Querying Timely Information in Mobile Ad hoc Networks
Outline

- Motivation
- Previous work
- Proposed approaches
  - Adaptive Learning Approach to Discovering timely Information (ALADIN)
- Implementations
- Simulation results
- Future work
Motivation

Definition of traditional **service/resource discovery** in MANET

- Search process is completed if *any* instance of the service of interest is found

What if a querier would like to discover the *instance whose generation time is the closest to the queried time* of the time index information?

**Time indexed information**

- Information whose instance generation time is used as the **search key** for its access
- Scenario - Vehicle tracking in battlefield
  - Where is the vehicle *now*?
Challenges

- Search process is *not* completed even though an instance of the service of interest is found
  - The information holder does not know if its instance is the latest one
- Lack of centralize control or infrastructure
- Frequent topology changes
Taxonomies of traditional service discovery approaches

- Cross-layered design vs. directory-based
  - Based on message reduction scheme
- Directory-based approach
  - Organize nodes into logical structure for scalability
    - Subset of nodes form a virtual backbone
    - Service providers register their services on nearby backbone nodes
    - Query messages are distributed on backbone only
- Directory-less approach
  - Broadcast query messages on the entire network
  - Adopt various mechanisms to restrict range of query message distribution
    - ZRP-like approach
    - TTL control
Taxonomies of traditional service discovery approaches

- Cross-layered design vs. Layered design
  - Based on query/reply routing functionality
- Cross-layered design
  - Provide dedicate routing protocol for query/reply messages
- Layered design
  - Make use of existing MANET routing protocols to route query/reply messages
Design Goals

- **Scalable**
  - Without incurring network-wide flooding

- **Robust**
  - Have higher chance to discover the closest instance in the presence of mobility

- **Responsive**
  - Discover identity of holder of closest instance as well as route to the holder concurrently
Proposed approaches

Adaptive Learning Approach to Discovering timely INformation (ALADIN)

- Timely information
  - Special case of time indexed information
  - Contents age with time as newer instances of the information are produced
  - Latest instance is preferred

Self-Organized Mechanism to querying Time indexed Information (SOMTI)

- Discover an instance whose generation time is the closest to the queried time
Idea behind ALADIN

- Directory-based and cross-layered design approach
  - Without incurring network-wide flooding
  - Discover both identity and route to holder with latest instance concurrently

- Information holders act as directory agents
  - Holders periodically discover routes to others with newer instance
  - Queriers are able to discover the latest instance via *any* holder with (out-of-date) instance of requested information
    - Query messages can be progressively forwarded to the holder with the latest instance via these discovered routes
Query forwarding process
Design issues

- How does a holder discover route to another with newer instance?
- How is a query to be forwarded to a holder?
- ADADIN adopts an *intelligent random walk scheme* to deal with these two issues
  - Swarm intelligence
Swarm Intelligence

- Ants likely choose paths with higher pheromone intensity
- Trail gets reinforced (positive feedback)

Without reinforcement, pheromone evaporates (negative feedback)

Ants lay pheromone
Swarm Intelligence

Most ants follow trail with highest intensity

But some may choose alternate paths with small probability (amplification of fluctuation)
Swarm intelligence

- Positive/negative feedback
  - Search good solutions and stabilize the results

- Amplification of fluctuation
  - discover new solutions and adapt to changing environment

- Multiple interaction
  - Allows collaborations among distributed entities to coordinate and self-organize
Approach detail

- Query processing
- Query path improvement
holders periodically issue newer-instance-query to probabilistically chosen neighbors. A route is discovered once a newer-instance-reply is received. A latest-instance-query follows discovered routes to holder with latest instance.
New links are established due to node mobility. Better routes are discovered once newer-instance-reply is received. A latest-instance-query follows better routes to holder with latest instance. Holders issue newer-instance-query to alternative neighbor with small probability.
Implementation

- Message format and local structure
- Forwarding tendency management
  - Newer-instance-query forwarding
    - Probabilistically forwarding
  - Latest-instance-query forwarding
    - Deterministically forwarding
      - Forward received latest-instance-query to neighbor with highest tendency
- Newer instance query reduction
Message format

