Lesson 3

P2P Hybrid Systems

Case Studies: Gnutella 0.6, Kazaa, Skype

Laura Ricci

27/02/2017
LESSON OUTLINE

• Hybrid Unstructured P2P networks

• Super Peer Election: Self Organization

• Resource indexing

• Case studies
  • Gnutella 0.6
  • Kazaa
  • Skype
HYBRID UNSTRUCTURED P2P

- Hybrid of centralized unstructured and pure unstructured overlays
- Two Roles
  - **Superpeers** act as local search hubs
    - similar to a Napster server for a small portion of the network
    - automatically chosen by the system
      - based on their capacities (storage, bandwidth, etc.) and availability (connection time)
      - dynamic definition of a hierarchical level in the network
    - periodically exchange information on peers’ resources
    - take charge of much of the load on slower nodes
  - **Peers**
    - upload their resource description to a Superpeer
    - query the Superpeers
    - involved in resource transfer
- Famous P2P-hybrid overlays
  - Gnutella (starting from v.0.6)
  - Kazaa & Morpheus (proprietary systems)
GNUTELLA 0.6: GOALS

- Gnutella 0.6: an improvement with respect to Gnutella 0.4
  - higher signalling efficiency than pure P2P
  - similar reliability (no single point of failure)
  - a good compromise

- the hosts with a reduced computational power/bandwidth do not belong to the Gnutella overlay, even if they can access the overlay through other nodes
  - “slow hosts” does not slow down the search of peers/resources, but can access the Gnutella services

- basic idea exploited also for other P2P systems (Kazaa, JXTA,..)

- a brief history: Gnutella 0.6 developed starting from Gnutella 0.4 in 2001.
  - different available clients (Limewire, Bearshare,..)
  - several protocol optimizations (privacy, scalability, performance,..)
HYBRID UNSTRUCTURED P2P: GNUTELLA 0.6 CASE STUDY

- Gnutella 0.6: improvements with respect to Gnutella 0.4
  - **UltraPeers**: execute all the functions of the Gnutella protocol
    - PING, PONG, QUERY, QUERYHIT, PUSH: shield the leaf nodes from the query routing
    - define a Ultrapeer overlay
    - are proxy for the leaf nodes they manage
    - must have suitable characteristics (example: high speed connections and connections not shielded by a NAT or a firewall)
    - 10-100 connections with the leave nodes, < 10 connections with other UltraPeers
  - **LeafNodes**:
    - connect to the UltraPeer to access the Gnutella network.
    - do not participate to the PING/PONG QUERY/QUERYHIT protocols and should not accept GNUTELLA CONNECT request
    - open only a few connections towards UltraPeer
      - often 1 connection
    - involved in file transfer
    - may dynamically ask for becoming a UltraPeer
HYBRID UNSTRUCTURED P2P: GNUTELLA 0.6 CASE STUDY

- Network Structure: hub-based network

- No central authority compares the computational powers of the different hosts
  - definition of a distributed algorithm to dynamically elect the UltraPeers
  - an example of self-organizing network
- The Gnutella 0.6 overlay characteristics are similar to those of Internet where low bandwidth nodes are connected to the routers that send data on large bandwidth connections (backbones)
ULTRAPEER CAPABILITY

Criteria for being UltraPeer Capable:

- **firewall/NAT absence**
- **Bandwidth:** approximated by computing download/upload throughput
  - e.g. Limewire, a famous client implementation, required 10kB/s Upload, 20kB/s download to be an UltraPeer
- **RAM** available to store the routing tables
- **operating system.**
  - Some operating system can manage a higher number of sockets with respect to others.
  - Limewire forbids a node from becoming a UltraPeer if it runs an old version of the operating system
- **computational power:** to manage the incoming queries
- **future uptime**
  - an heuristic: the future uptime is proportional to the past one
ELECTING ULTRAPEERS

• an UltraPeer capable node may become UltraPeer if more UltraPeer are needed in the overlay
  • the need of further UltraPeer may be estimated by considering the total amount of UltraPeer currently available in the overlay

• how can we perform this estimation?
  • gossip aggregation algorithms (you will develop them in the laboratory lessons)
  • simplified solution for Gnutella 0.6
    • local decision of each UltraPeer
    • communicate information at handshake time

• We show in the next slides a concrete example of self organizing algorithm implemented in Gnutella 0.6
  • new message headers are required
ELECTING ULTRAPEERS: GNUTELLA 0.6

Simplified solution for Gnutella 0.6 based on handshake messages

- exploit new headers introduced by the Gnutella 0.6 protocol
- recall general structure of headers
  \[ X\text{-}\text{header}\text{-}\text{Name}: \text{headervalue}_1, \text{headervalue}_2 \]

- **X-UltraPeer**
  - tells whether a host plans on acting as a ultrapeer (if true) or a shielded node (if false)

- **X-UltraPeerNeeded**
  - used to balance the number of UltraPeers.

