Opportunity is the Mother of Invention

How Personal Delay Tolerant Networking led to Data Centric Networking & Understanding Social Networks.

Jon Crowcroft
Jon.crowcroft@cl.cam.ac.uk
Outline Narrative History of Haggle

1. Haggle Software Architecture
2. How we got to Declarative Data Driven Nets
3. Why we got diverted into Social Networks
The Internet Protocol Hourglass (Deering)
Putting on Weight

- requires more functionality from underlying networks
Mid-Life Crisis

- doubles number of service interfaces
- requires changes above & below
- major interoperability issues
Give it to me, I have 1G bytes phone flash.

I have 100M bytes of data, who can carry for me?

I can also carry for you!

Thank you but you are in the opposite direction!

Don't give to me! I am running out of storage.

Reach an access point.

Finally, it arrive...

Internet

Search La Bonheme.mp3 for me

There is one in my pocket...

Search La Bonheme.mp3 for me

Search La Bonheme.mp3 for me
1. Motivation 2001-2004

- Mobile users currently have a very bad experience with networking
  - Applications do not work without networking infrastructure such as 802.11 access points or cell phone data coverage
  - Local connectivity is plentiful (WiFi, Bluetooth, etc), but very hard for end users to configure and use

- Example: Train/plane on the way to London
  - How to send a colleague sitting opposite some slides to review?
  - How to get information on restaurants in London? (Clue: someone else is bound to have it cached on their device)

- Ad Hoc Networks were a complete washout
  - Failed to account for heavy tailed density distribution
  - Use of 802.11 as radio was at best misguided.
Underlying Problem

- Applications tied to network details and operations via use of IP-based socks interface
  - What interface to use
  - How to route to destination
  - When to connect

- Apps survive by using directory services
  - Address book maps names to email addresses
  - Google maps search keywords to URLs
  - DNS maps domain names to IP addresses

- Directory services mean infrastructure
Phase transitions and networks

- **Solid networks:** wired, or fixed wireless mesh
  - Long lived end-to-end routes
  - Capacity scarce

- **Liquid networks:** Mobile Ad-Hoc Networking (MANET)
  - Short lived end-to-gateway routes
  - Capacity ok (Tse tricks with power/antennae/coding)

- **Gaseous networks:** Delay Tolerant Networking (DTN), Pocket Switched Networking (PSN)
  - No routes at all!
  - Opportunistic, store and forward networking
  - One way paths, asymmetry, node mobility carries data
  - Capacity Rich (Grossglauser&Tse) (but latency terrible... ... ...)

- **Haggle targets all three,** so must work in most general case, i.e. “gaseous”
Decentralisation & Disconnectivity

- Absence of infrastructure for
  - Routing, searching, indexing
  - Names, Identity, Currency

- When everything’s adhoc, even pagerank has to be
  - Bad joke about french pronunciation of “Haddock”

- As early pub/sub systems, interest itself is data
  - So we take event/notify+pub/sub and apply to
  - Discovery of users, nodes, routes, interest
  - everyone soaks it all up and runs ego-centric pagerank
Current device software framework

Application

File System

Networking

User Data

App logic + GUI

Protocol

Delivery (IP)

Interfaces

Isolated from network

App has two orthogonal parts

Synchronous, node-centric API

Delivery uses anonymous IP
Haggle framework design

Applications

Haggle

App Logic + GUI

Less work for new app developers

Not tied to one app; exposed metadata

Asynchronous, data-centric API

Key component missing before

Multiple protocols usable for each task
Data Objects (DOs)

• DO = set of attributes = \{type, value\} pairs
  • Exposing metadata facilitates search
  • Another bad (Diot) joke

• Can link to other DOs
  • To structure data that should be kept together
  • To allow apps to categorise/organise

• Apps/Haggle managers can “claim” DOs to assert ownership

<table>
<thead>
<tr>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO-Type</td>
</tr>
<tr>
<td>Content-Type</td>
</tr>
<tr>
<td>From</td>
</tr>
<tr>
<td>To</td>
</tr>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>Body</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO-Type</td>
</tr>
<tr>
<td>Content-Type</td>
</tr>
<tr>
<td>Keywords</td>
</tr>
<tr>
<td>Creation time</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>
DO Filters