- Packet type
  - newer_instance_query
  - newer_instance_reply
  - latest_instance_query
  - latest_instance_reply
- Querier ID
- Sequence number
- Information ID
- Instance timestamp
- Hop count to holder
- Index of visited node
- List of visited nodes

<table>
<thead>
<tr>
<th>type</th>
<th>q_id</th>
<th>seq_no</th>
<th>info_id</th>
<th>i_time</th>
<th>q_list</th>
<th>n_index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Num_VNodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VisitedNode_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VisitedNode_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VisitedNode_n</td>
</tr>
</tbody>
</table>
Local structure

- Neighbor table
- Forwarding tendency table
  - Maintain forwarding tendencies of newer-instance-query message for neighbors
  - List of neighbors and their associated forwarding tendencies for information item $info$
    - $[nb, \text{tendency}(info, nb)]$
- Instance table
  - List of known instances
    - $[\text{gtime}(i), \text{hop\_cnt}(i), \text{next\_hop}]$
Forwarding tendency management

Positive feedback
- Forwarding tendency of a neighbor $B$ is increased once receiving messages other than latest-instance-query from $B$
- Messages other than latest-instance-query can be viewed as the advertisement of their originators’ instances

Negative feedback
- Forwarding tendency evaporation
  - Forwarding tendencies of all neighbors are decreased every period of time

Forwarding probability calculation
Positive feedback

\[ np_n = cp_n + (gt_i \times (\text{MAX\_HOPCNT} - \text{hop\_cnt}_i)) \]

- \( np_n \): New forwarding tendency of next hop \( n \)
- \( cp_n \): Current forwarding tendency of next hop \( n \)
- \( gt_i \): Generation time of new instance of holder \( i \)
- \( \text{MAX\_HOPCNT} \): Maximum hop count to holders of new instances
  - Messages are allowed to traverse \( \text{MAX\_HOPCNT} \) hops at most
- \( \text{hop\_cnt}_i \): Hop count to holder of new instance
Pheromone evaporation

\[ np_n = \begin{cases} 
\text{DEFAULT}_\text{PHR} & \text{if } cp_n * \text{evaporation\_factor} \leq \text{DEFAULT}_\text{PHR} \\
cp_n * \text{evaporation\_factor} & \text{if } cp_n * \text{evaporation\_factor} > \text{DEFAULT}_\text{PHR} 
\end{cases} \]

- \( np_n \): New forwarding tendency of next hop \( n \)
- \( cp_n \): Current forwarding tendency of next hop \( n \)
- \( \text{DEFAULT}_\text{PHR} \): Minimum of forwarding tendency
- \( \text{Evaporation\_factor} \): Predefined constant between 0 and 1
Probability of forwarding received newer-instance-query $m$ regarding information item $i$ to neighbor $n$ is calculated as

$$fp(i, n) = \begin{cases} 
0 & , n \in VN(m) \\
\frac{\sum_{n \in ntab - VN(m)} tend(i, n)}{\sum_{n \in VN(m)} tend(i, n)} & , n \notin VN(m) 
\end{cases}$$
Newer resource query reduction

Holder issue newer-instance-query every DEFAULT_Q_INTERVAL seconds if no newer instance is found

Time of issuing next newer-instance-query
- Current_time + (timestamp of latest discovered instance – its instance timestamp)
- Holders with newer instances issue newer-instance-query frequently while those with older instances issue newer-instance-query less frequently
Performance evaluation

- Compare ALADIN with flood-based approach (FLOOD)

FLOOD

- Broadcast query messages to the entire network and collect reply messages from all holders
- Discover identity and route to holder with latest instance
# Simulation parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terrain Dimension</td>
<td>1000 x 1000 m²</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>30 minutes</td>
</tr>
<tr>
<td>Communication Range</td>
<td>370 m</td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td>2 Mbps</td>
</tr>
<tr>
<td>MAC protocol</td>
<td>IEEE 802.11DCF</td>
</tr>
<tr>
<td>NUM QUERY</td>
<td>3</td>
</tr>
<tr>
<td>HOP LIMIT</td>
<td>10</td>
</tr>
</tbody>
</table>
## Simulation parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Q INTERVAL}</td>
<td>5 seconds</td>
</tr>
<tr>
<td>\textit{EXPIRE FACTOR}</td>
<td>0.9</td>
</tr>
<tr>
<td>\textit{EXPIRATION INTERVAL}</td>
<td>1 second</td>
</tr>
<tr>
<td>Average Instance Generation Rate</td>
<td>0.01 instance/sec</td>
</tr>
<tr>
<td>Average Query Generation Rate</td>
<td>0.5 – 5 query/sec</td>
</tr>
<tr>
<td>Mobility (Vmax)</td>
<td>0 – 10 m/sec</td>
</tr>
<tr>
<td># of nodes</td>
<td>50 – 200</td>
</tr>
</tbody>
</table>
Performance metrics

