Adaptive Lookup for Unstructured Peer-to-Peer Overlays

K Haribabu, Dayakar Reddy, Chittaranjan Hota
Computer Science & Information Systems
Birla Institute of Technology & Science
Antti Ylä-Jääski, Sasu Tarkoma
Telecommunication Software and Multimedia Laboratory
Helsinki University of Technology
Presented in

Second Workshop on Intelligent Networks: Adaptation, Communication & Reconfiguration (IAMCOM 2008)

held in conjunction with
3rd International Conference on COMmunication System softWAre and Middleware (COMSWARE 2008)

10 January 2008, Bangalore, India
Outline

- Introduction
- Research Gap
- System Model
- Experimental Results
- Summary
- References
Peer-to-Peer Overlay Networks

- Application-level logical networks built on top of the Internet
- Have own topology and routing

Benefits:
- No special administrative or financial arrangement. Self-organizing
- Adaptive
- Distributed and decentralized.
- Support the distribution of huge storage and computational problems.
Unstructured Overlays

- No fixed topology i.e. random topology
- No fixed placement of data
Structured Overlays

- Topology is tightly controlled
- Content is placed not at random peers but at specified locations
- Examples: Chord, CAN, Pastry, Tapestry
Chord: Structured Overlay

A node forwards a query for key $k$ to the node in its finger table with the maximum id less than $k$. Each forward will be closer to the destination. Lookup completely depends on successors.

“Where is key 80?”

“N90 has K80”
Research Gap

- Lack of QoS assurance routing
- Lack of user preference in determining goodness of neighbor
  - Routing indices approach uses number of documents and messages required to reach the documents [Crespo2002]
  - Intelligent search uses cosine similarity to choose neighbor [Kalogeraki2002]
  - Adaptive probabilistic search uses past query response and current query result [Tsoumakos2003]
  - In Ant based search goodness is judged by number of documents and path length [Michlmayr2006]
System Model

- We are looking at unstructured peer to peer overlay networks.
- QoS parameters we are looking at:
  - Bandwidth
  - Latency
  - Past response
- Each node:
  - has limited knowledge about other peers known as neighbors.
  - probes and stores the bandwidth and delay of the links leading to neighbors.
  - forwards queries to selected neighbors if it doesn’t find a match.
  - selects neighbors to forward a query using a composite function.
A-H are nodes
A is looking up object ‘X’.
H has the object ‘X’
System Model

- User while looking up object 'X' provides minimum BW and maximum link delay to requester node apart from the key words.
- The lookup of an object is by forwarding query to neighbors selected using composite function.
- At each node the query is forwarded to node N only if the link leading to N has the composite cost less than or equal to the composite cost provided by the user.
System Model

Composite function

- A weighted cost computation of bandwidth, link delay and past response

\[ \text{BW} \times W_{bw} + \text{latency} \times W_{latency} + \text{past response} \times W_{pr} \]

- The BW and link delay are link level constraints and past response is node level constraint

- While forwarding the query, the cost is computed at every link and added to the cost of already traversed path
System Model

Composite function

- The weighted past response is added to the cost at the requester node when it receives the query hit message.
- In this algorithm we try to minimize the composite function by selectively choosing the neighbors.
- To have uniform metric, the BW, latency, past response values are normalized.
System Model

Query
Query Hit
(Bandwidth in mbps, Latency in ms)

Min. Bandwidth: 2 Mbps
Max. link latency: 20 ms
Maximum Cost=6.25
System Model

For the network in the example, the importance of bandwidth availability is more when compared to latency and past response. The following weights are given:

- Bandwidth 65%
- Latency 20%
- Past response 15%

User constraints: BW 2Mbps, Latency 20 ms

There are two hit nodes F and H, and the cost for the paths is 17.50 and 17.05.

If the past response of F and H are 5 and 4, the costs of F and H will be updated to 18.19 and 17.80. So H node will be ranked as 1.
Pseudo Code

/* Pseudo code for a node which receives a query*/
ProcessQuery(Q){
    N: set of all neighbors
    bw: Array of bandwidths of links of neighbors
    ll: Array of link latencies links of neighbors
    Q: Query that has come from another node
    SN: Node that has sent the query
    MC: Maximum cost as per user requirements

    /* avoids loop */
    if Q.message id found in local cache then
        drop the query, exit
    end if
    if Q.keyword matches files in local db then
        make Query Hit Message
        send Query Hit Message to SN
    end if

    else if Q.hopcount = 0 then
        drop the query
    else
        store the Q.MessageId in cache
        for each neighbour in N
            if N <> SN then
                if computeCompositeCost(neighbor) <= MC then
                    CQ=copy of Q
                    CQ.hopcount=CQ.hopcount-1
                    CQ.cost=CQ.cost+cost of neighbor
                    forward CQ to neighbor
                end if
            end if
        end for
    end if
}
/* Function for computing cost of link to neighbor*/
ComputeCompositeCost(neighbor) {
    BW: bandwidth of link to neighbor
    LL: link latency
    NBW: normalized bandwidth
    NLL: normalized link latency
    MAXBW: Maximum bandwidth possible in the network
    MAXLL: Maximum link latency possible in the network