• Queries on fields of data objects
• E.g. “content-type” EQUALS “text/html” AND “keywords” INCLUDES “news” AND “timestamp” >= (now() – 1 hour)
• DO filters are also a special case of DOs
• Haggle itself can match DOFilters to DOs – apps don’t have to be involved
• Can be persistent or be sent remotely...
**DO Filter is a powerful mechanism**

<table>
<thead>
<tr>
<th></th>
<th>One-Off</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local</strong></td>
<td>“Desktop” Search (find mp3s with artist “U2”)</td>
<td>Listen (wants to receive webpages)</td>
</tr>
<tr>
<td><strong>Remote</strong></td>
<td>“Web” Search (find “london restaurants”)</td>
<td>Subscribe (send all photos created by user X to X’s PC)</td>
</tr>
</tbody>
</table>
Layerless Naming

• Haggle needs just-in-time binding of user level names to destinations

• Q: when messaging a user, should you send to their email server or look in the neighbourhood for their laptop’s MAC address?
  • A: Both, even if you already reached one. E.g. you can send email to a server and later pass them in the corridor, or you could see their laptop directly, but they aren’t carrying it today so you’d better email it too…

• Current layered model requires ahead-of-time resolution by the user themselves in the choice of application (e.g. email vs SMS)
Name Graphs comprised of Name Objects

- Name Graph represents full variety of ways to reach a user-level name
- NO = special class of DO
- Used as destinations for data in transit
- Names and links between names obtained from
  - Applications
  - Network interfaces
  - Neighbours
  - Data passing through
  - Directories

<table>
<thead>
<tr>
<th>DO-Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>James Scott</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DO-Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>00:0E:F6:23:91:34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DO-Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><a href="mailto:jamesscott@acm.org">jamesscott@acm.org</a></td>
</tr>
</tbody>
</table>
Forwarding Objects

- Special class of DO used for storing metadata about forwarding
  - TTL, expiry, etc
- Since full structure of naming and data is sent, "intermediate" nodes are empowered to:
  - Use data as they see fit
  - Use up-to-date state and whole name graph to make best forwarding decision
Connectivities and Protocols

- Connectivities (network interfaces) say which “neighbours” are available (including “Internet”)
- Protocols use this to determine which NOs they can deliver to, on a per-FO basis
  - P2P protocol says it can deliver any FO to neighbour-derived NOs if corresponding neighbour is visible
  - HTTP protocol can deliver FOs which contain a DOFilter asking for a URL, if “Internet” neighbour is present
- Protocols can also perform tasks directly
  - POP protocol creates EmailReceiveTask when Internet neighbour is visible
## Forwarding Algorithms

Forwarding algorithms create Forwarding *Tasks* to send data to suitable next-hops. They can also create Tasks to perform signalling. Many forwarding algs can run *simultaneously*.

### Table

<table>
<thead>
<tr>
<th>Protocol, Name, Neighbour</th>
<th>FOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

*algorithm 1*

*algorithm 2*

x = scalar

“benefit” of forwarding task
Aside on security etc

- Security was “left out” for version 1 in this 4-year EU project, but threats were considered
- Data security can reuse existing solutions of authentication/encryption
  - With proviso that it is not possible to rely on a synchronously available trusted third party
- Some new threats to privacy
  - Neighbourhood visibility means trackability
  - Name graphs could include quite private information
- Incentives to cooperate an issue
  - Why should I spend *any* bandwidth/energy on your stuff?
- Did address later (Social Nets 2009-2011)
  - see safebook.us by Eurecom folks...
D³N*

2. Programming Distributed Computation in Pocket Switched Networks (CCN/NDN etc)

* Data Driven Declarative Networking
PSN: Dynamic Human Networks

- Topology changes every time unit
- Exhibits characteristics of Social Networks
Time Dependent Networks