- **Overhead**
  - # of packets received and transmitted

- **Query success rate**
  - Percentage of successful query operations

- **Latest instance query latency**
  - Time for querying latest instance

- **Freshness of retrieved instance**
  - Time interval between the timestamp of the latest instance on the network and that of the “discovered” latest instance
    - The shorter the better
ALADIN saves at least 50% message overhead as compared to FLOOD.
Random walk and fewer reply generated make ALADIN have lower query success rate.

PF/NF, AF, MI help ALADIN improve query success rate.
FLOOD suffers from low freshness of discovered instance due to serious contentions and collisions.
Random walk make ALADIN have longer query latency

PF/NF, AF, MI help ALADIN improve query latency
Performance improvement

- Improve freshness of discovered instance in the case of high mobility
- ALADIN/Adaptive
  - Emphasize the impact of amplification of fluctuation
    - Holders aggressively discover alternative paths instead of reinforce discovered paths
  - Incorporate NLFF with forwarding tendency evaporation

\[
np_n = \begin{cases} 
  \text{DEFAULT\_PHR} & \text{if } cp_n \times \text{evaporation\_factor} \leq \text{DEFAULT\_PHR} \\
  \frac{cp_n \times \text{evaporation\_factor}}{\text{NLFF}} & \text{if } cp_n \times \text{evaporation\_factor} > \text{DEFAULT\_PHR}
\end{cases}
\]
Performance comparison
Observations

- Adaptive ALDAIN increases freshness of discovered instance while sacrificing
  - Overhead
  - Query latency
  - Query success rate

- Amplifying the impact of AF will degrades that of PF/NF
Conclusions

- ALADIN outperforms FLOOD in terms of overhead.
- ALADIN has a little bit lower query successful rate and longer query latency, but is more likely to discover fresher instance than FLOOD.

Performance improvement tradeoff among:
- Overhead
- Query success rate
- Query latency
- Freshness of discovered instance
Idea behind SOMTI

- Holders periodically discover routes to others whose instance generation time is the closest to a set of given time points before/after their own instance’s generation time
  - Pivot time points
  - Pivot holders

- Query messages can be progressively forwarded to the holder with closest instance to requested time point in the manner of \textit{N-ary search}
Concept of N-ary Search

Pivot time points = \{T(h), T(h)/2\}

Pivot time points = \{T(h), (T(h)+T'(h))/2\}, T'(h) = 9
Issues

- How does a holder discover routes to pivot holders?
  - Swarm intelligence
- Forwarding tendency management
- Query message reduction scheme
Message format and local structure

- SOMTI’s message format is similar to that of ALADIN, expect that one more field (requested time point) is added.

Packet types
- Closest-instance-query
- Closest-instance-reply
- Previous-instance-query
- Previous-instance-reply
- Next-instance-query
- Next-instance-reply
Forwarding tendency management

- Tendency table list
  - A list of tendency tables which each table is associated with generation time of an instance
  - \{tbl(t_1), tbl(t_2), tbl(t_3), \ldots\}

- Forwarding tendency table
  - A list of tendencies which each entry is associated with a neighbor
  - (nb, tendency(t_1, nb))
  - A new forwarding tendency table is created once receiving a unknown instance

