Opportunity is the Mother of Invention

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

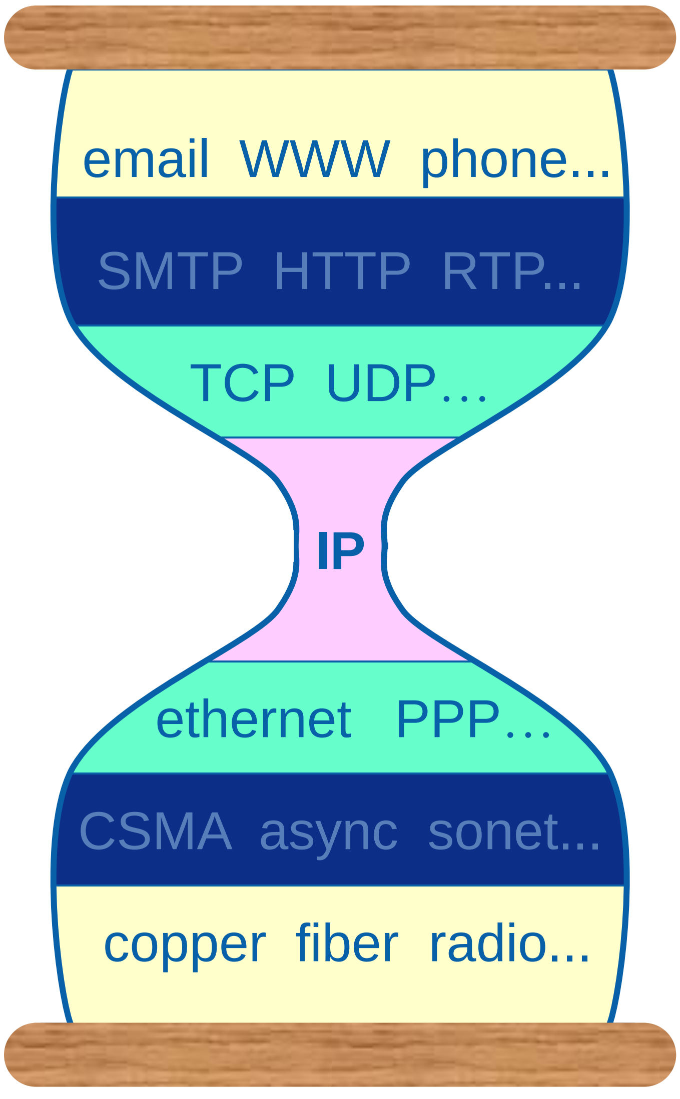
Jon Crowcroft

Jon.crowcroft@cl.cam.ac.uk

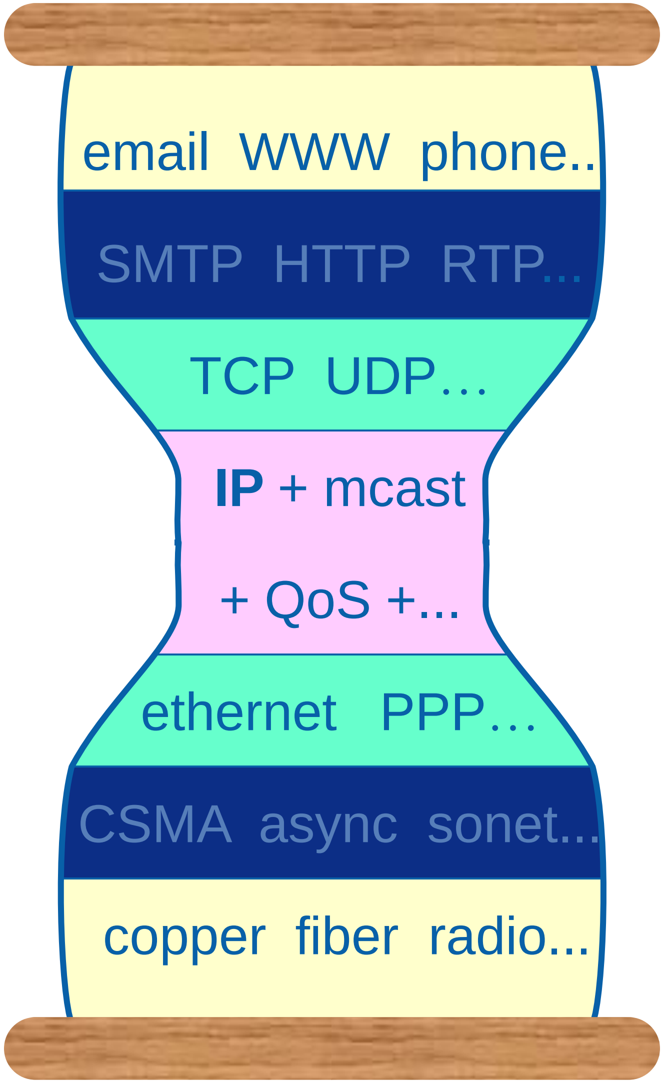
Outline Narrative History of Haggle

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)

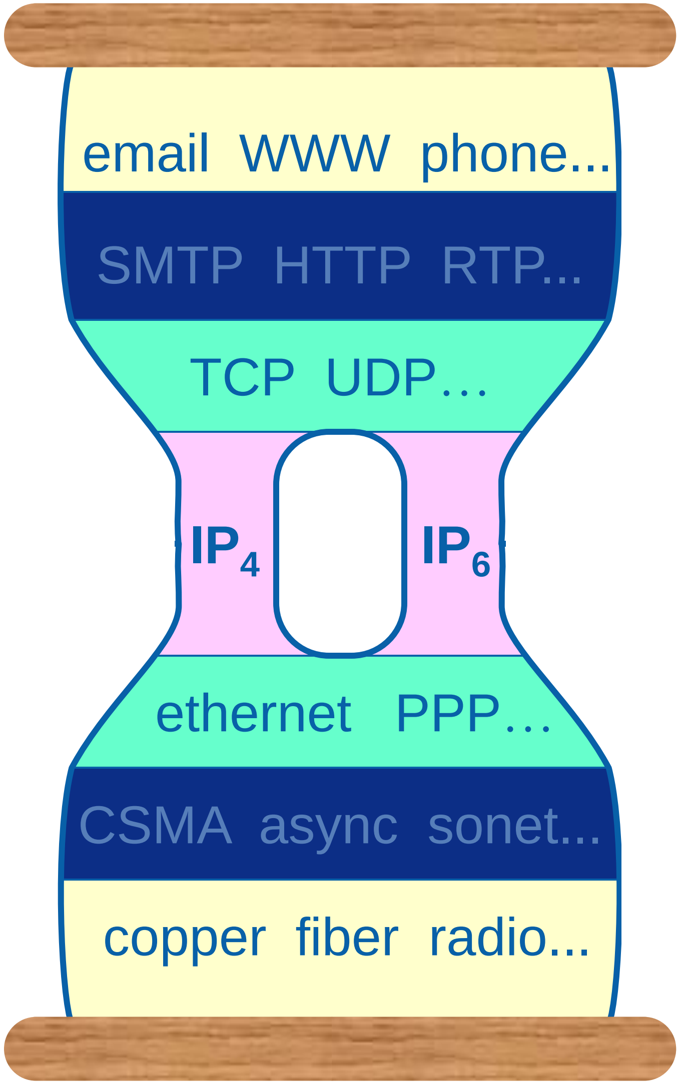


Putting on Weight

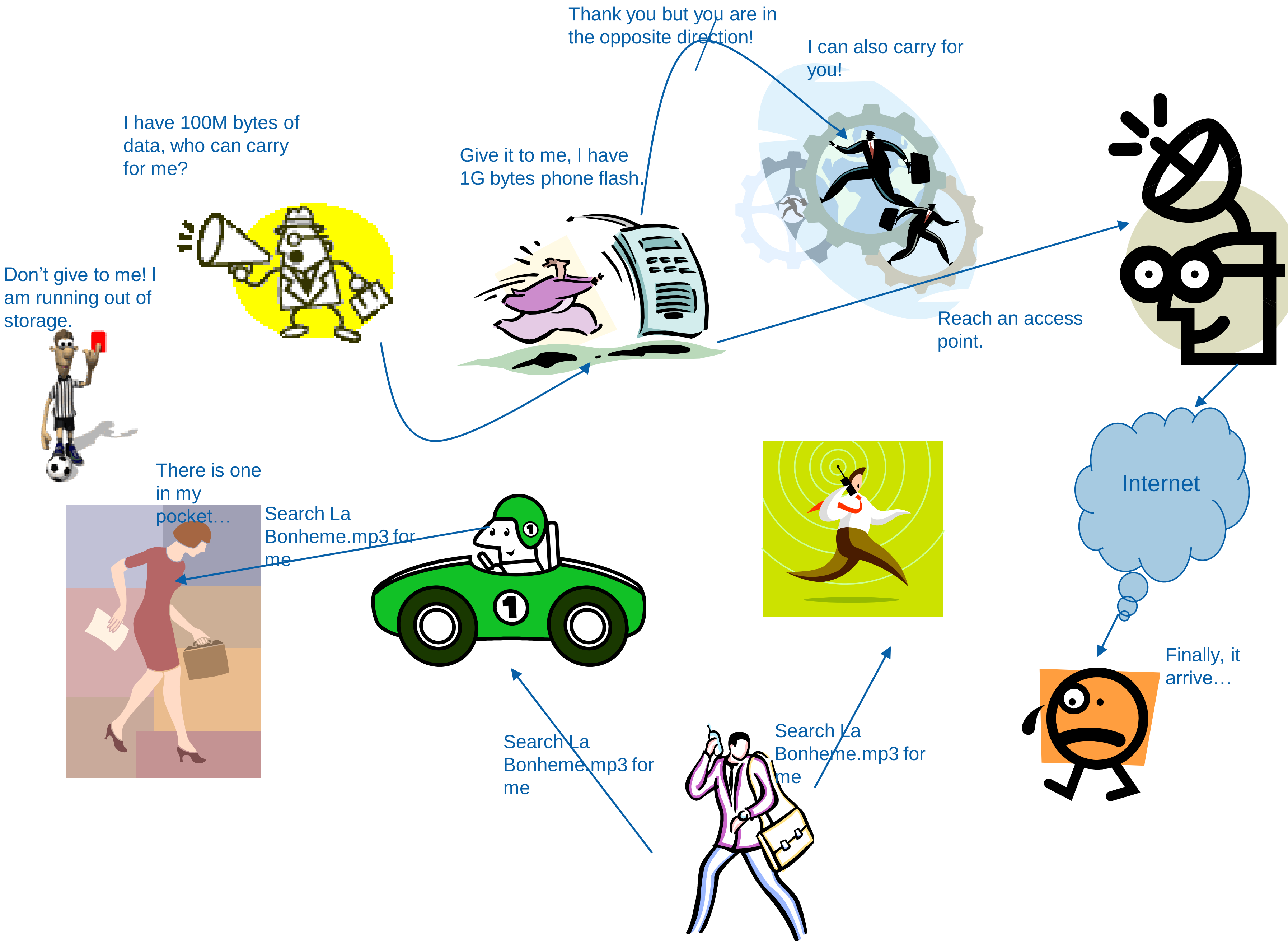


- requires more functionality from underlying networks

Mid-Life Crisis



- doubles number of service interfaces
- requires changes above & below
- major interoperability issues



