Collecting and analysing data over smart participatory networks

Radu-Ioan Ciobanu
Faculty of Automatic Control and Computers
University Politehnica of Bucharest
radu.ciobanu@cs.pub.ro

Introduction

- Mobile devices collect mobility data, but what can we do with it?
- Create an Android app, launch it, and hope for the best?
- No!
  - Experiments usually involve hundreds or thousands of devices
  - Creating efficient routing and dissemination solutions for ONs can be difficult
  - Frameworks that allow the testing of such solutions before deploying them in real-life are extremely useful
  - Developers can test their solutions and tweak them in controlled scenarios, without having to incur high costs (in terms of money and time)
  - **Evaluate in simulation, and only then in live research**

MobEmu

- Opportunistic framework used for replaying mobility traces/models and emulating data routing and dissemination algorithms
- Can run a user-created algorithm on a desired mobility trace or synthetic model, as long as certain implementation rules are followed
- Written in Java
  - highly modular
  - easy to understand and/or modify
- https://github.com/raduciobanu/mobemu

Motivation

- An alternative to ONE (Opportunistic Network Environment), which has some caveats:
  - no support for **data dissemination**
  - **community detection** not implemented
  - no **social relationships** information
  - **selfishness** not modelled
  - **context data** absent
- Other solutions include ns-2, ns-3, OMNet++, but:
  - they do not offer complete support for ON simulations
  - their granularity levels can lead to unnecessary complexities and various challenges in running large experiments