- Data paths may not exist at any one point in time but do exist over time
- Delay Tolerant Communication
Regularity of Network Activity

- Size of largest fragment shows network dynamics

![Graph of 5 Days activities in Bath Trace](image1)

![Graph of Tuesday in Bath Trace](image2)

5 Days

Tuesday
Haggle Node Architecture

- Each node maintains a data store: its current view of global namespace
  - Persistence of search: delay tolerance and opportunism
- Semantics of publish/subscribe and an event-driven + asynchronous operation

- Multi-platform
  (written in C++ and C)
  - Windows mobile
  - Mac OS X, iPhone
  - Linux
  - Android
D³N Data-Driven Declarative Networking

• How to program distributed computation?
  • Use Declarative Networking?
• The Vodafone Story....
  • Need tested or verified code....so also good...
Declarative Networking

- Declarative is new idea in networking
  - e.g. Search: ‘what to look for’ rather than ‘how to look for’
  - Abstract complexity in networking/data processing
- **P2**: Building overlay using Overlog
  - Network properties specified declaratively
- **LINQ**: extend .NET with language integrated operations for query/store/transform data
- **DryadLINQ**: extends LINQ similar to Google’s Map-Reduce
  - Automatic parallelization from sequential declarative code
- **Opis**: Functional-reactive approach in OCaml
**D³N Data-Driven Declarative Networking**

- How to program distributed computation?

- Use Declarative Networking
  - Use of Functional Programming
    - Simple/clean semantics, expressive, inherent parallelism
  - Queries/Filer etc. can be expressed as higher-order functions that are applied in a distributed setting

- Runtime system provides the necessary native library functions that are specific to each device
  - Prototype: F# + .NET for mobile devices
D$^3$N and Functional Programming I

- Functions are first-class values
  - They can be both input and output of other functions
  - They can be shared between different nodes (code mobility)
  - Not only data but also functions flow
- Language syntax does not have state
  - Variables are only ever assigned once; hence reasoning about programs becomes easier
    (of course message passing and threads $\rightarrow$ encode states)
- Strongly typed
  - Static assurance that the program does not ‘go wrong’ at runtime unlike script languages
- Type inference
  - Types are not declared explicitly, hence programs are less verbose


**D³N and Functional Programming II**

- Integrated features from query language
  - Assurance as in logical programming
- Appropriate level of abstraction
  - Imperative languages closely specify the implementation details (how); declarative languages abstract too much (what)
  - Imperative – predictable result about performance
  - Declarative language – abstract away many implementation issues
## Overview of D³N Architecture

- Each node is responsible for storing, indexing, searching, and delivering data
- Primitive functions associated with core D³N calculus syntax are part of the runtime system
- Prototype on MS Mobile .NET
D³N Syntax and Semantics I

- Very few primitives
  - Integer, strings, lists, floating point numbers and other primitives are recovered through constructor application
- Standard FP features
  - Declaring and naming functions through let-bindings
  - Calling primitive and user-defined functions (function application)
  - Pattern matching (similar to switch statement)
  - Standard features as ordinary programming languages (e.g. ML or Haskell)
D³N Syntax and Semantics II

• Advanced features
  • Concurrency (fork)
  • Communication (send/receive primitives)
  • Query expressions (local and distributed select)
Runtime System

- Language relies on a small runtime system
  - Operations implemented in the runtime system written in F#
- Each node is responsible on data:
  - Storing
  - Indexing
  - Searching
  - Delivering
- Data has Time-To-Live (TTL)
- Each node propagates data to the other nodes.
- A search query w/TTL travels within the network until it expires
- When the node has the matching data, it forwards the data
- Each node gossips its own metadata when it meets other nodes
Example: Query to Networks

- Queries are part of source level syntax
  - Distributed execution (single node programmer model)
  - Familiar syntax

D³N: \texttt{select name from poll() where institute = “Computer Laboratory”}

F#: \texttt{poll()}
   \texttt{|> filter (fun r -> r.institute = “Computer Laboratory”)}
   \texttt{|> map (fun r -> r.name)}