- Tbl(t_i) is updated once
  - Receiving messages regarding instance generated at time t_i, expect closest-instance-query messages
    - Positive feedback
  - Every period of time
    - Negative feedback
Forwarding tendency update

\[
\text{goodness}(h, n) = \frac{\text{default\_tendency}}{HOP\_\text{LIMIT} - \text{hcnt}(n, h)}
\]

\[
\text{tend}(h, n) = \text{tend}(h, n) + \text{goodness}(h, n)
\]

\[
\text{tend}(h, n) = \frac{\text{tend}(h, n)}{\text{EXPIRE\_FACTOR}}
\]
Query messages forwarding

Choose $\text{tbl}(t_i)$ where $t_i$ is the closest to requested time point from tendency table list
- Default forwarding table is used if forwarding tendency table list is empty
  - Randomly forwarded

Closest-instance-query is forwarded to neighbor with highest forwarding tendency in $\text{tbl}(t_i)$

Previous/next-instance-query is forwarded to a probabilistically chosen neighbor in $\text{tbl}(t_i)$
Message reduction

Challenge
- ALADIN’s message reduction scheme cannot be adopted here
- Access pattern of queriers’ requested time points is unknown

Messages incurred by closest instance query
- Traversing long path incurs large message overhead and suffers from long querying latency and low query successful rate

Messages incurred by previous/next instance query
- Number of messages increases explosively as the number of holders grows if previous/next instance queries are issued in fixed rate
Observations

Best case

Request time point = 0.5

Worse case
Message reduction scheme

- Holders with old instances send out more previous instance queries and less next instance queries.
- Holders with new instances send out more next instance queries and less previous instance queries.
- Instance advertisement
- Message reduction
Scheduling next previous/next instance queries

\[
T_{\text{prev}} = \begin{cases} 
T + Q - \text{INTERVAL} & \text{ogt}(h) = \text{NULL} \\
T + (\text{igt}(h) - \text{ogt}(h)) & \text{Otherwise}
\end{cases}
\]

\(T: \) time of issuing previous instance query

\(T_{\text{prev}}: \) time of issuing next previous instance query

\(\text{igt}(h): \) instance generation time of holder \(h\)

\(\text{ogt}(h): \) generation time of the oldest instance holder \(h\) is aware of

\[
T_{\text{next}} = \begin{cases} 
T' + Q - \text{INTERVAL} & \text{lg} \, t(h) = \text{NULL} \\
T' + (\text{lg} \, t(h) - \text{igt}(h)) & \text{Otherwise}
\end{cases}
\]

\(T': \) time of issuing previous instance query

\(T_{\text{next}}: \) time of issuing next next instance query

\(\text{igt}(h): \) instance generation time of holder \(h\)

\(\text{lg} \, t(h): \) generation time of the latest instance holder \(h\) is aware of
Performance metrics

Message overhead
- Average number of messages sent per successful query operation

Query successful rate

Query latency

Goodness of discovered instance
- Difference between generation time of discovered instance and generation time of the closest instance
Competitors of our approach

- **FLOOD with reply reduction**
  - A query carries the closest instance generation time of the holder it has visited
  - A reply is suppressed if a "closer" reply has been sent back to the querier

- **Adaptive dynamic backbone (ADB) approach**
  - Clustering scheme
  - Directory-based approach
Message overhead
Query successful rate

![Graph showing query success rate vs number of nodes]

Query Success Rate (%) vs Number of Nodes

![Graph showing query success rate vs query generation rate]

Query Success Rate (%) vs Query Generation Rate

![Graph showing query success rate vs mobility speed]

Query Success Rate (%) vs Mobility Speed (m/s)
Querying latency
Goodness of discovered instance

- Graph 1: Average Time Differences vs. Number of Nodes
  - QFLOOD
  - ALADIN
  - DIR/QUERY

- Graph 2: Average Time Differences vs. Query Generation Rate
- Graph 3: Average Time Differences vs. Mobility Speed (m/s)

Parameters:
- timedef.node.speed-0.service-0.01.query-1.0
- timedef.node.speed-0.01.service-0.01.query-1.0
- timedef.speed.node-50.service-0.01.query-1.0
Conclusions

- Our approach for discovering time indexed information enables queries to be forwarded in the manner of \textit{N-ary search}
  - Avoid network-wide flooding

Simulation results show that

- Our approach issues less messages compared to FLOOD, and comparable number of messages as compared to directory-based approach
- Our approach always discovers better instances than FLOOD and directory-based approach