I have 100M bytes of data, who can carry for me?
Don't give to me! I am running out of storage.
Give it to me, I have 1G bytes phone flash.
Thank you but you are in the opposite direction!
I can also carry for you!
Reach an access point.
Internet
Finally, it arrive…
There is one in my pocket…
Search La Bonheme.mp3 for me
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

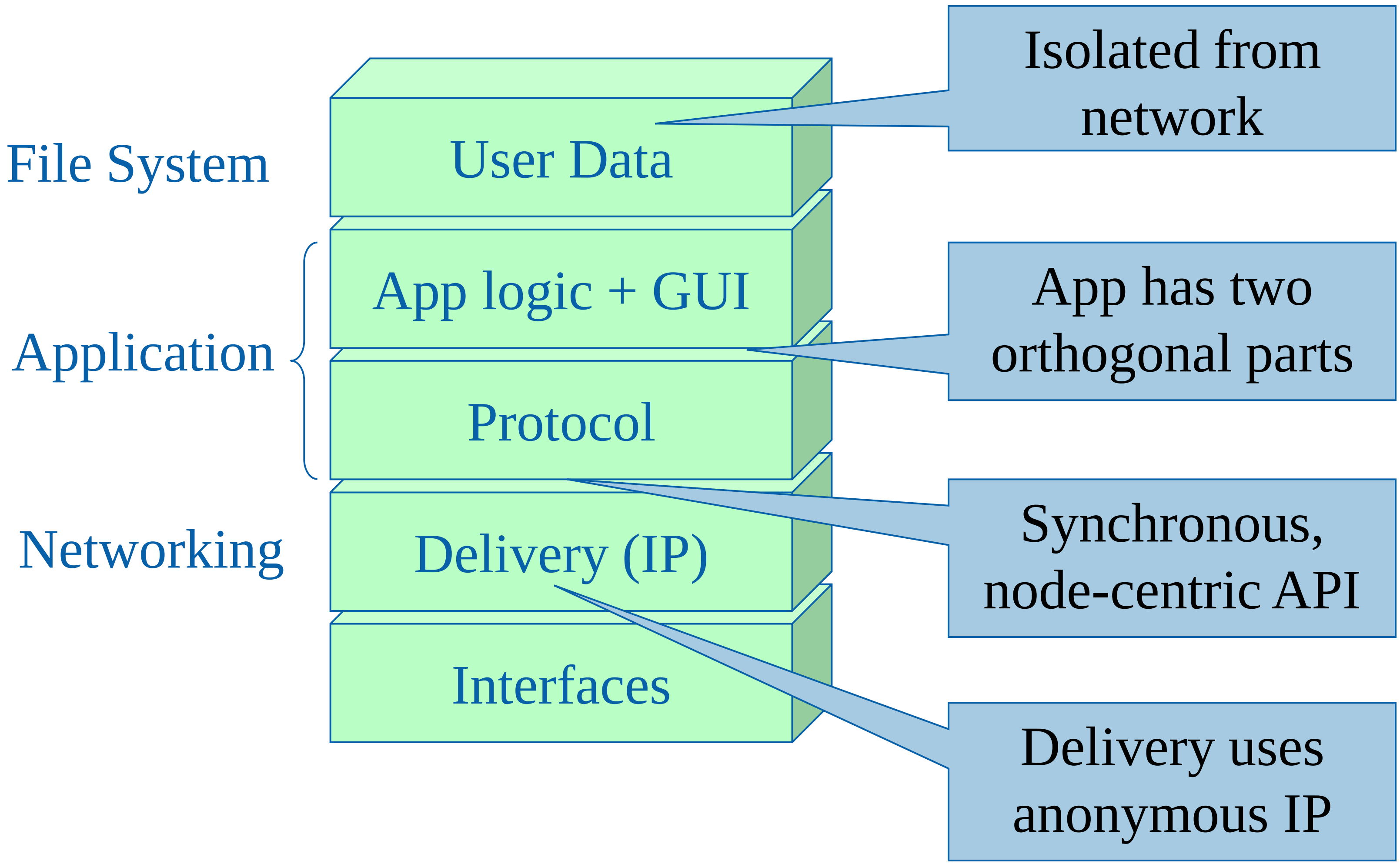
Phase transitions and networks

- 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”

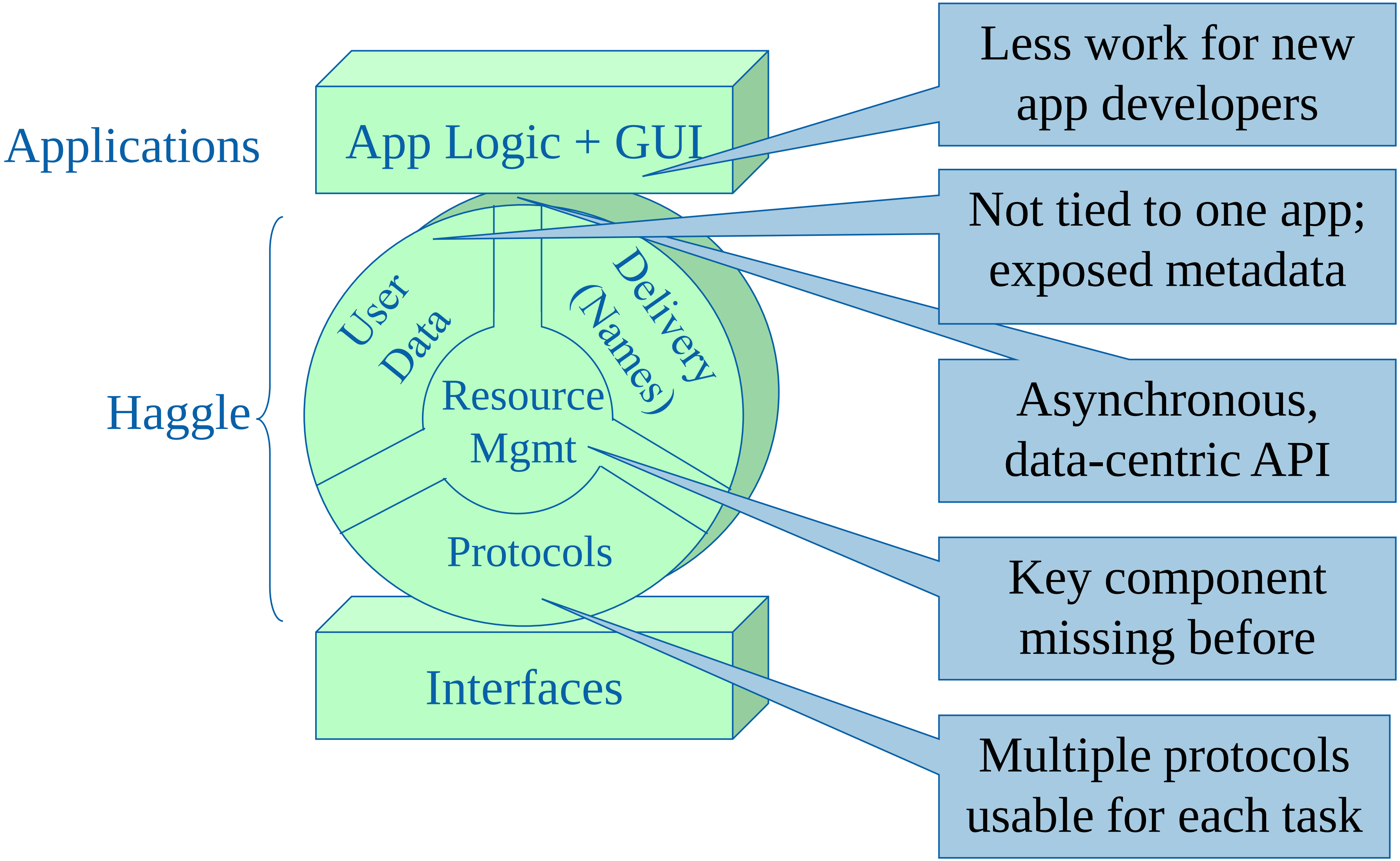
Decentralisation&Disconnectivity

- Absence of infrastructure for
  - Routing, searching, indexing
  - Names, Identity, Currency
- When everything's adhoc, even pagerank has to be
  - Hence "Ad Hoc Google" -> "Haggle" Intel Cam 2004.
  - 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



Haggle framework design



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

Message

DO-Type	Data
Content-Type	message/rfc822
From	James Scott
To	Richard Gass
Subject	Check this photo out!
Body	[text]

Attachment

DO-Type	Data
Content-Type	image/jpeg
Keywords	Sunset, London
Creation time	05/06/06 2015 GMT
Data	[binary]

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

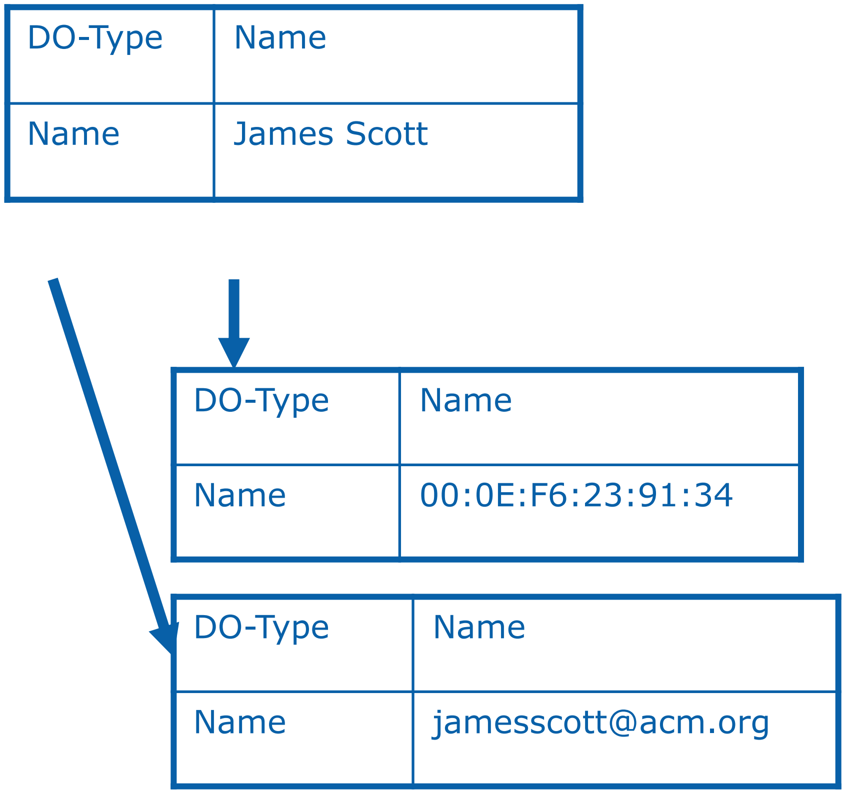
	One-Off	*Persistent*
Local	“Desktop” Search (find mp3s with artist “U2”)	Listen (wants to receive webpages)
Remote	“Web” Search (find “london restaurants”)	Subscribe (send all photos created by user X to X’s PC)