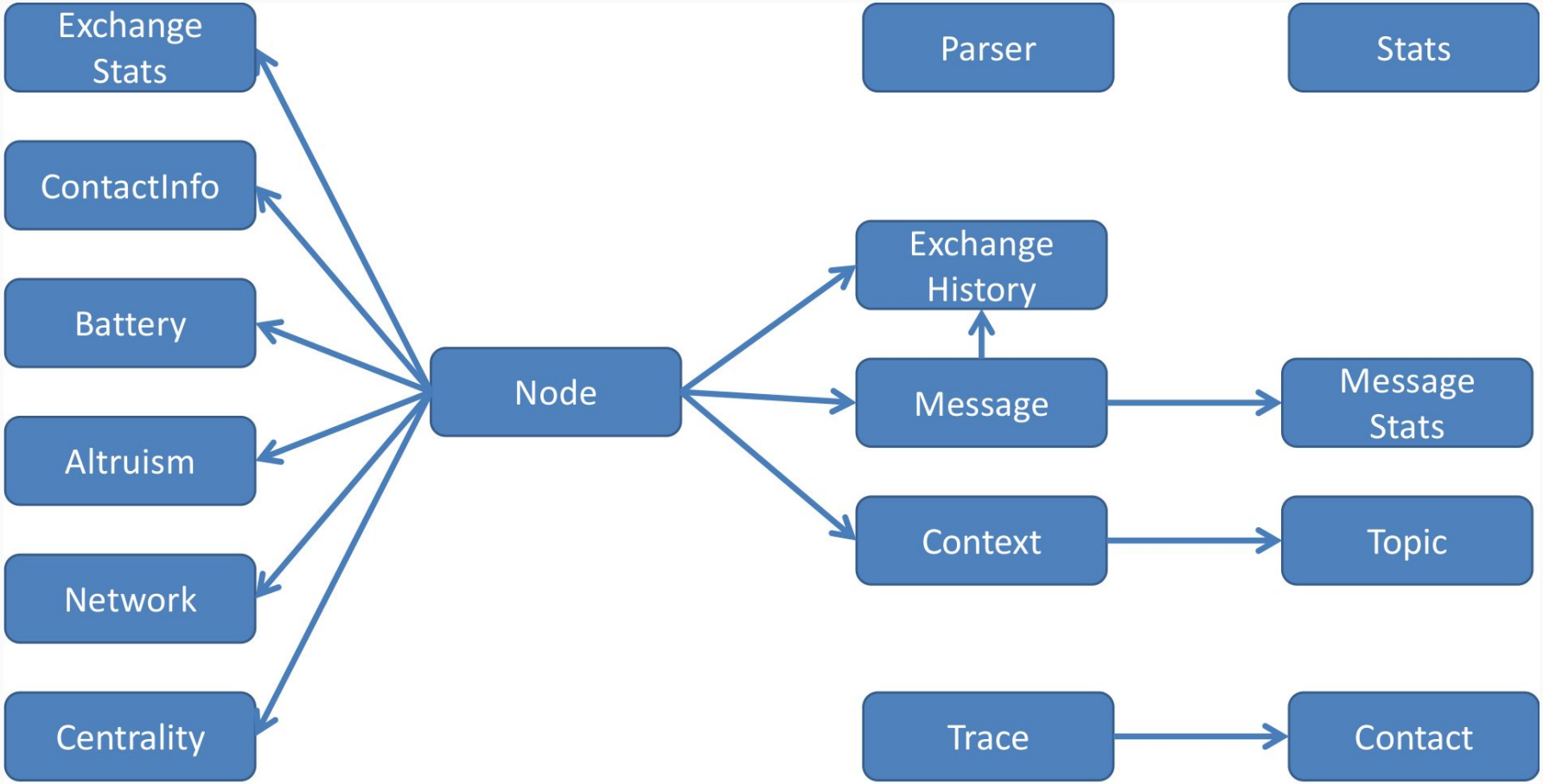
Functionality

- Parses a **mobility trace** or runs a **synthetic model**
- At every step of the trace:
  - checks whether a **contact** between two nodes occurs
  - checks if nodes should generate **messages**
  - computes a node's **community** and **centrality**
- If a contact occurs, a **routing** or **dissemination** algorithm is applied for each node
- Various **statistics** are collected

Functionality (2)

- User can control:
  - data memory size
  - stored contact history size
  - network speed
  - altruism level
  - battery behavior
  - number of messages generated
  - message destinations
  - etc.

Components


Exchange Stats
ContactInfo
Battery
Altruism
Network
Centrality
Node
Parser
Stats
Exchange History
Message
Message Stats
Context
Topic
Trace
Contact

Components (2)

- `Trace` - list of `Contacts` (node IDs, timestamps)
- `Parser`
  - `getTraceData`
  - `getContextData` - map of `Context` objects (node ID, set of `Topics`)
  - `getSocialNetwork`
  - `getNodesNumber`
- **HCMM** is also implemented
  - synthetic mobility model
  - social relations
  - physical locations