Message: \texttt{(code, nodeid, TTL, data)}
Example: Vote among Nodes

• Voting application: implements a distributed voting protocol of choosing location for dinner

• Rules
  • Each node votes once
  • A single node initiates the application
  • Ballots should not be counted twice
  • No infrastructure-base communication is available or it is too expensive

• Top-level expression
  • Node A sends the code to all nodes
  • Nodes map in parallel (pmap) the function `voteOfNode` to their local data, and send back the result to A
  • Node A aggregates (reduce) the results from all nodes and produces a final tally
Sequential Map function (smap)

- Inner working
  - It sends the code to execute on the remote node
  - It blocks waiting for a response waiting from the node
  - Continues mapping the function to the rest of the nodes in a sequential fashion
  - An unavailable node blocks the entire computation

```ocaml
let rec smap f lst =  // Sequential map
  match lst with
  | [] → []
  | n::ns → send f n; receive n :: smap f ns
```
Parallel Map Function (pmap)

- Inner working
  - Similar to the sequential case
  - The send/receive for each node happen in a separate thread
  - An unavailable node does not block the entire computation

```ocaml
let rec pmap f lst = // Parallel map
  match lst with
  | [] → []
  | n ⊹ ns →
    fork (fun () →
      send f n; receive n
    ) :: pmap f ns
```
Reduce Function

• Inner working
  • The reduce function aggregates the results from a map
  • The reduce gets executed on the initiator node
  • All results must have been received before the reduce can proceed

```
let rec reduce f se lst = // Reduce with starting element
  match lst with
  | [] → se
  | x::xs → f x (reduce f se xs)
```
Voting Application Code

```ocaml
type ballot = { locationA : int; locationB : int };
let emptyBallot = { locationA = 0; locationB = 0 };
let graph = getSocialGraph();
let voteForA() : ballot = { locationA = 1; locationB = 0 };
let voteForB() : ballot = { locationA = 0; locationB = 1 }

let rec smap f lst = // Sequential map
  match lst with
  | [] → []
  | n::ns → send f n;receive n :: smap f ns

let rec pmap f lst = // Parallel map
  match lst with
  | [] → []
  | n :: ns →
    fork (fun () →
      send f n;receive n
    ) :: pmap f ns

let rec reduce f se lst = // Reduce with starting element
  match lst with
  | [] → se
  | x::xs → f x (reduce f se xs)

let countVote (b1:ballot) (b2:ballot) : ballot =
  { locationA = b1.locationA + b2.locationA;
    locationB = b1.locationB + b2.locationB }
reduce countVote emptyBallot (pmap voteOfNode graph)
```
Outlook and Future Work

• Current reference implementation:
  • F# targeting .NET platform taking advantage of a vast collection of .NET libraries for implementing D³N primitives

• Future work:
  • Security issues are currently out of the scope of this paper. Executable code migrating from node to node
  • Validate and verify the correctness of the design by implementing a compiler targeting various mobile devices
  • Disclose code in public domain
3. Connectivity and Routing & How I Got into Social Nets #1

- Motivation and context
- Experiments
- Results
- Analysis of forwarding algorithms
- Consequences on mobile networking
Three independent experiments

- In Cambridge
  - Capture mobile users interaction.

- Traces from Wifi network:
  - Dartmouth and UCSD

<table>
<thead>
<tr>
<th>User Population</th>
<th>Intel</th>
<th>Cambridge</th>
<th>UCSD</th>
<th>Dartmouth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>iMote</td>
<td>iMote</td>
<td>PDA</td>
<td>Laptop/PDA</td>
</tr>
<tr>
<td>Network type</td>
<td>Bluetooth</td>
<td>Bluetooth</td>
<td>WiFi</td>
<td>WiFi</td>
</tr>
<tr>
<td>Duration (days)</td>
<td>3</td>
<td>5</td>
<td>77</td>
<td>114</td>
</tr>
<tr>
<td>Granularity (seconds)</td>
<td>120</td>
<td>120</td>
<td>20</td>
<td>300</td>
</tr>
<tr>
<td>Nodes participating</td>
<td>141</td>
<td>238</td>
<td>261</td>
<td>6648</td>
</tr>
<tr>
<td>Number of contacts</td>
<td>3,984</td>
<td>8,856</td>
<td>175,105</td>
<td>4,058,284</td>
</tr>
</tbody>
</table>
iMote data sets

