Work

- Several issues still to be addressed
- More uses to be proposed
- Recommendation mechanisms
- Trust: reputation mechanisms for users
    - With a social network, several can be done

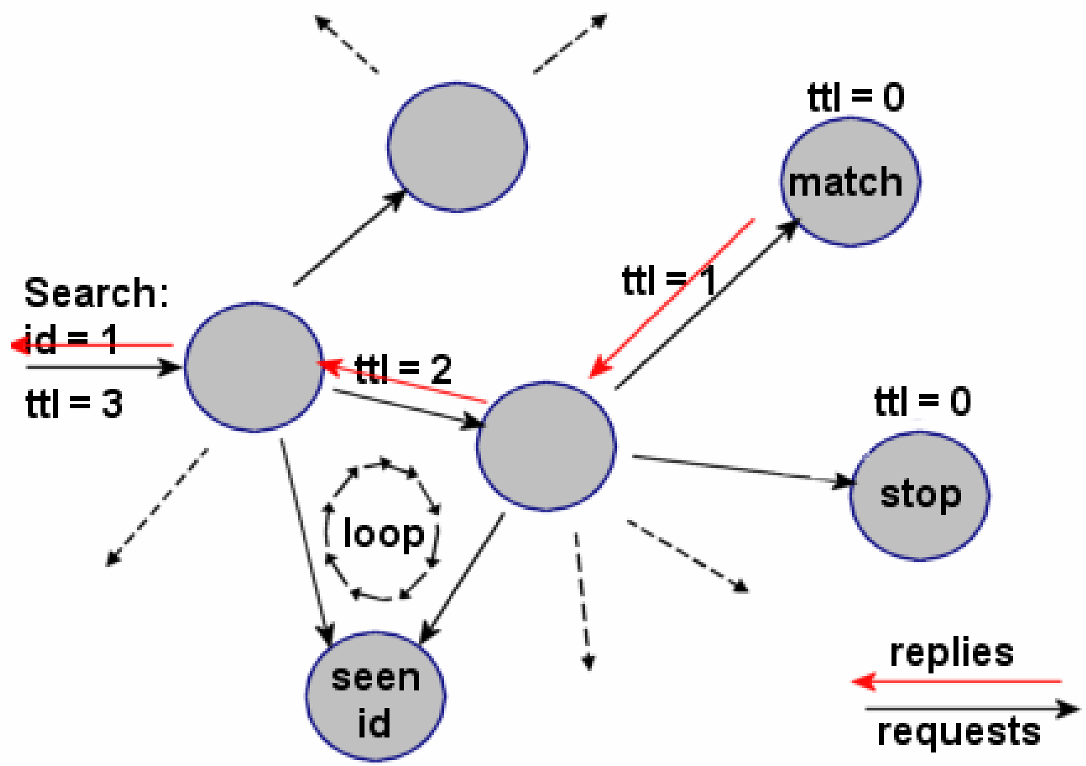
Thank you!

Thank you for your time!!!