- **X-TryUltraPeer**
  - like X-try in Gnutella 0.4, but includes only di UltraPeers addresses.

- **X-Degree**
  - number of leaves managed by an UltraPeer

- **X-QueryRouting**
  - support QueryRouting Protocol (QRP) (see later)
A LEAF CONNECTS TO AN ULTRAPEER

Node A

GNUTELLA CONNECT/0.6
User-Agent: LimeWire 1.9
X-Ultradeer: False
X-Query-Routing: 0.1
X-My-Address: 10.254.0.16:6349

UltraPeer B

GNUTELLA/0.6 200 OK
User-Agent: LimeWire 1.9
X-Ultradeer: True
X-Ultradeer-Neeed: False
X-Try-Ultradeeers: 23.35.1.145:6346,
X-Query-Routing: 0.1
X-My-Address: 10.254.0.16:6346
GNUTELLA/0.6 200 OK

at the end of the handshake, the NODE A:
- is a shielded node of the UltraPeer
- should drop any incoming connection request
- sends a QRP routing table (assuming QRP is used).
• An UltraPeer $U$ may refuse a connection from a leaf node.

• In this case, $U$ gives, at handshaking time, the addresses of further UltraPeers, by information previously received through connection PONG.
  
  Example: $X$-try-SuperPeers: 68.37.233.44:9376, ......

• Some connection PONG are returned even if the connection is accepted. In this case the information received may be exploited to define connections with alternative UltraPeers.
Leaf A trying to connect to shield Leaf B

- Leaf A may try to connect to Leaf B
  - B could have been an UltraPeer before
  - now it’s a shielded leaf and current state is still not updated on the bootstrapping nodes
  - B has to refuse incoming connection

<table>
<thead>
<tr>
<th>Node Leaf A</th>
<th>Existing Leaf B</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNUTELLA CONNECT/0.6</td>
<td>GNUTELLA/0.6 503 I am a shielded leaf node</td>
</tr>
<tr>
<td>X-Ultrapeer: False</td>
<td>X-Ultrapeer: False</td>
</tr>
<tr>
<td></td>
<td>X-Try-Ultraceers: 18.2.3.14:6346, 18.1.17.2:6346</td>
</tr>
</tbody>
</table>

- Sometimes nodes are ultrapeer-incapable and unable to find a ultrapeer
- in this case, they behave exactly like old, unrouted Gnutella 0.4 connections (they connect to unshielded peers)
New connections emerge among ultrapeers

- PING/PONG mechanism is performed by ultrapeers

**Ultraceeper A**

<table>
<thead>
<tr>
<th>GNUTELLA CONNECT/0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultraceeper: True</td>
</tr>
</tbody>
</table>

**Ultraceeper B**

<table>
<thead>
<tr>
<th>GNUTELLA/0.6 200 OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Ultraceeper: True</td>
</tr>
</tbody>
</table>

A new connection is established between A and B
B already manages a large number of connections

B refuses A’s connection request
LEAF GUIDANCE

• Sometimes there can be too many UltraPeer-capable nodes on the network

• Leaf guidance mechanism
  • goal: to balance leaf nodes and UltraPeers
  • what: forces the choice of the role for a new peer connecting to the overlay
  • how: local mechanism, if executed by all the UltraPeers, the whole network tends to balance

• Implemented through the X-Ultrapeer-Needed Gnutella 0.6 header
  • X-Ultrapeer-Needed = True the role of UltraPeer is accepted
  • X-Ultrapeer-Needed = False the node has a low number of leaf nodes
    “do you accept to become my leaf node?”
LEAF GUIDANCE

• Each UltraPeer has
  • a set of $k$ slots available for connection with further UltraPeers
  • $n$ slots for connections with leaf nodes

• If a leaf node asks for a connection
  • this is accepted if there is room
  • otherwise it is refused

• If an UltraPeer $A$ asks for a connection to a UltraPeer $B$
  • if $B$ has just a few leaf nodes, leaf guidance is activated to accept $A$ as a shielded leaf, instead of leaving an additional UltraPeer in the overlay
  • otherwise, $A$ remains UltraPeer and a new connection towards it is accepted by $B$ only if there is room
LEAF GUIDANCE

