Network Abstractions for Context-Aware Mobile Computing

Gruia-Catalin Roman
Christine Julien
Qingfeng Huang

Mobile Computing Laboratory
In the highly dynamic ad hoc environment, both hardware and software components are constantly moving and changing.

In context-aware computing, application components adapt to their environment.

We extend the scope of a component’s context to include information anywhere in the network.

Our new notion of context is also location and motion sensitive.
Objectives

- Simplify mobile application development
  - Provide generalized view of context extended to include information about distant hosts
  - Provide more flexibility to applications by allowing application specific context specifications
  - Provide transparent maintenance of these contexts
  - Help programmer cope with problems inherent in ad hoc networks
Motivating Example
Solution Strategy

- Allow computational components to have multiple contexts that change over time
  - Expand contexts to encompass a neighborhood surrounding the specifying component
- Provide a mechanism for declarative specifications of these contexts
- Provide a protocol for calculating contexts
Acquaintance List Problem

- Acquaintance List – set of nodes that contribute information used to build a context for a reference node
- Network Abstraction – key concept that allows specification of this list
  - Treat the network as a graph and impose application specific metric to assign weights
  - Compute acquaintance list using an application supplied function that determines costs of paths in this graph
Solution Sketch

1. Represent the ad hoc network as a graph
2. Abstract properties of nodes and links to a weight on an edge in this graph
3. Calculate the cost of paths from the reference node
4. Determine the shortest possible path to each node, and build a tree of these paths
5. Limit this tree using a bound, D
Limiting the Scope

- Define a subtree of the network by placing a bound on the cost of each path from the reference.
- The bound is useful only if the value of the shortest path is strictly increasing.

Note: All links have a weight of 1.
Minimum Cost Path

- Given multiple paths to a node, choose the shortest one

Note: All links have a weight of 1
Computing Path Cost

- Assuming each link has a single weight, an application can define a cost function:
  - \( f_{v_0}(P_k) = Cost(f_{v_0}(P_{k-1}), m_{k-1,k}) \)
  - \( f_{v_0}([v_0]) = 0 \)
- Cost function must be strictly increasing along a path
Weight Assignment

- Individualized for a particular application
- Host properties combine to form $r_i$; link properties combine to form $w_{ij}$
  - $r_i$ can include battery power, CPU power, physical location, and other host properties
  - $w_{ij}$ can include physical distance, bandwidth, throughput, and other link properties
- Each link’s weight combines the properties of the link in the physical network and the properties of both nodes connected by the link
  \[ m_{ij} = G(r_i, r_j, w_{ij}) \]
Context Calculation

- Service new queries
  - Respond at the application level
  - Remember the cost, the parent, the sequence number, and information about the computation
  - Propagate the query
- Service shorter distances (same sequence number)
  - Remember the cost and the new parent
  - Propagate the query
- Disregard longer distances (same sequence number)
  - But remember the cost if within bound
- Halt when computation reaches boundary
- Similar to ad hoc routing protocols
Protocol Example

Query = (source, new cost, ...)  
State = [cost, parent, ...]

{(neighbor, cost through neighbor), ...}  

Bound = 6
Cost = additive
Context Maintenance

- Persistent queries require context maintenance
- During context calculation, remember costs to other neighbors
- React to increase in link weights
  - If parent, adjust cost and propagate information
  - Otherwise update local information
- React to decreases in link weights
  - If non-parent link, recalculate the cost for the neighbor on the other end
- Regain resources when context is no longer used
Maintenance Example

Query = (source, source cost, new cost, ...)
State = [cost, parent, ...]
{(neighbor, cost through neighbor), ...}

Bound = 6
Cost = additive
Conclusions and Future Work

- Declarative specifications provide flexible and general treatment of context
- Protocol shows feasibility of transparent context maintenance

- Weaken strong atomicity assumptions
  - E.g., we assume each configuration change to be atomic
- Optimization opportunities that consider interactions between multiple contexts
Thank you

- For more information:
  - http://www.cs.wustl.edu/mobilab