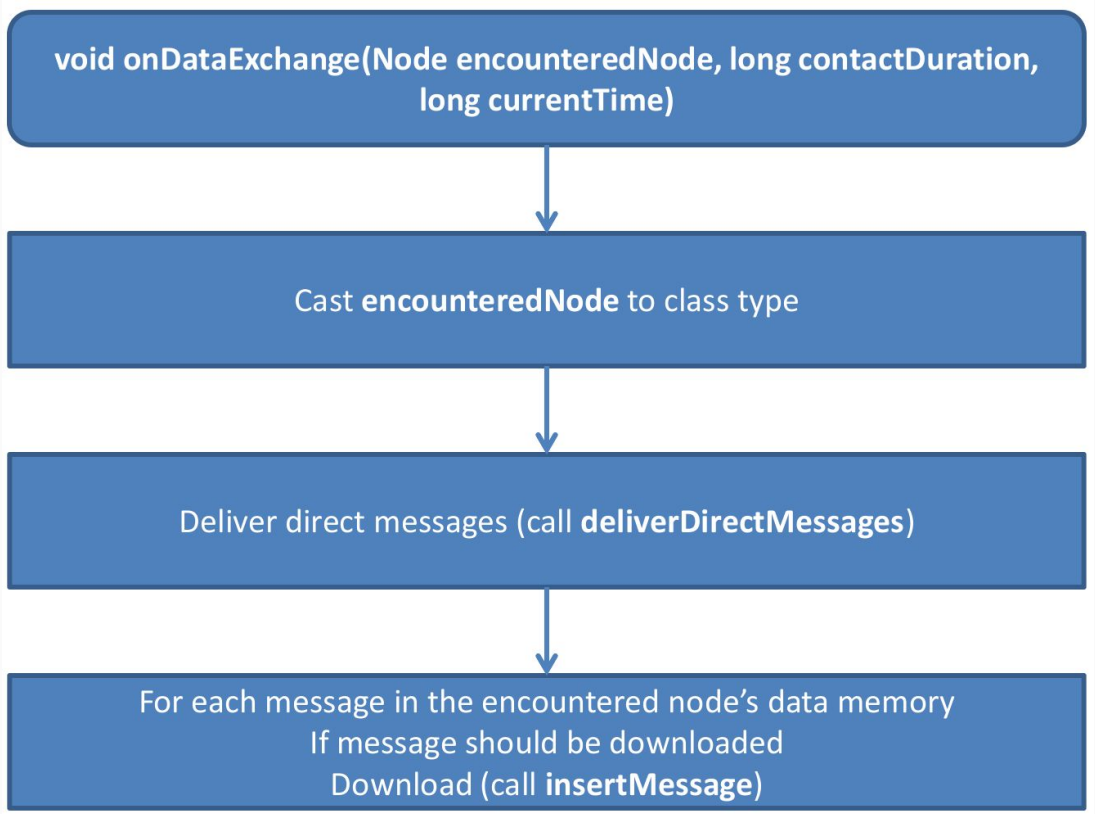
Trace	Devices		Duration (days)	Communication	Trace type	Social data	Interest data
	Mobile	Fixed					
St. Andrews	27	0	79	Bluetooth	Academic and urban	Yes	No
Intel	8	1	6	Bluetooth	Academic	No	No
Cambridge	12	0	5	Bluetooth	Academic	No	No
Infocom	41	0	4	Bluetooth	Conference	No	No
Infocom 2006	78	20	4	Bluetooth	Conference	No	Yes
Content	36	18	25	Bluetooth	Urban	No	No
UPB 2011	22	0	25	Bluetooth	Academic	Yes	No
UPB 2012	66	0	64	Bluetooth and Wi-Fi	Academic	Yes	Yes
Sigcomm 2009	76	0	3	Bluetooth	Conference	Yes	Yes
NUS	22341	0	118	Student schedule	Academic	No	No
GeoLife	182	0	1885	GPS	Urban	No	No
SocialBlueConn	15	0	9	Bluetooth	Academic	Yes	Yes
NCCU	115	0	15	Bluetooth	Academic	No	No

Components (3)

- `Node`
  - data memory, own memory - lists of `Messages` (ID, source, destination, `Context`)
  - contact history - map of `ContactInfo` objects
  - data exchange history - `ExchangeHistory`, `ExchangeStats`
  - social community information - `Centrality`, `CommunityDetection`
  - social network connections - array of booleans
  - interests - `Context`
  - battery information (drain rate, charge duration, usability) - `Battery`
  - network information (messages per contact) - `Network`
  - selfishness information - `Altruism`

Routing and dissemination

Inherit from the `Node` class and implement the `onDataExchange` function



Routing and dissemination (2)

- Algorithms implemented:
  - Epidemic
  - BUBBLE Rap
  - Spray and Wait/Focus
  - ML-SOR
  - Social Trust
  - JDER
  - IRONMAN
  - SENSE
  - SPRINT
  - ONSIDE
  - Interest Spaces
- Metrics analyzed:
  - hit rate
  - delivery cost
  - delivery latency
  - hop count
- Context data:
  - social network and communities
  - history of contacts/exchanges
  - battery
  - selfishness
  - interests

Example

- **Epidemic** algorithm on the **UPB 2012** trace

Assignment

- Implement your own routing or dissemination solution and compare it to an existing algorithm

Thank you!

radu.ciobanu@cs.pub.ro