• Easy to carry devices

• Scan other devices every 2mns
  – Unsync feature

• log data to flash memory for each contact
  – MAC address, start time, end time

• 2 experiments
  – 20 motes, 3 days, 3,984 contacts, IRC employee
  – 20 motes, 5 days, 8,856 contacts, CAM students
What an iMote looks like
What we measure

- For a given pairs of nodes:
  - contact times and inter-contact times.

Duration of the experiment

an inter-contact

a contact time
What we measure (cont’d)

- Distribution per event.
  ≠ seen at a random instant in time.

- Plot log-log distributions.

- We aggregate the data of different pairs.
  (see the following slides).
Example: a typical pair

\[ \alpha \]

cutoff
Examples: Other pairs
Aggregation (1): for one fixed node
Aggregation (2) : among iMotes

Tail Distribution Function of the inter-contact time.
Summary of observations

- Inter-contact time follows an approximate power-law shape in all experiments.
- $\alpha < 1$ most of the time (very heavily tailed).
- Variation of parameter with the time of day, or among pairs.
Problem

• Given that all data set exhibit approximate power law shape of the inter-contact time distribution:

  • Would a purely opportunistic point-to-point forwarding algorithm converge (i.e. guarantee bounded transmission delays) ?
  • Under what conditions ?
Forwarding algorithms

• Based on opportunities, and “Stateless”:
  • Decision does not depend on the nodes you meet.

• Between two extreme relaying strategies:
  • Wait-and-forward.
  • Flooding.

• Upper and Lower bounds on bandwidth:
  • Short contact time.
  • Full contact time (best case, treated here).
Two-hop relaying strategy

- Grossglauser & Tse (2001):
  - Maximizes capacity of dense ad-hoc networks.
  - Authors assume nodes location i.i.d. uniform.
Our assumptions on Mobility

• Homogeneity
  • Inter-contact for every pairs follows power law.
  • No cut-off bound.

• Independence
  • In “time”: contacts are renewal instants.
  • In “space”: pairs are independent.

\[ P[X \geq t] = t^\alpha \]
Two-hop: stability/instability

- $\alpha > 2$
  - The two hop relaying algorithm converges, and it achieves a finite expected delay.

- $\alpha < 2$
  - The expected delay grow to infinity with time.
Two-hop: extensions

- Power laws with cut-off:
  - Large expected delay.

- Short contact case:
  - By comparison, all the negative results hold.
  - Convergence for $\alpha > 3$ by Kingman's bound.
  - We believe the same result holds for $\alpha > 2$. 
The Impact of redundancy

• The Two-hop strategy is very conservative.
  • What about duplicate packet? Or epidemics forwarding?

• This comes to the question:

\[ D_1, D_2 \text{ independent, with same law,}
\]

if \( \mathbb{E}[D_1] = \mathbb{E}[D_2] = \infty \) what is \( \mathbb{E}[\min(D_1, D_2)] \)?
Forwarding with redundancy:

- **For** $\alpha > 2$
  
  Any stateless algorithm achieves a finite expected delay.

- **For** $\alpha < 1$
  
  No stateless algorithm (even flooding) achieve a bounded delay (Orey’s theorem).

- **For** $\alpha = \frac{m + 1}{m}$ and $\# \{ \text{nodes} \} \geq 2m$
  
  There exist a forwarding algorithm with $m$ copies and a finite expected delay.
Forwarding w. redundancy (cont’d)

• Further extensions:
  • The short contact case is open for $1 < \alpha < 2$.
  • Can we weaken the assumption of independence between pairs?
Consequences on mobile networking

- Mobility models need to be redesigned
  - *Exponential decay of inter contact is wrong.*
  - *Mechanisms tested with that model need to be analyzed with new mobility assumptions.*

- Stateless forwarding does not work
  - *Can we benefit from heterogeneity to forward by communities?*
  - *Scheme for peer-to-peer information sharing.*
Give it to me, I have 1G bytes phone flash.