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)

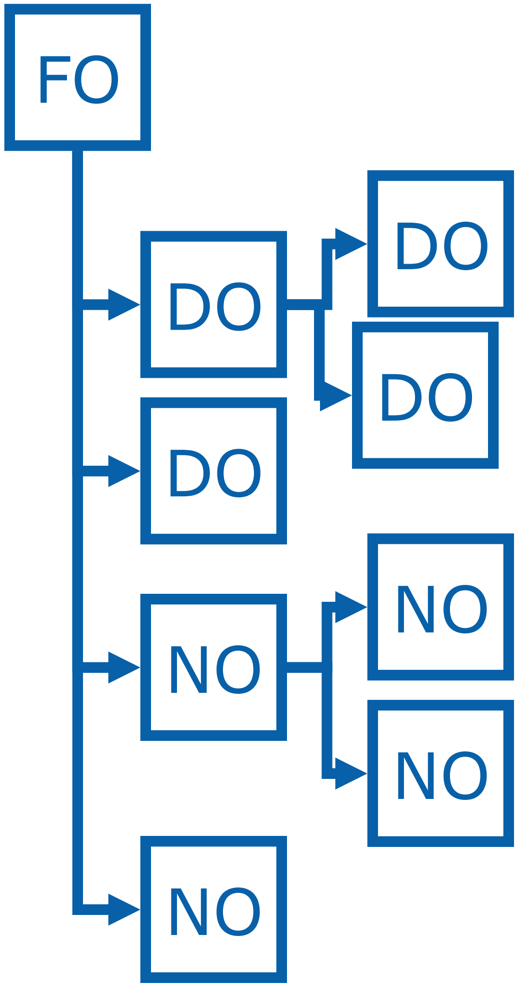
Name Graphs comprised of Name Objects

- 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



Forwarding Objects

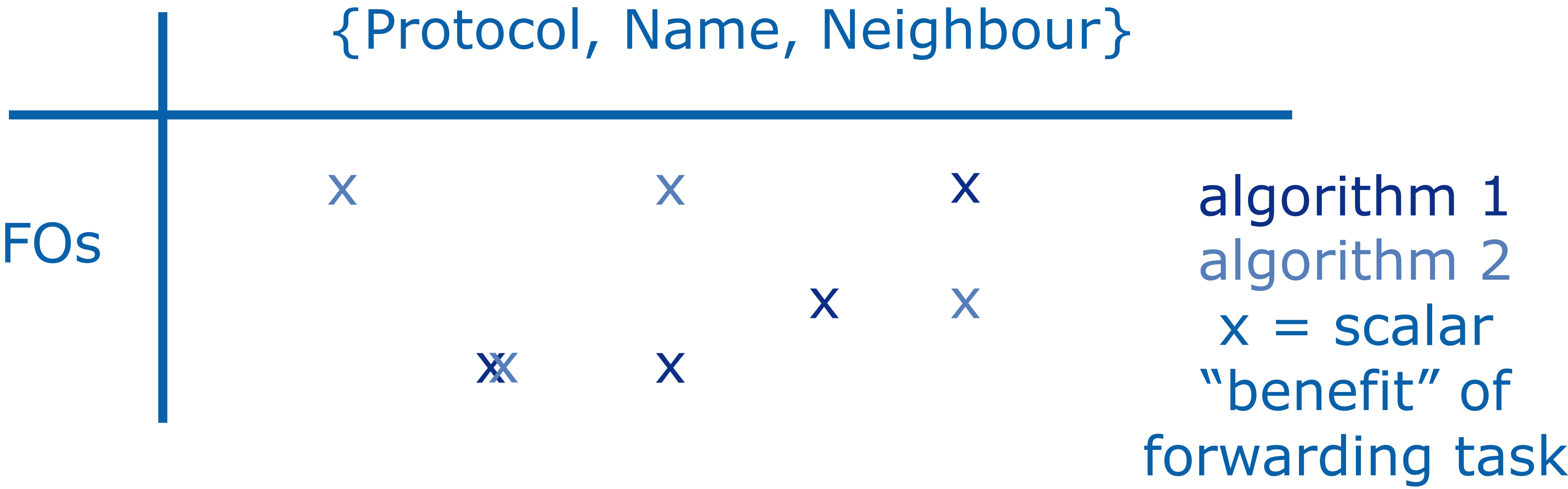
- Special class of DO used for storing metadata about forwarding
  - TTL,expiry, etc
- Since full structure of naming and data is sent, “intermediate” nodes are empowered to:
  - Use data as they see fit
  - Use up-to-date state and whole name graph to make best forwarding decision



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
- Can also create Tasks to perform signalling
- Many forwarding algs can run *simultaneously*