    /* The ranges are divided as follows
    rating 10: (0, MAXBW/10]
    rating 9: (1 * MAXBW/10, 2 * MAXBW/10]
    rating 8: (2 * MAXBW/10, 3 * MAXBW/10]
    ... (9 * MAXBW/10, MAXBW] = rating 1
    */

    NBW = rating of BW
    NLL = rating of LL

    return (0.65 * NBW + 0.20 * NLL)
}
/*pseudo code for requester node */

ReceiveQueryHit()

QH: Query hit message
N: Node form which query hit originated
i: rank
for each query hit QH received from node N
if N not found in local history cache then
    store the address of N in local history cache
end if
retrieve past response for N
past response = 0.8 * past response + 0.2 * rating for QH.number of documents matched
QH.cost = QH.cost + 0.15 * past response of N
save past response in history cache
end for

i=1
for each query hit in {QH sorted by QH.cost, QH.no of files in ascending order}
    assign rank i
end for
display results for user with ranks
}
Experimental Setup

Objective: To show that the proposed protocol consumes low bandwidth of the network and delivers the results according to user preferences.

1000 nodes and 50 different objects spread randomly across the network. The degree of node varies from 3 to 12 with average 6. Each peer has maximum of 15 objects.

The bandwidth and link latency is randomly assigned to each link. The bandwidth of the links is changed after every query during simulation to reflect the dynamic nature of the network congestion.

The TTL limit is varied from 1 to 5.
Experimental Results: Message Overhead

Flooding Vs QoS w.r.t message overhead
Numbers in the brackets indicate number of queries
Experimental Results

<table>
<thead>
<tr>
<th>Hop count</th>
<th>% reduction in message overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>74</td>
</tr>
<tr>
<td>5</td>
<td>81</td>
</tr>
</tbody>
</table>

As the hop count is increasing, the % reduction in bandwidth consumption is increasing. This is the reflection of exponential explosive nature of flooding.
Experimental Results: Hits

Flooding Vs QoS w.r.t hits
Numbers in the brackets indicate number of queries
Experimental Results: Unwanted Hits

Unwanted Query Hits

Flooding(30)  QoS (30)  Difference (30)

Flooding Vs QoS w.r.t. unwanted query hits
Numbers in the brackets indicate number of queries
Option Explicit

Private nom As Integer
Private i As Integer
Private NODENO As Integer
Private non As Integer
Private nok As Integer
Private nof As Integer
Private n() As node
Private bv() As Variant
Private id() As Variant
Private pr() As Variant
Private files() As String
Private kw() As String
Private msgids() As Integer

Private Sub Class_Initialize()
    nom = 0
    non = 0
    nok = 0
    nof = 0
    ReDim n(nom + 1)
    ReDim bv(nom + 1)
    ReDim id(nom + 1)
    ReDim pr(100, 100)
    ReDim kw(nok + 1)
    ReDim files(nof + 1)
    ReDim msgids(100, 3)
End Sub

Public Function getnodeo(nno) As Integer
    NODENO = nno
End Function

Public Function getnodeo() As Integer
    getnodeo = NODENO
End Function

Public Sub setPR(ByVal node As node, ByVal K As String, P As Variant)
    ' ReDim Preserve pr|non, nok
    Dim ni, ki
    ki = getNeighborIndex(node)

Public hopcount As Integer
Dim n(1000) As node
Dim kw(200) As String
Dim MAX_NODES As Integer
Public locount As Integer

Private Sub Form_Load()
locount = 0
End Sub

Private Sub init_Click()
MAX_NODES = 1000

Dim str As String
Open "topology.txt" For Output As #1

kw(0) = "urban"
kw(1) = "wmsd"
kw(2) = "ora"
kw(3) = "ate"
kw(4) = "bac"
kw(5) = "cat"
kw(6) = "doc"
kw(7) = "java"
kw(8) = "pink"
kw(9) = "pek"
kw(10) = "zip"
kw(11) = "vin"
kw(12) = "vem"
kw(13) = "lite"
kw(14) = "kits"
kw(15) = "mast"
kw(16) = "hrt"
kw(17) = "sit"
kw(18) = "kin"
kw(19) = "tin"
Summary

- We propose a QoS based adaptive heuristic search protocol.
- Objective of heuristics is to find routes that satisfy the user given constraints.
- The protocol reduces the bandwidth consumed by the network by reducing message overhead per lookup.
- The protocol also gives the ranking of hit nodes.
References


Thank You
Q&A