I have 100M bytes of data, who can carry for me?

I can also carry for you!

Thank you but you are in the opposite direction!

Don’t give to me! I am running out of storage.

Reach an access point.

Internet

Finally, it arrive…

Search La Bonheme.mp3 for me

There is one in my pocket…

Search La Bonheme.mp3 for me

Search La Bonheme.mp3 for me

Search La Bonheme.mp3 for me

3b Connectivity & Routing Ever More Social
K-clique Communities in Cambridge Dataset
K-clique Communities in Infocom06 Dataset

Lausanne Group

Barcelona Group

Paris Groups

Paris Group A

Paris Group B

K=4
Human Hubs: Popularity
Forwarding Scheme Design Space

Explicit Social Structure

Label

Bubble

Human Dimension

Structure in Cohesive Group

Clique Label

Network Plane

Rank, Degree

Structure in Degree
Use affiliation+hubs to fwd inter+intra cliques
Give it to me, I have 1G bytes phone flash.

I have 100M bytes of data, who can carry for me?

I can also carry for you!

Thank you but you are in the opposite direction!

Don't give to me! I am running out of storage.

Reach an access point.

Internet

Finally, it arrive...

Search La Bonheme.mp3 for me

There is one in my pocket...

3c Connectivity & Routing 3 - Community Detection
Community improves forwarding

- Identifying communities (e.g. affiliations) improves forwarding efficiency. [label]
- Evaluate on Infocom06 data.
Centralized Community Detection

K-clique Detection [Palla04]
Weighted Network Analysis [Newman05]
Betweenness [Newman04]
Modularity [Newman06]
Information theory [Rosvall06]
Statistical mechanics [Reichardt]
Survey Papers [Danon05] [Newman04]
K-clique Detection

Union of k-cliques reachable through a series of adjacent k-cliques

Adjacent k-cliques share k-1 nodes

Members in a community reachable through well-connected well subsets

Examples
  • 2-clique (connected components)
  • 3-clique (overlapping triangles)

Overlapping feature

Percolation threshold

\[ p_c (k) = \frac{1}{[(k-1)N]^{1/(k-1)}} \]
K-clique Communities in Infocom06 Dataset

Barcelona Group

Paris Groups

Lausanne Group

- Barcelona Group
- Paris Group A
- Paris Group B
- Lausanne Group

K=3
K-clique Communities in Infocom06 Dataset

Lausanne Group

Paris Groups

Barcelona Group

K=4
K-clique Communities in Infocom06 Dataset

- Italian
- Paris Group A (French)
- Paris Group B (French)
- Barcelona Group (Spanish)

K=5
Calculate the unweighted edge betweenness.

Divide each calculated betweenness value by its weight.

Remove the edge with the highest edge betweenness and repeat from 1 until there are no more edges in the network.

Recalculate the modularity value of the network with the current community partitioning. Select those splitting with local maxima of modularity.
### Community Detection using WNA

<table>
<thead>
<tr>
<th>Experimental dataset</th>
<th>Infocom06</th>
<th>Cambridge</th>
<th>Reality</th>
<th>Infocom05</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{max}$</td>
<td>0.2280</td>
<td>0.4227</td>
<td>0.5682</td>
<td>0.3039</td>
</tr>
<tr>
<td>Max. Community Size</td>
<td>13</td>
<td>18</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>No. Communities</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Avg. Community Size</td>
<td>8.000</td>
<td>16.500</td>
<td>9.875</td>
<td>6.5</td>
</tr>
<tr>
<td>No. Community Nodes</td>
<td>32</td>
<td>33</td>
<td>73</td>
<td>26</td>
</tr>
<tr>
<td>Total No. of Nodes</td>
<td>78</td>
<td>36</td>
<td>97</td>
<td>37</td>
</tr>
</tbody>
</table>
Distributed Community Detection

**SIMPLE, K-CLIQUE, MODULARITY**

Terminology: Familiar Set ($F$), Local Community ($C$)

Update and exchange local information during encounter

Build up Familiar Set and Local Community

- $CommunityAccept()$, $MergeCommunities()$
We want to see whether $v_i$ should be added to $C_0$.