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…

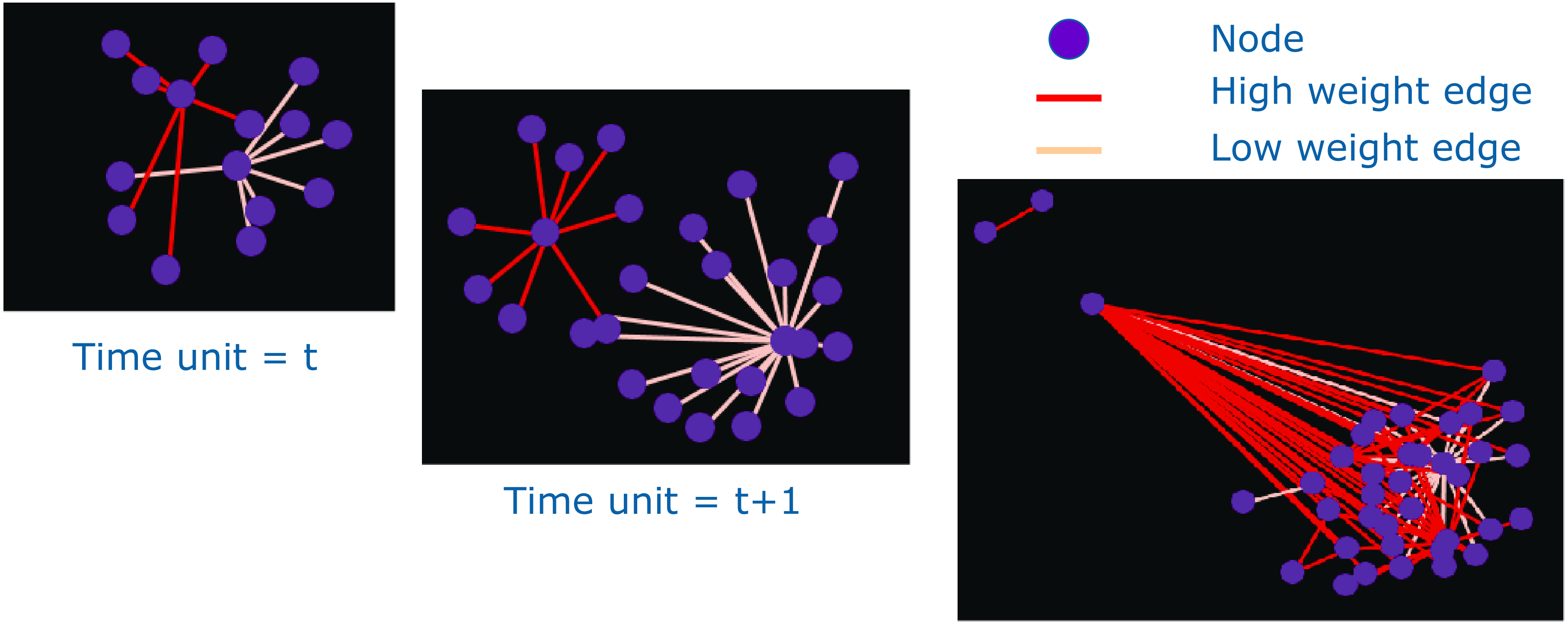
D^{3}N*

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

* **D**ata **D**riven **D**eclarative **N**etworking

PSN: Dynamic Human Networks

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



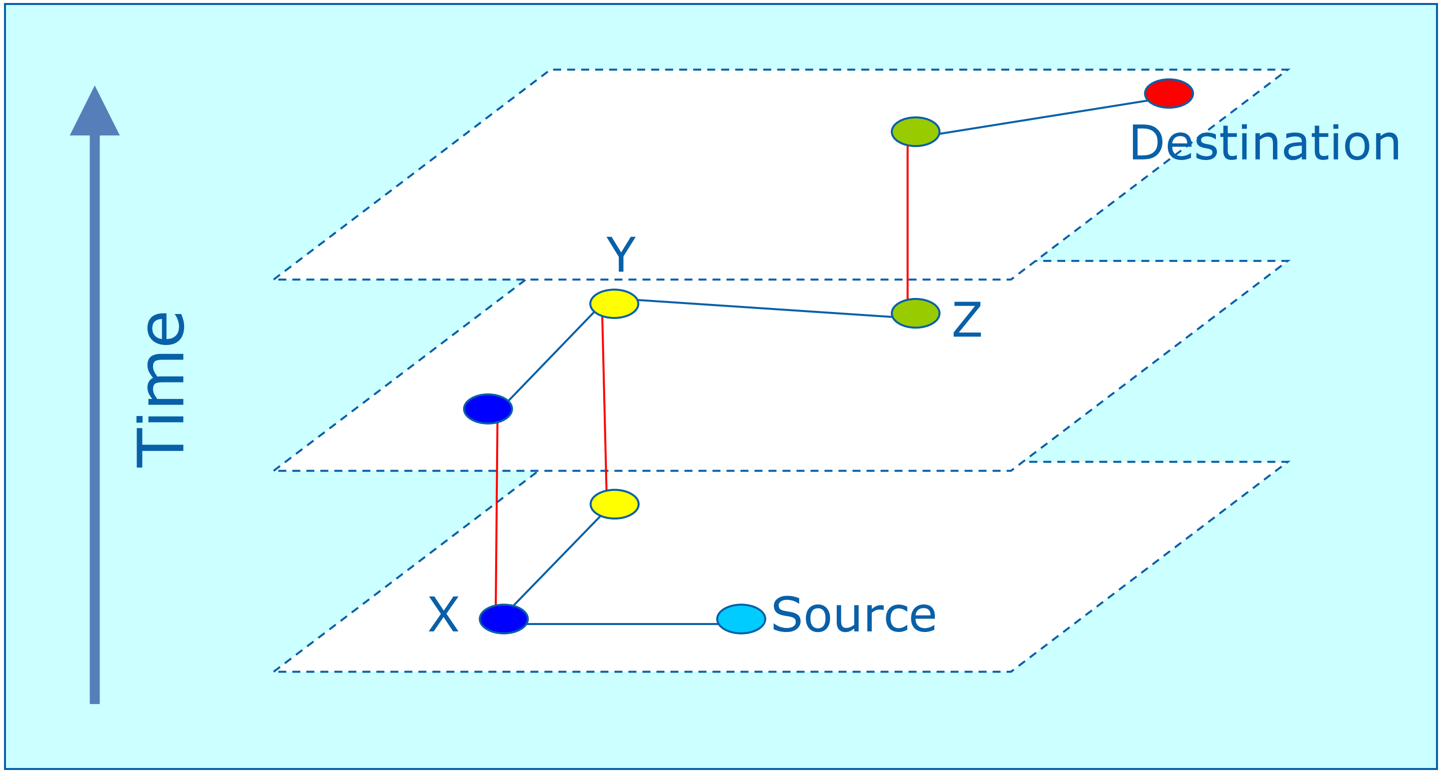
Time unit = t

Time unit = t+1

Time unit = t+2

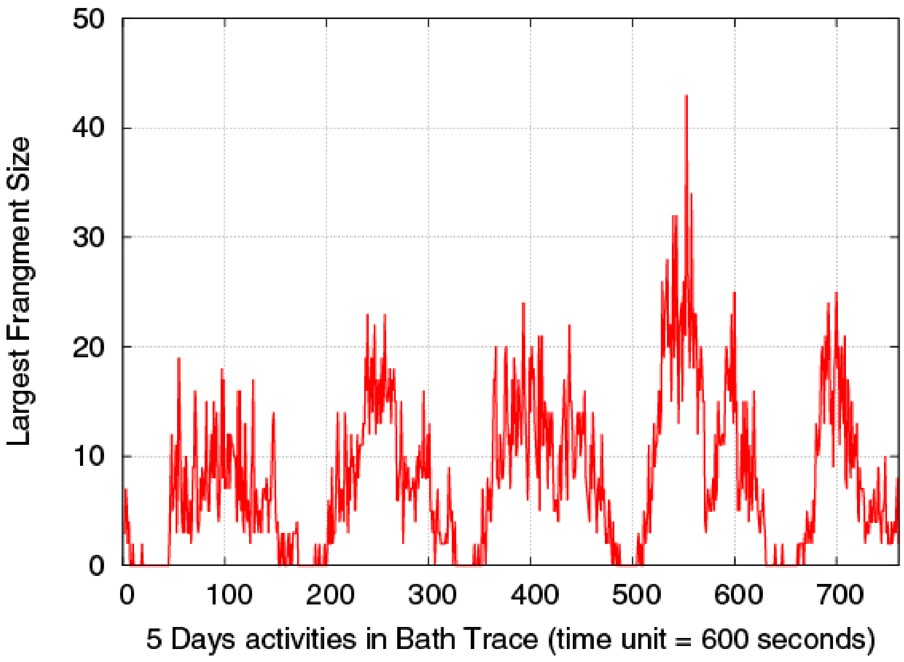
Time Dependent Networks

- Data paths may not exist at any one point in time but do exist *over time*
- Delay Tolerant Communication

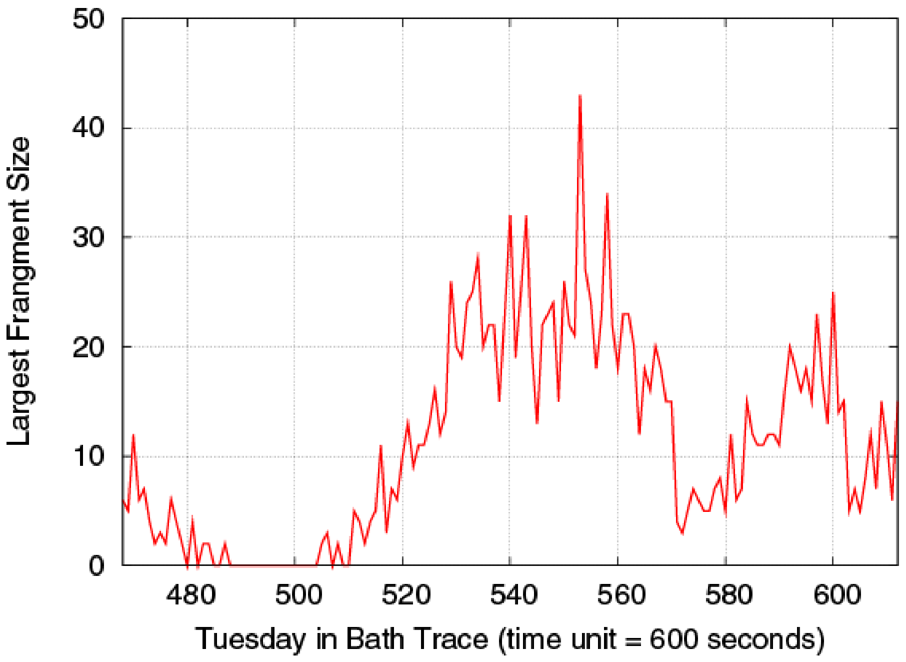


Regularity of Network Activity

- Size of largest fragment shows network dynamics



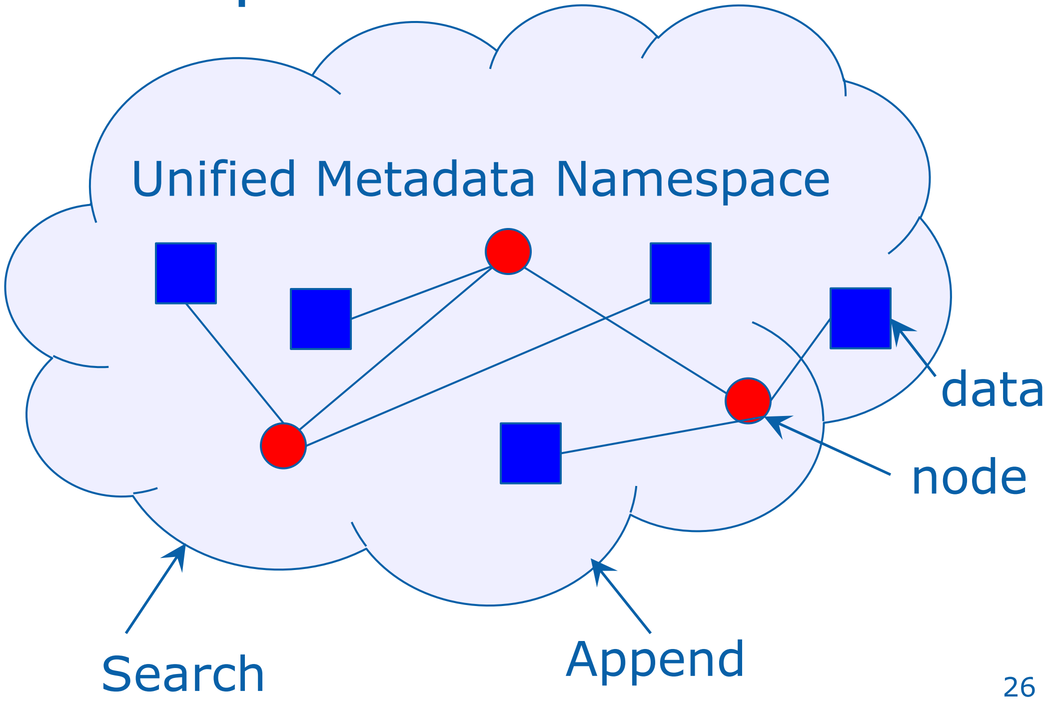
5 Days



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

D^3N 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^3N 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

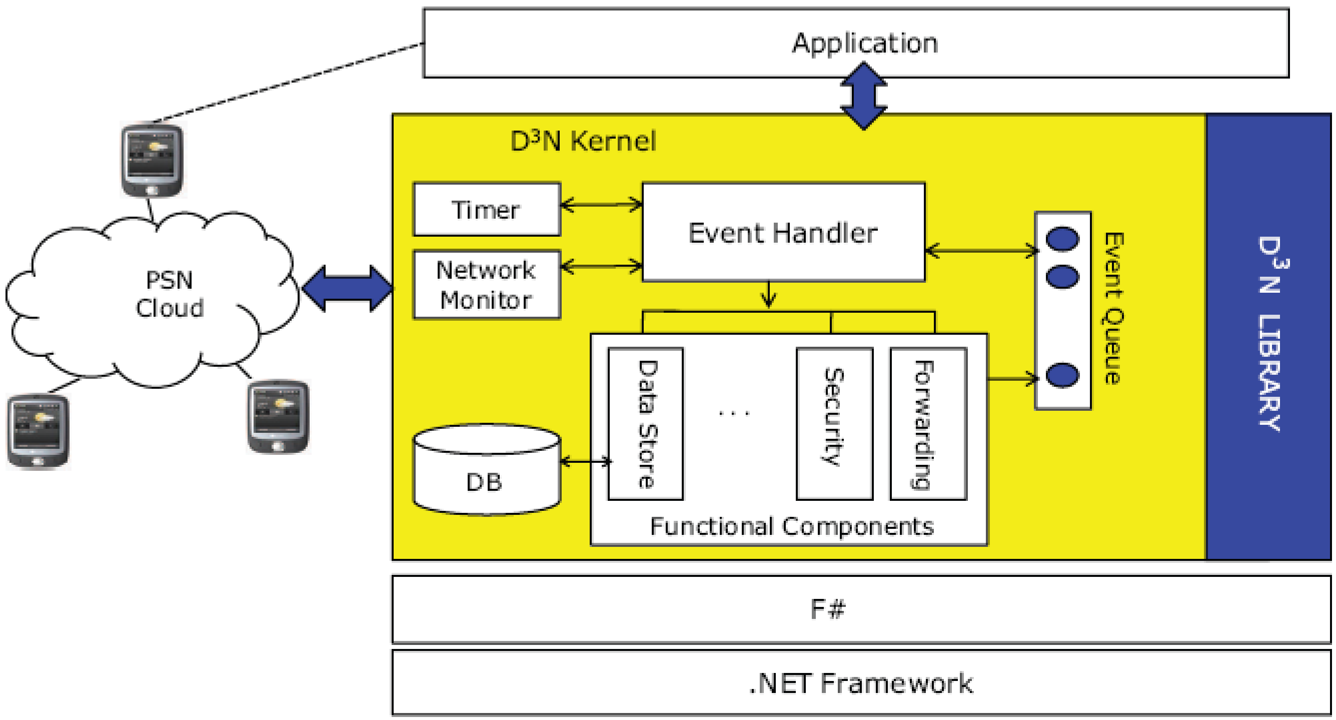
  (of course message passing and threads → encode states)
- Strongly typed
  - Static assurance that the program does not 'go wrong' at runtime unlike script languages
- Type inference
  - Types are not declared explicitly, hence programs are less verbose

D^3N 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 D^{3}N Architecture

- Each node is responsible for storing, indexing, searching, and delivering data
- Primitive functions associated with core D^{3}N calculus syntax are part of the runtime system
- **Prototype on MS Mobile .NET**



D^3N 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)