**Node A**

GNUTELLA CONNECT/0.6  
X-Ultradeer: True

**Ultradeer B**

GNUTELLA/0.6 200 ok  
X-Ultradeer: True  
X-Ultradeer-Needed: False

- A, which is Ultradeer capable, connects to B which manages a few leaves
- B has few leaves and directs A to become a leaf node
  - If A has no leaf connections, it stops fetching new connections, drops any Gnutella 0.4 connections, and sends its QRP table to B
  - Then B will shield A from all traffic
  - If A has leaves, the leaf guidance is refused, A remains Ultradeer and establishes a new connection to B
ULTRAPEER PROTOCOL: A SUMMARY

• Message Types:
  • Content Request: QUERY/QUERYHIT
  • Keep Alive: PING/PONG

• The ping-pong messages are exchanged only among UltraPeers
  • UltraPeers never propagate ping messages to their leaves
  • UltraPeers “shield” leaves from keep-alive traffic

• An UltraPeer may receive a ping from its LeafNodes
  • it replies pong with the information received from other SuperPeers
  • A LeafNode may exploit this information when its UltraPeer disconnects from the overlay or in the case it wants to open connections with further UltraPeers
QUERY ROUTING PROTOCOL (QRP)

- QRP governs how the Ultrapeer filter queries by only forwarding them to the leaf nodes that most likely have a match.

- **Goal**: avoid forwarding a query that cannot match
  - require resource indexes
  - done by looking the query words through a big hash table, the **Query Routing Table (QRT)**
    - sent by each leaf node to its Ultrapeer
  - requires combined action of Leaf and Ultra Peer
QRP LEAF NODE ROLE

- When a new file is available on a leaf, break file name into individual words.
  - Word: consecutive sequence of letters and digits (ASCII characters only).
  - Words are separated by space in the query

- Hash each word with a well-known hash function and insert a "present" flag in the corresponding hash table slot.
  - a big boolean array more than a real hash table
  - only the fact that a key ended up filling some slot.

- Re-hash words by removing their trailing 1, 2, or 3 letters
  - store the re-hashed word only if is at least 3 letters after such trimming
  - simple attempt to remove plural from words.
  - optionally, chop off more letters from the end

- The "boolean vector" built is
  - optionally compressed
  - broken up in small messages
  - sent mixed with regular Gnutella traffic from a peer to its Ultrapeer.
QRP Leaf Node Role

- Indexing a new song on peer A: “Bohemian Rhapsody - Queen”
  - break it into 4 words “bohemian”, “rhapsody”, “-”, “queen”
  - discard “-”
  - apply the hash function \( h(k, m) \) to each word \( k \)
    - \( m = \) the QRT size
    - the hash function should hash keys of arbitrary length reasonably uniformly
    - should be easy to implement and efficient to compute on most platforms
  - \( h(’bohemian’, 8) = 7 \), \( h(’rhapsody’, 8) = 2 \), \( h(’queen’, 8) = 5 \)
  - Peer A’s routing table = bitmap 00100101
- Query Routing Table: a boolean vector \( R \), including information on the content shared by the leaves
  - size from 64 kilobit slots up to 2 megabit

Hash function example
  - integer value on 32 bits; \( k[i] = i\text{-th} \) byte from \( k \)
- \( h(n, m) = \lfloor m ((n A) \mod 1) \rfloor \) where \( A = (\sqrt{5}-1)/2 \approx 0.6180339887\ldots \)
QUERY RESOLUTION

- Store updated "boolean vector" as the Routing table for each leaf node
- Query resolution:
  - queries are broken into individual words, all accentuated letters are removed.
  - for each leaf node with a Query Routing table:
    - each word is then hashed and looked up in the Query Routing table.
    - using the same hashing function used by the leaf nodes
    - to declare a Query Hit
      - all the words have to be found in the Query Routing Table
      - only some of them
    - only those queries that were declared a hit at the previous stage will be forwarded to a given leaf node.
  - False positive are possible
  - NO False negative
    - if all the positions of the bitvector corresponding to the hash of the keys in Q are 0, the leaf does not possess the resource
  - a simplified version of Bloom Filters (which exploit probabilistic techniques (to be seen later)
QUERY ROUTING PROTOCOL: QUERY FORWARDING

- query message: content is required by one leaf/UltraPeer
  - leaf sends request to its known UltraPeer

- UltraPeer looks up in its routing tables whether content is offered by one of its leaf nodes.
  - if so, request is forwarded to this node

- additionally the UltraPeer increases the hop counter and forwards this request to the neighbours
  - if a UltraPeer receives such a request from another UltraPeer, this request is handled the same way, as if it would have received it from one of its leaf nodes

- after the hop counter of the request reaches the TTL-value it is not forwarded any further