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ

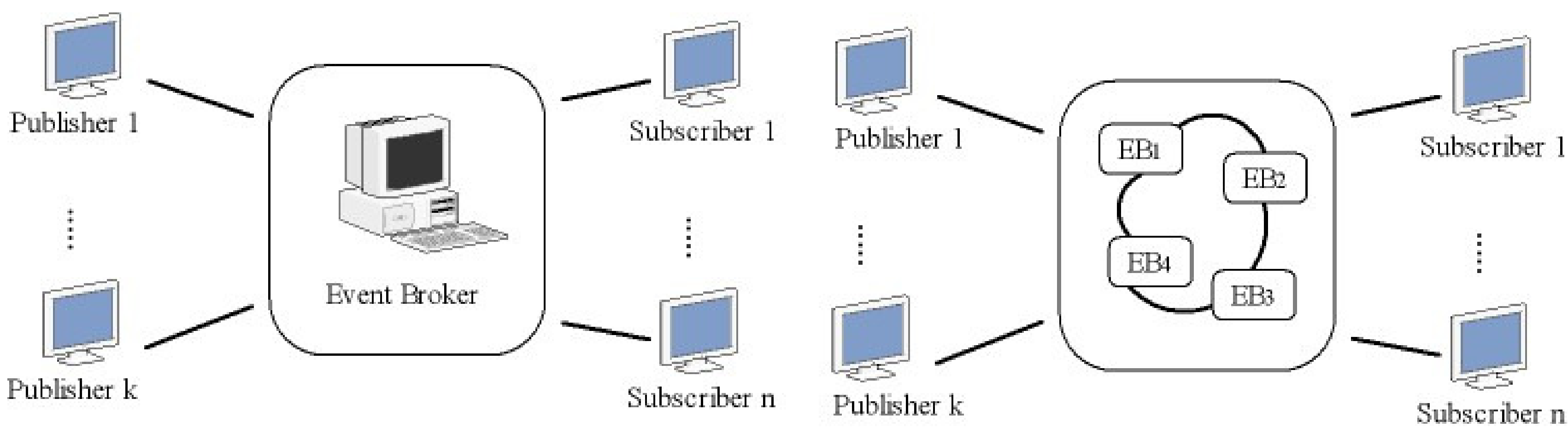
Backup Slides

Unstructured P2P system



Distributed Pub/Sub

Distributed solutions are always scalable



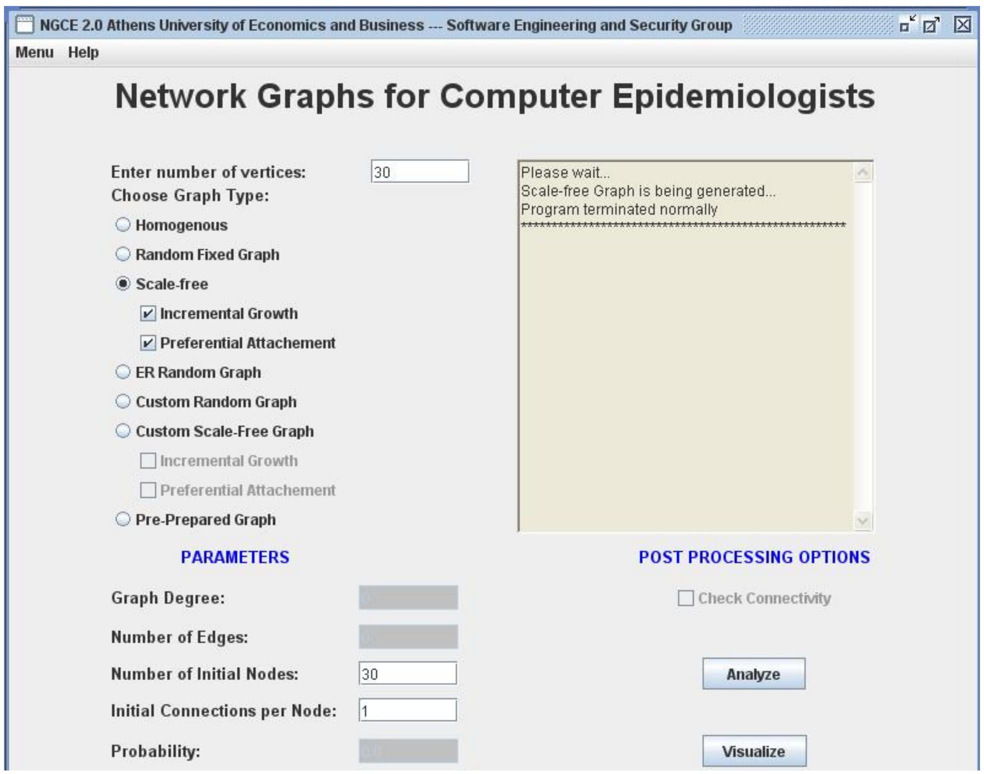
The algorithm

```
GossipTell()
{
 while(true)
 {
   sleep(gossip interval)
   ResourceAger.increaseAll()
   ResourceAger.discharOldEntries()

   msg := ResourceMemory.selectRandom()
   i = 0
   while(i<maxNodesToSent)
  {
   if(msg.BloomFilter.overloaded()==true)
     msg.BloomFilter.clear();

   BF := new BloomFilter(msg.peopleAlreadySent + msg.Rid +
             msg.origin-peer-name + msg.Name)

   selected_node := selectUserRandomly(friends-list ∪ random-nodes)
   if(BF.check() ==true)
     selectUserRandomly(friends-list ∪ random-nodes) repeat while
  else
  {
   i = i + 1
   // The gossip message is created here
   make package := msg.Rid | users-name | msg.origin-peer-name |
                  msg.Name | BF
   udp_send(selected_node, package)
  }
 }
}
```

```
//event triggered method, as soon as a gossip message received
GossipHear()
{
 ResourceMemory.has(msg.Rid , !msg.type))
 unpack package := msg.Rid | users-name | msg.origin-peer-name | msg.Name |
BF

 if user-name is blacklisted OR already seen any Name in msg
   discard message and exit function

 if(ResourceMemory.has(msg.Rid , !msg.type))
 {
   intParties := ResourceMemory.getInterestedParties(msg.Rid , !msg.type))

   for each party in intParties
    inform each party and keep statistics
 }
 else
   ResourceAger.insert(msg.Rid , 0)

 // if already inside ResourceMemory, will not be re-inserted
 // but Bloom Filters might changed accordingly
 ResourceMemory.insert(msg.Rid , msg.origin-peer-name, BF)
}
```