D^{3}N 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

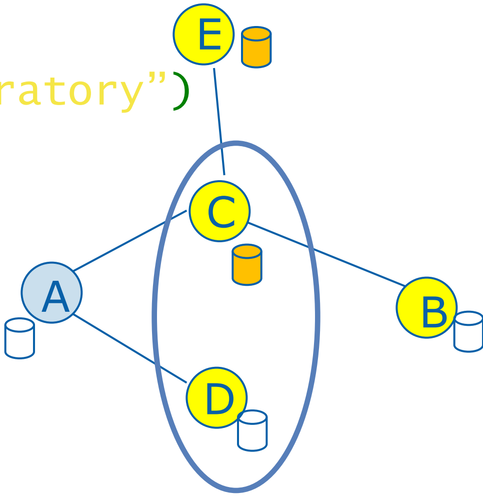
Example: Query to Networks

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

D³N:
```
select name from poll() where institute = “Computer Laboratory”
```

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

Message:
```
(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)

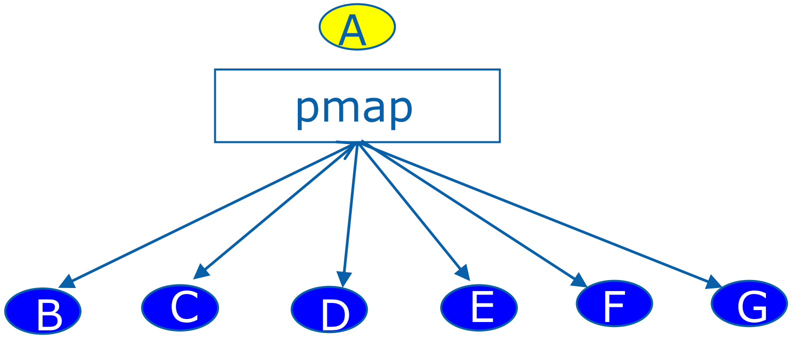
- Inner working
  - It sends the code to execute on the remote node
  - It blocks waiting for a response waiting from the node
  - Continues mapping the function to the rest of the nodes in a sequential fashion
  - An unavailable node blocks the entire computation

```
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

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



```
Event.register( Event.OnEncounter, fun d:device ->
  if d.nID = “B” && distance(self,d) < 3 then
    dispatch NodeEncountered(d);
)
```