- TTL-flooding limited to UltraPeers only
QUERY ROUTING PROTOCOL: QUERY RESPONSE

- When a leaf node receives a request
  - double-checks whether it shares the file
  - may have not the content, due the approximation of QRT
- if success
  - the leaf node sends a content reply (QUERYHIT) back to the requesting peer, by means of backward routing
    - by sending it back to that node (UltraPeer) it received the message from
    - hop by hop the message can thus be routed back to the requesting node
- Content exchange:
  - directly between the leaf nodes, via HTTP connections
GNUTELLA 0.6: FURTHER MESSAGES

• Content Request
  • QUERY (the same of Gnutella 0.4)
  • QUERY_HIT (the same of Gnutella 0.4)

• Keep alive:
  • PING (the same of Gnutella 0.4)
  • PONG (the same of Gnutella 0.4)
OPTIMIZATIONS

- Routing indexes
  - each UltraPeer sends its routing table to its **neighbours**
    - by exploiting regular query forwarding traffic
    - in some Gnutella extension
      - propagate the routing tables to all the hosts within a given radius, defined by the TTL
      - modify the content of the QRT such that every position points out the distance from the host owning the content
  - A routing index for each connection

- Last Hop Saving:
  - do not send the query to a neighbour if TTL = 1 and the neighbour does not own a Leaf Node owning the content
    - bit=0 in all the positions corresponding to keywords which identify the content
CASE STUDY 2: KAZAA

- Created in March 2001 by Estonian programmers from BlueMoon Interactive (the same who developed Skype)
  - FastTrack protocol
    - uses encryption and not documented by its creators, closed source software clients
    - client-supernode communication has been reverse-engineered
    - but the supernode-supernode communication protocol remains largely unknown

- Two Tier Architecture (similar to Gnutella 0.6)
  - Ordinary Nodes (ON): notify metadata describing the shared content to the SN
  - Super Nodes (SN): connect the ON to the Kazaa overlay; maintain the overlay; track the metadata of the ON; query management
CASE STUDY 2: KAZAA

- **Load balancing:**
  - choice of SN on the basis of their workload

- **Connections Shuffling:**
  - the connections defining the overlay are continuously “shuffled”
  - a gossip approach
  - propagates the query in different parts of the overlay
  - detection of alternative peers if a disconnection occurs during the download

- **Locality exploitation**
  - host closeness on the basis of the network latency and topology
  - **advantage:** reduce the query reply time, because the traffic is contained within a single AS
  - **disadvantage:** the search is limited, the content search tends to remain within an AS
HINT FOR THE MID TERM: KAZAA

An hint for the mid term:

- an interesting approach based on reverse rengeneering the protocol
CASE STUDY 3: SKYPE

- Skype architecture is derived from Kazaa
  - developed by makers of Kazaa, now owned by Microsoft
  - exploits a proprietary protocol using encryption
    - no documentation, some analysis based on sniffer
  - centralized server for logging and billing

- SuperPeers responsible of:
  - looking for the peers with whom the call has to be established
  - forwarding calls and call traffic
    - act as relays for NATed peers
    - 60% of the user is behind a NAT

- Supports VOIP multi-user sessions
  - algorithm for merging different voice flows
CASE STUDY 3: SKYPE

1. Login procedure performed through super nodes
   find super nodes to perform login with by sending UDP packets to
   bootstrap (defaults) super nodes and wait for their responses

2. Establish TCP connections with retrieved super nodes (SNs) based on
   responses

3. Acquire the address of a login server, through the connected SNs, and
   authenticate user

4. Send UDP packets to a pre-set number of nodes to advertise presence (a
   backup connectivity list
   • Host Cache (HC) is a list of super node IP address and port pairs that
     Skype Client maintains
CASE STUDY 3: SKYPE CALL

- Case 1: caller & callee have public IP addresses
  - Caller establishes TCP connection with callee Skype client; UDP for media transfer
- Case 2: Caller is behind port-restricted NAT, callee has public IP
  - Caller uses online Skype super node to forward signalling packets over TCP
  - SN acts as a relay only for establishing the call
  - UDP direct communication is used for media content streaming
- Case 3: Both caller and callee behind port-restricted NAT and UDP restricted firewall
  - Exchange signalling info with a Skype super node using TCP
  - Caller sends media over TCP to an online node which forwards to callee via TCP and vice-versa
    - the SN acts as a full relay between NATed nodes A and B
    - A and B can exchange traffic because relay SN can bounce data back and forth. This solution is robust and fully functional but has the big disadvantage of involving SN with large workload.
HINT FOR THE MID TERM: SKYPE

An hint for the mid term:

CONCLUSIONS: HYBRID SYSTEMS

- Advantages
  - The presence of superpeers makes the network scale better
    - by reducing the number of network nodes involved in message handling and routing and reducing the actual traffic among them
    - low performance peers are not involved in query routing / keep alive messages
  - No single point of failure
  - Can provide anonymity
  - Can be adapted to special interest groups