Graph generation

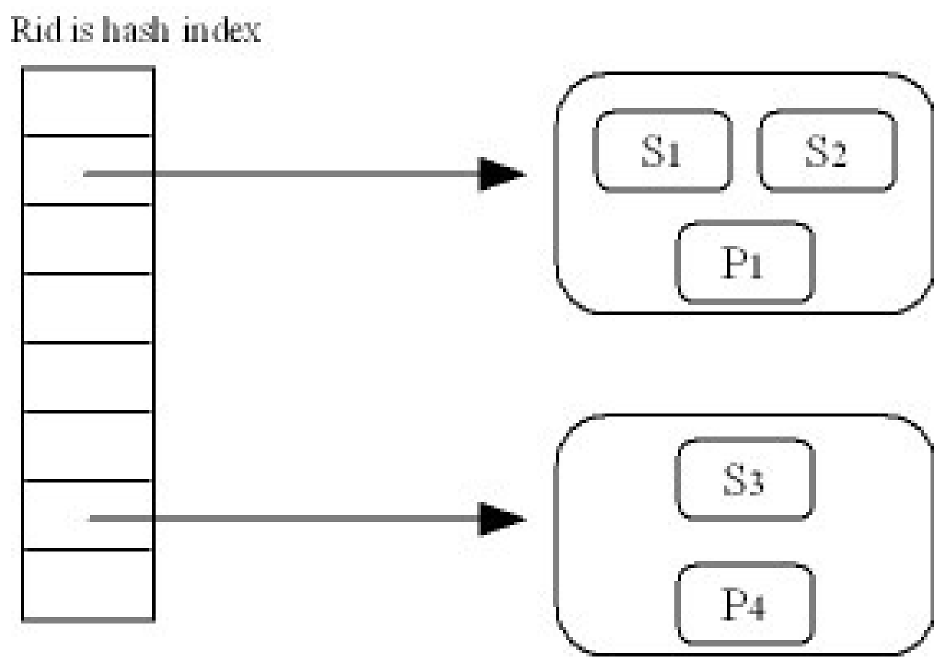
Using NGCE 2.0 by ISTLab




```
Node 0 has 1 Edges *
Node 1 has 1 Edges *
Node 2 has 3 Edges ***
Node 3 has 1 Edges *
Node 4 has 1 Edges *
Node 5 has 4 Edges ****
Node 6 has 3 Edges ***
Node 7 has 1 Edges *
Node 8 has 1 Edges *
Node 9 has 1 Edges *
Node 10 has 2 Edges **
Node 11 has 1 Edges *
Node 12 has 2 Edges **
Node 13 has 5 Edges *****
Node 14 has 1 Edges *
Node 15 has 3 Edges ***
....
```

```
#Nodes:= 30
#Initial:= 30
#m:= 1
#Class:= FullScaleFreeGraph
#Seed:= 2
#Version:= 2.0
0        24
1        6
2        14
2        22
2        26
3        13
4        26
5        11
5        21
5        28
....
```

Data structures



With each advertisement their paths in bloom filters are stored

Bloom Filter formula

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

n : number of elements to insert

m : number of bits

p : desired false probability

*Example -ln(0.06)*200/ln(2)^2 ~ 1000 bits ~ 128 bytes*