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

```
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 D^3N 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

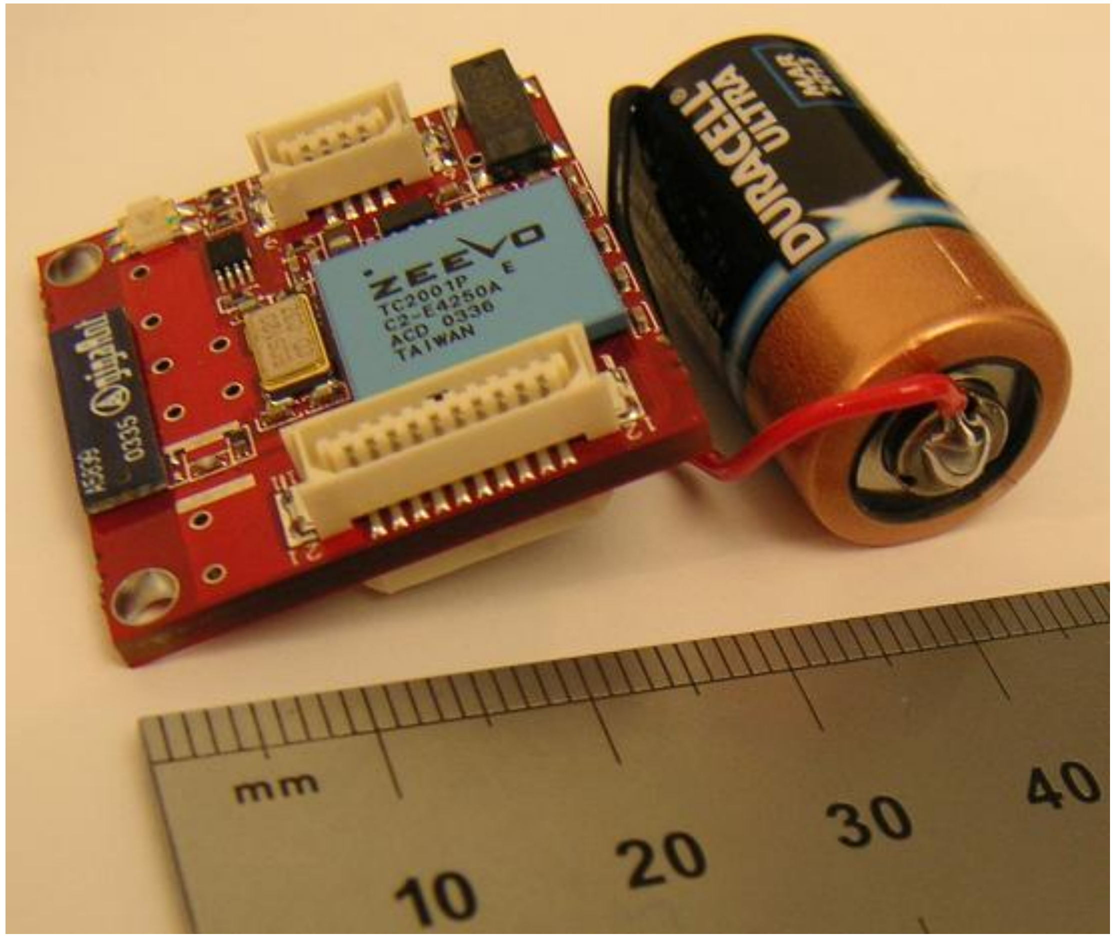
- In Cambridge
  - Capture mobile users interaction.
- Traces from Wifi network :
  - Dartmouth and UCSD

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

iMote data sets

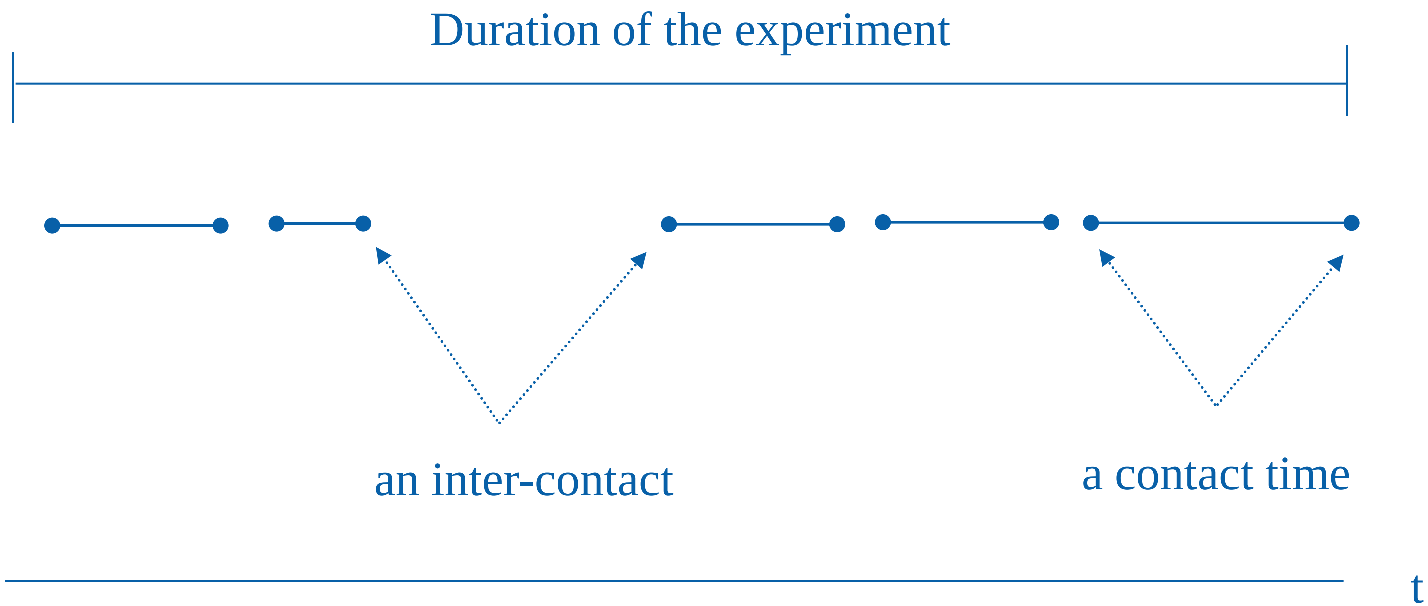
- Easy to carry devices
- Scan other devices every 2mns
  - Unsync feature
- log data to flash memory for each contact
  - MAC address, start time, end time
- 2 experiments
  - 20 motes, 3 days, 3,984 contacts, IRC employee
  - 20 motes, 5 days, 8,856 contacts, CAM students

What an iMote looks like



What we measure

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



What we measure (cont'd)

- Distribution per event.

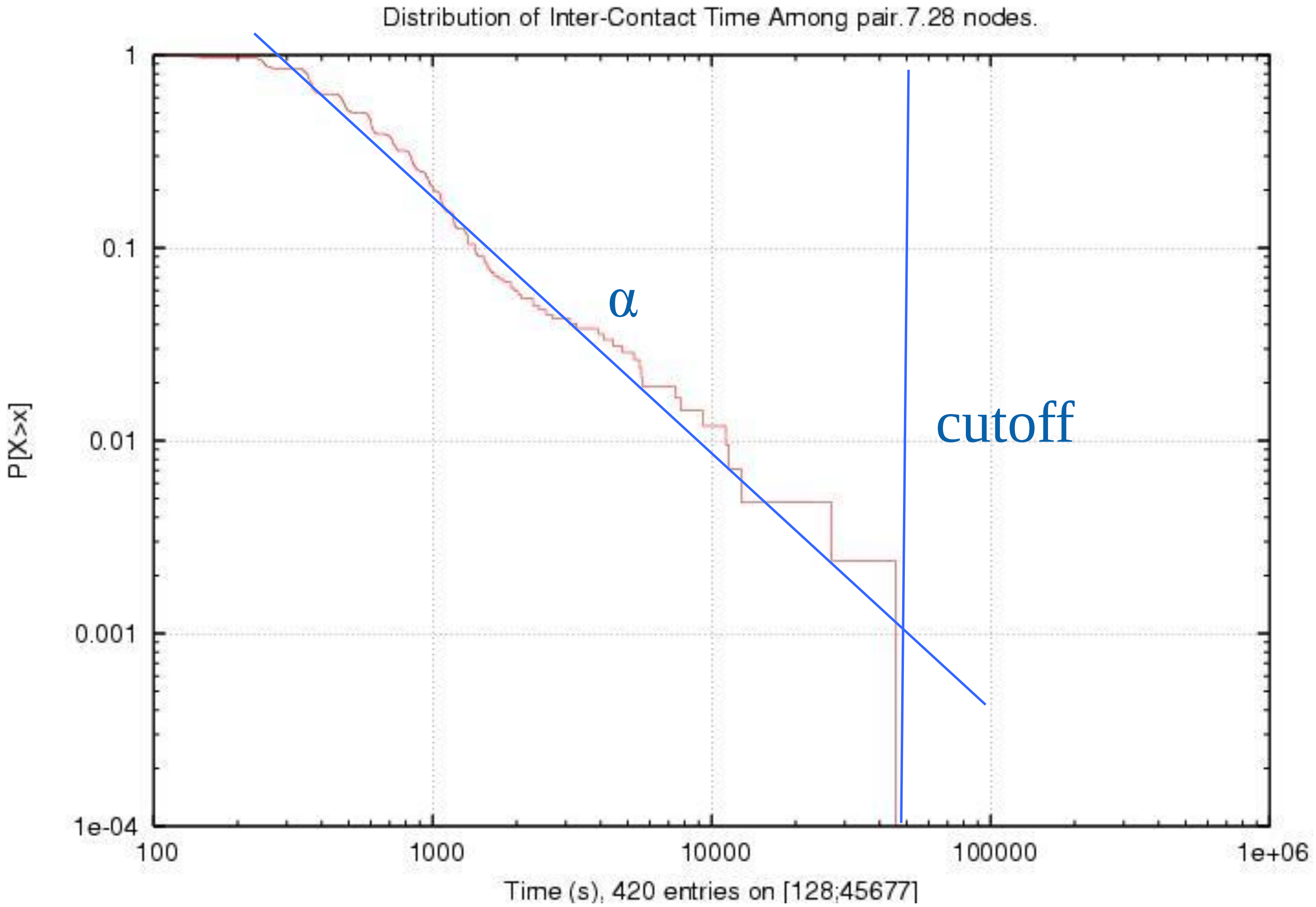
  ≠ seen at a random instant in time.

- Plot log-log distributions.

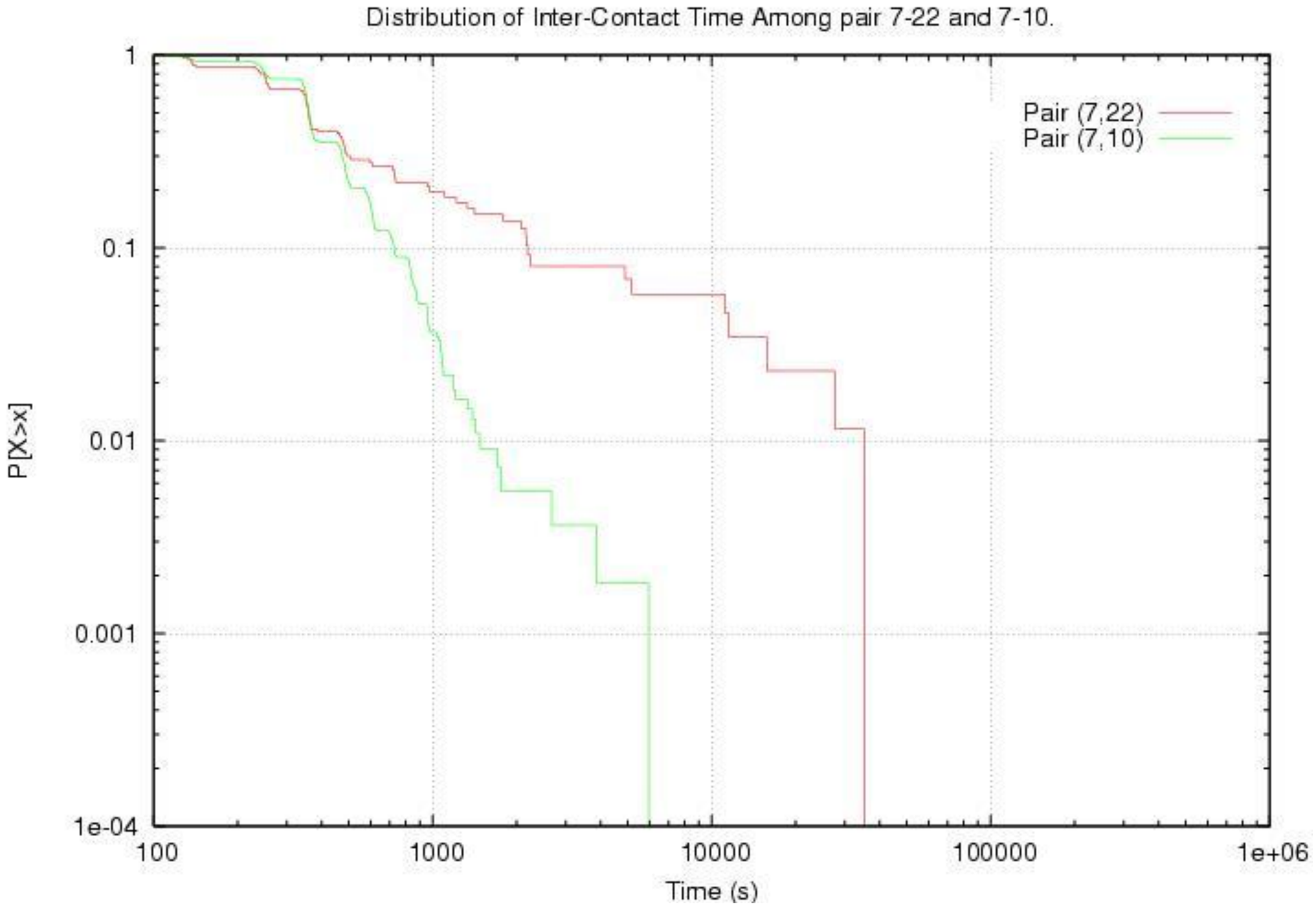
