Designing an asynchronous group communication middleware for wireless users

Surendar Chandra, Unaffiliated

Xuwen Yu, VMware

Group communication middleware

- group of users, each creating updates
- Goal: design middleware to propagate update from each user to all other group members
  - Our middleware: delivery order unimportant
    - application can order updates

- mechanisms:
  - Synchronous: members simultaneously available
  - Asynchronous: eventually propagate to all users

- performance depends on user availability

Behavior for wireless users

- modern users wireless: we focus on WLAN users
  - @Notre Dame: 44% of devices wireless (incl. servers)
- users operate from many places: home, work …
- availability traces used:
  - University (Notre Dame): Zeroconf: 12/07 – 8/08
  - Corporate (IBM Research)*: SNMP, AP: 7/02 – 8/02
  - Hotspot federation (Île Sans Fil)*: auth log: 8/04 - 8/07

* - CRAWDAD archive

User availability characteristics

- diurnal variation, weekday/weekend variation
- small median session and getting smaller
  - 2.8 hrs- corporate, 35 min - hotspot, 20 min - university
- large duration between session
  - 3.5 hrs- corporate, 9.6 hrs- hotspot, 1.78 hrs- university
- session overlap minimal
  - cannot sustain synchronous communications
- significant node churn

Policies investigated

- server mediated: always-ON servers host updates
- distributed: propagate via other group members

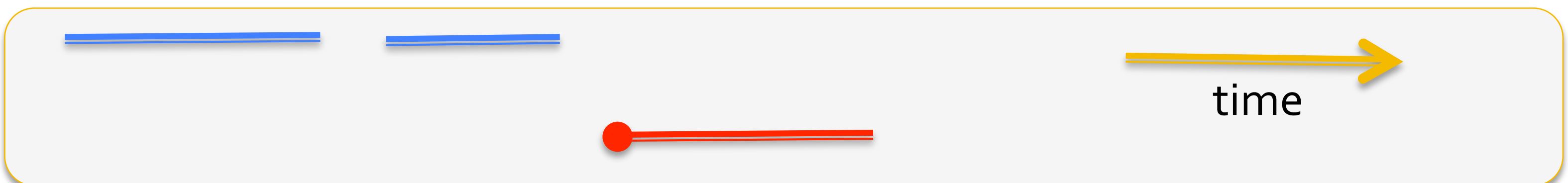
- initiator: periodic push or pull with online users
  - Svr-ServInit: server pulls (and pushes) updates
  - Svr-NodeInit: users push (and pull) updates to server
  - P2P-Pull: distributed pull from other users
  - P2P-Push: distributed push to other users

Practical policy parameters

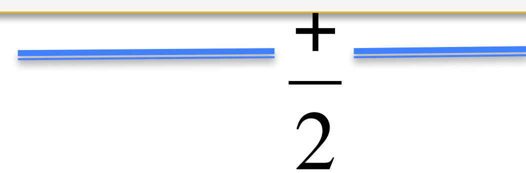
- when to propagate updates
  - first: when online or at fixed times
  - next: periodically : 5, 15, 30, 60 mins
    - adaptive policy based on prior history
  - final: not explicitly before going offline
- % of neighbors – prior work showed reducing # neighbors while increasing freq. beneficial
- update propagation delay not considered

Performance metrics

- lagAmount: measures entropy
  - average amount of updates unavailable at a node
  - assume update creation rate is constant
    - amount of updates = duration online without update



    - at , lagAmount =



- # gossips & # unnecessary gossips
  - unnecessary if no updates routed using distributed
  - unnecessary if no new updates in Serv-NodeInit

Questions investigated and results

details in paper

1. *Gossips considered ill-suited for quick dissemination*. Quantify conventional wisdom

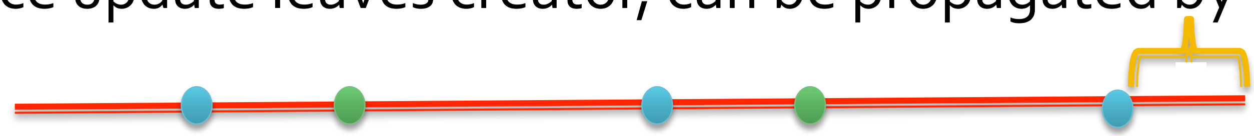
- entropy (lagAmount) depends on:
  - user churn and time between sessions
    - cannot propagate updates to unavailable users
    - users that left will never receive updates
  - amount of updates depend on session duration
- entropy high for best case (Svr-ServInit, delay=0)
  - corporate: high availability during weekdays
- Relative overhead: distributed competitive

Questions investigated and results

2. Are push & pull mechanisms complementary?

- No, pull better randomizes update propagation
  - updates created after last propagation increases entropy, especially when large duration bet. sessions
  - push: last propagation decided by own frequency
    - too frequent = high gossips

  - pull: last propagation decided by group (random)
    - once update leaves creator, can be propagated by others



  - push+pull: higher cost

Questions investigated and results

3. tradeoffs for more frequently propagating messages to fewer nodes
   - simultaneously available nodes small, better to send to everyone (correspondingly less frequent)

- further details in paper