So we first count the number of vertices in $v_i$'s familiar set:

$$\left( (C_i \cap C_0) + C_0 \right)$$

Then we count the number of vertices in both $v_i$'s familiar set and also in the local community of $v_0 = \emptyset$.

And we admit $v_0$ to $C_0$ iff

$$\emptyset > (C_i \cap C_0) \times \lambda$$

We only consider merging the two communities $C_0$ and $C_i$ if the fraction of them in common $\gamma > \gamma$.

---

**MergeCommunities** ($C_0$, $C_i$)

**CommunityAccept** ($v_i$)
CommunityAccept \( (v_i) \):

\[ |F_i \cap C_0| \geq k - 1 \]

\text{MergeCommunities(} C_o, C_i): \hspace{1cm} \[ |\tilde{F}_j \cap C_0| \geq k - 1 \]

\begin{itemize}
  \item a) We want to see whether \( v_i \) should be added to \( C_0 \)
  \item b) So we count the number of vertices in both \( v_i \)'s familiar set and also in the local community of \( v_0 = () \)
  \item c) And we admit \( v_0 \) to \( C_0 \) iff \( () \geq k \)
\end{itemize}
Boundary Set

\[ B_0 = \{ v_i \mid (v_i \in C_0) \text{ and } ((F_i \setminus C_0) \neq \emptyset) \} \]

Local Modularity

Measure of the sharpness of local community

\[ R_0 = \frac{I}{|T|} \]
**CommunityAccept** $(v_i)$:

$$(F_i \subseteq C_0 \text{ and } B_0 \neq \emptyset) \text{ or } \Delta R_i^0 > 0$$

**MergeCommunities** $(C_o, C_i)$: for each $v_k$ in set $K$,

$$\tilde{F}_k \subseteq C_0 \text{ or } \Delta R_k^0 > 0$$
## Results and Evaluations

<table>
<thead>
<tr>
<th>Data Set</th>
<th>SIMPLE</th>
<th>K-CLIQUE</th>
<th>MODULARITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reality</td>
<td>0.79/0.81</td>
<td>0.87</td>
<td>0.89</td>
</tr>
<tr>
<td>UCSD</td>
<td>0.47/0.56</td>
<td>0.55</td>
<td>0.65</td>
</tr>
<tr>
<td>Cambridge</td>
<td>0.85/0.85</td>
<td>0.85</td>
<td>0.87</td>
</tr>
<tr>
<td>Complexity</td>
<td>O(n)</td>
<td>O(n^2)</td>
<td>O(n^4)/O(n^2k^2)</td>
</tr>
</tbody>
</table>

Newman weighted analysis

Palla et al, k-Clique

\[
\sigma_{Jaccard} = \frac{|\Gamma_i \cap \Gamma_j|}{|\Gamma_i \cup \Gamma_j|}
\]
Results and Evaluations

Distributions of Local Community Views
Evolution of communities

More general Familiar Set threshold (e.g. hours per day)

Detection of different categories of relationship by specifying contact duration and number of contacts

Dynamic selection of Familiar Set threshold (e.g. fuzzy logic)

Aging effect

Temporal communities

Evaluation on more data sets (e.g. Dartmouth WiFi, iMote experiments)
The End

• With much thanks & acknowledgements to
• James Scott, Ebon Upton, Menghow Lim, Pan Hui
• Eiko Yoneki, Ioannis Baltopoulos, Shu-yan Chan
• Jing Su, Ashvin Goyal, Eyal de Lara
• Christophe Diot, Augustin Chaintreau, Richard Gass