- We aggregate the data of different pairs.

  (see the following slides).

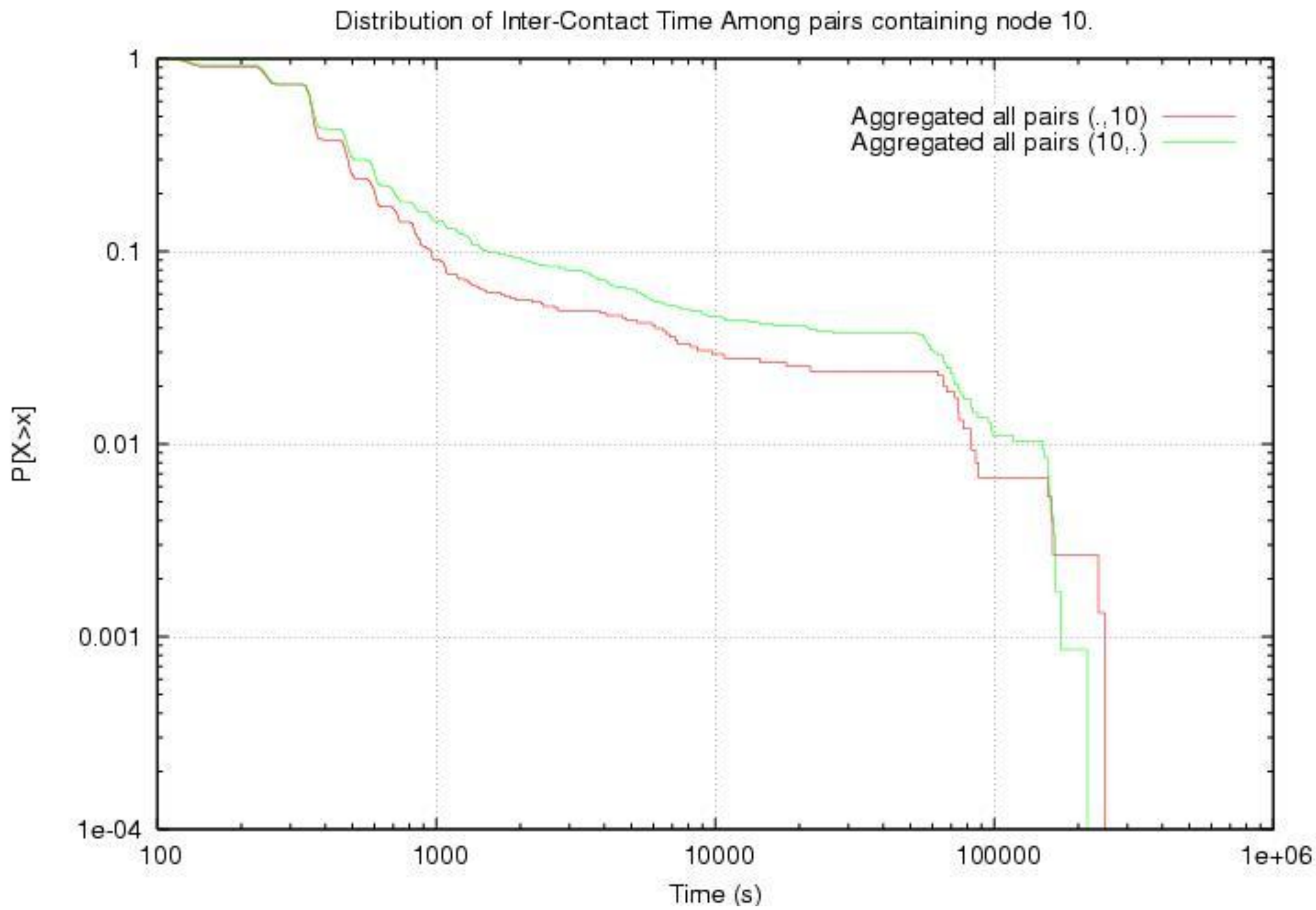
Example: a typical pair



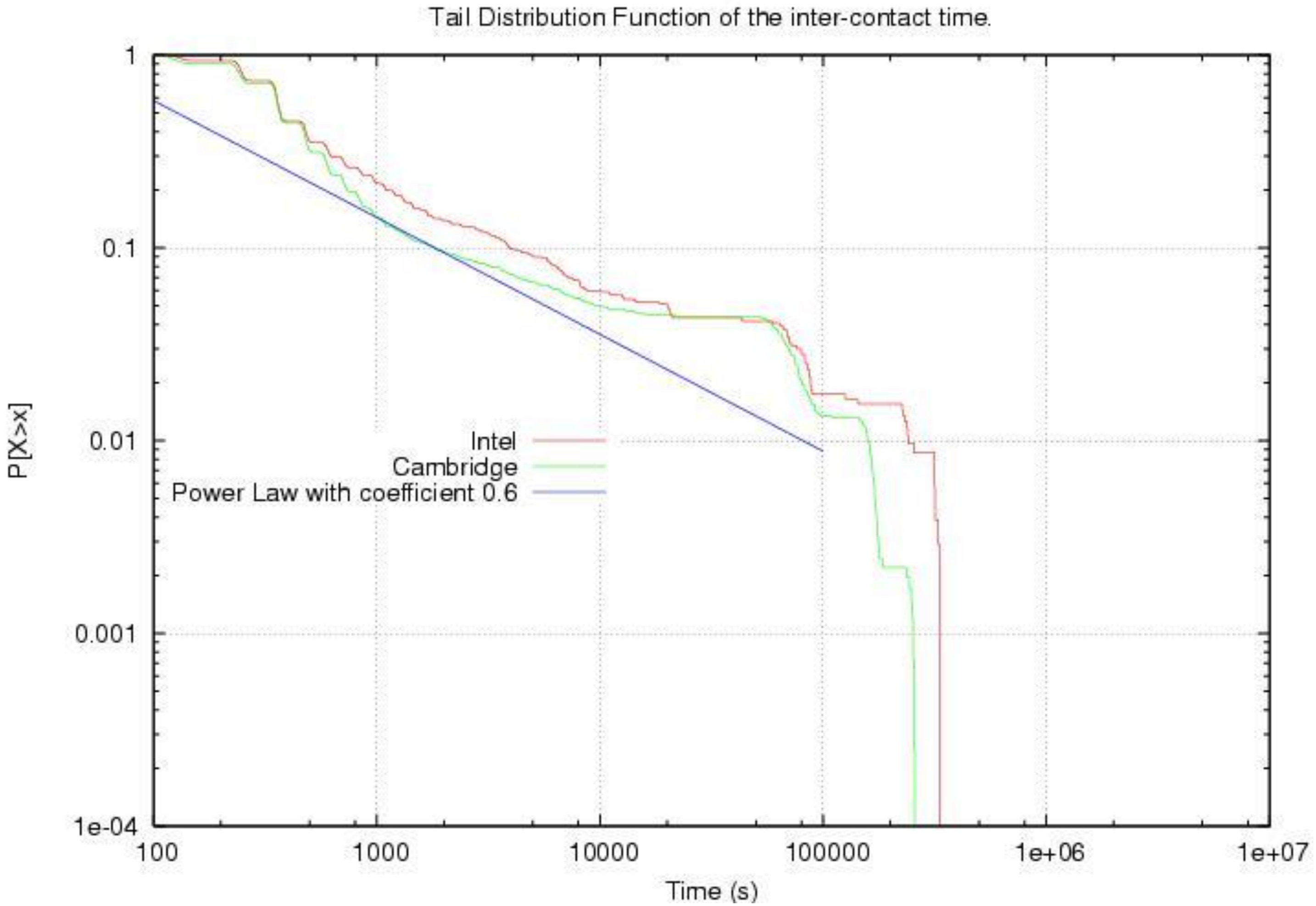
Examples : Other pairs



Aggregation (1): for one fixed



Aggregation (2) : among iMotes



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:

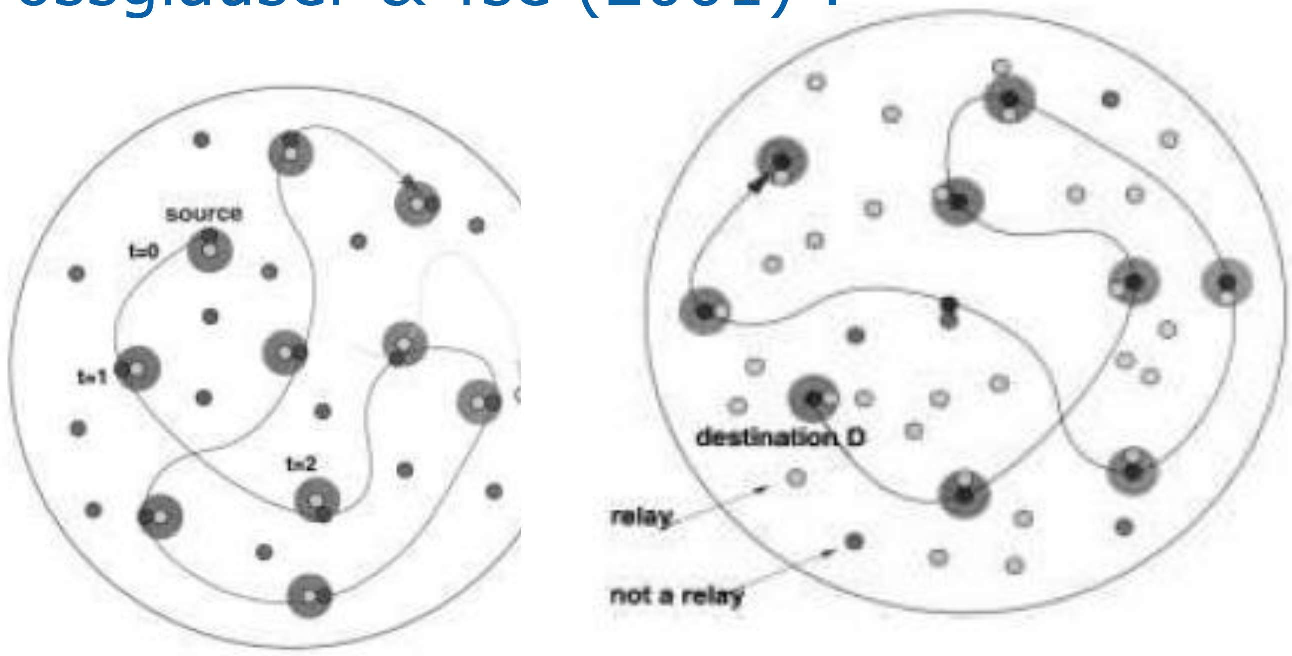
  - Would a purely opportunistic point-to-point forwarding algorithm converge (i.e. guarantee bounded transmission delays) ?
  - Under what conditions ?

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.

$$P\left[X \geq t\right] = t^{\alpha}$$

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

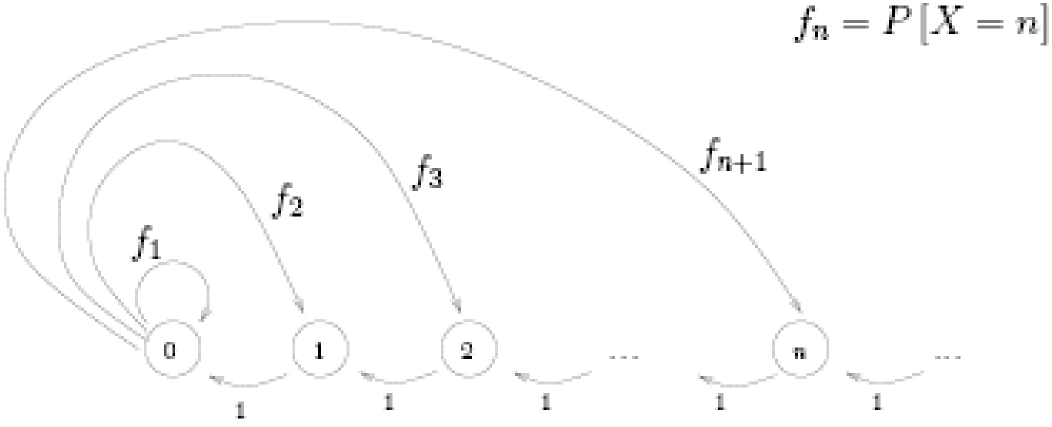
Two-hop: stability/instability

- $\alpha > 2$

  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:

$$D_1, D_2 \text{ independent, with same law },$$
$$\text{if } \mathbb{E}[D_1] = \mathbb{E}[D_2] = \infty \text{ what is } \mathbb{E}[\min(D_1, D_2)]?$$

Forwarding with redundancy:

- For $\alpha > 2$

  Any stateless algorithm achieves a finite expected delay.

- For $\alpha \geq \frac{m+1}{m}$ and $\#\{\text{ nodes }\} \geq 2m$:

  There exist a forwarding algorithm with m copies and a finite expected delay.

- For $\alpha < 1$

  No stateless algorithm (even flooding) achieve a bounded delay (Orey's theorem).

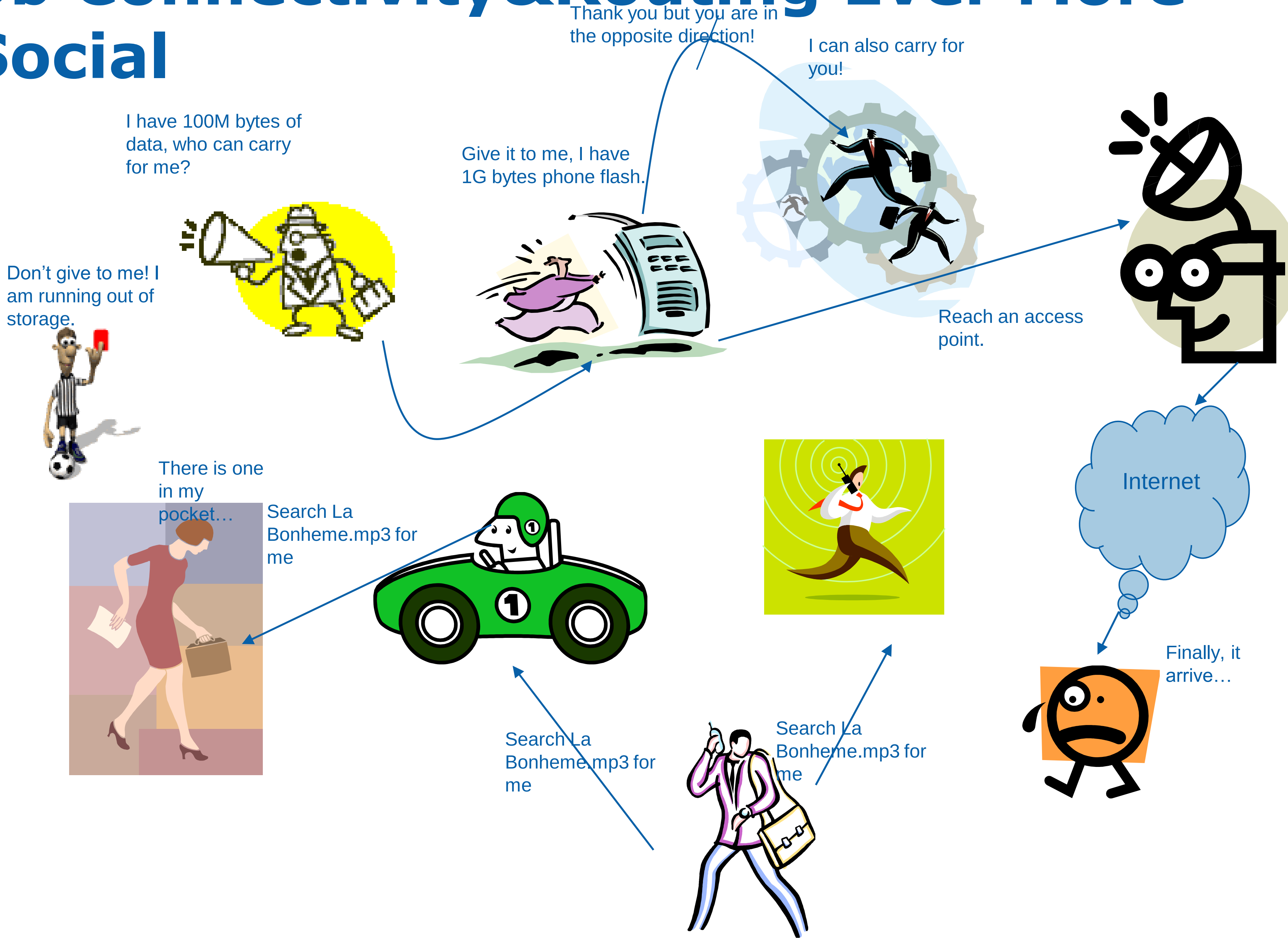
Forwarding w. redundancy (cont'd)

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

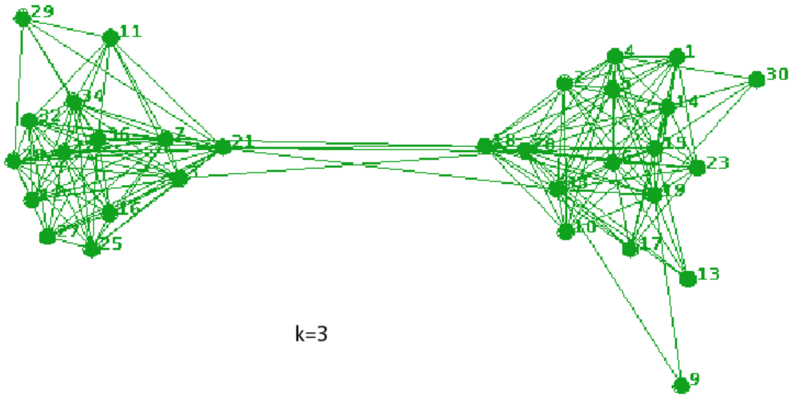
Consequences on mobile networking

- Mobility models needs 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.*

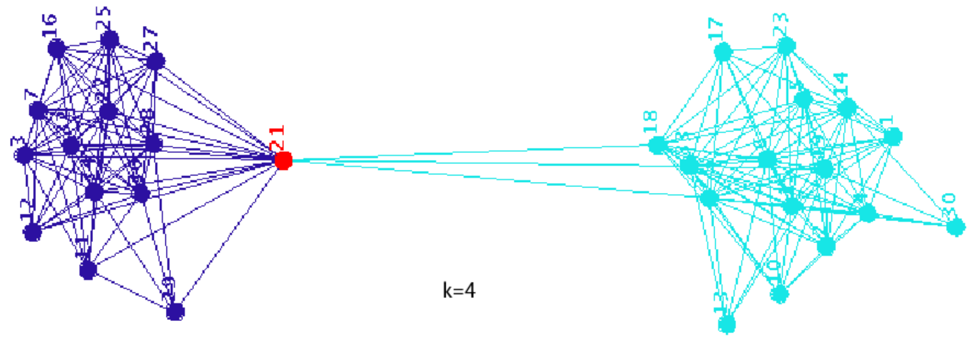
3b Connectivity&Routing Ever More Social



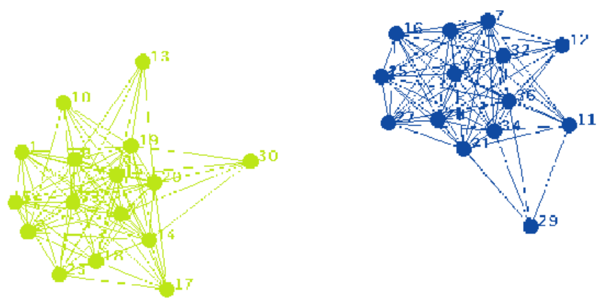
K-clique Communities in Cambridge Dataset



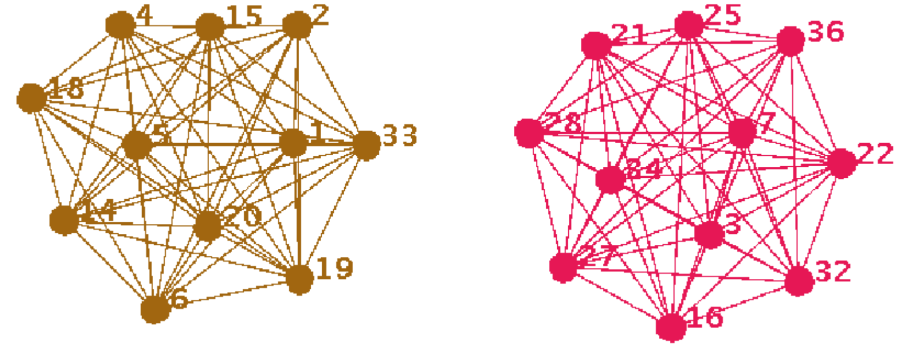
k=3



k=4

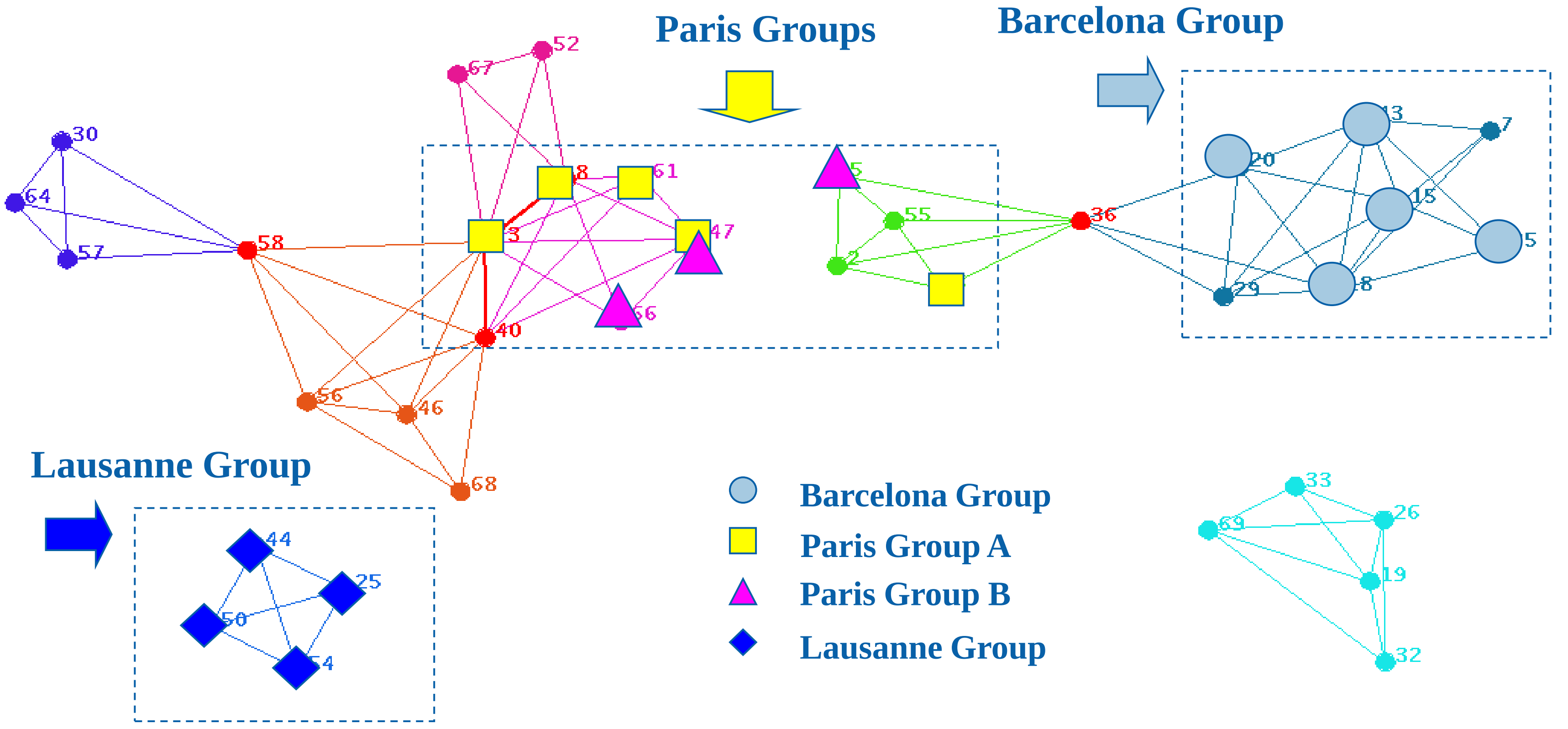


k=5



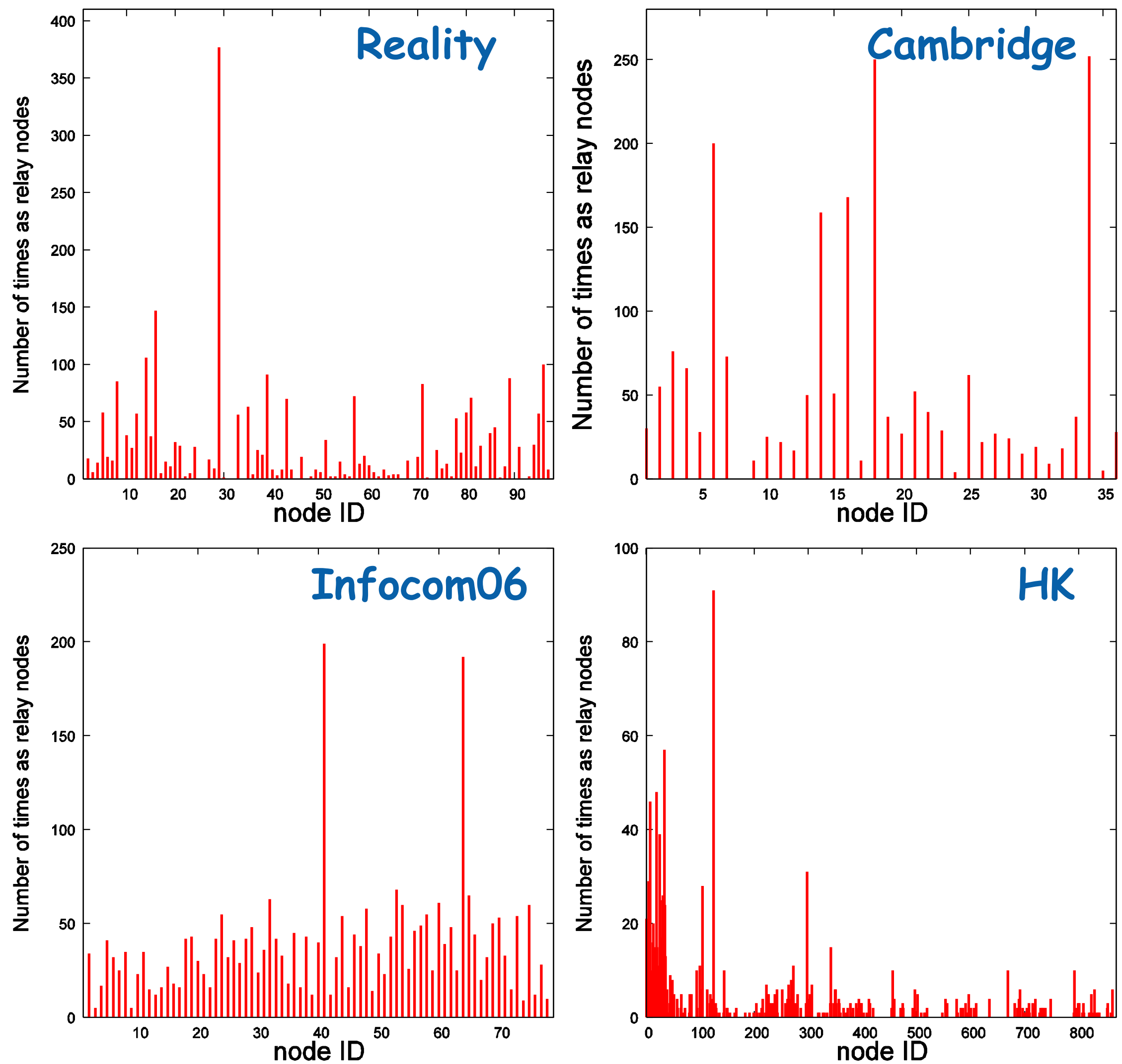
k=10

K-clique Communities in Infocom06 Dataset

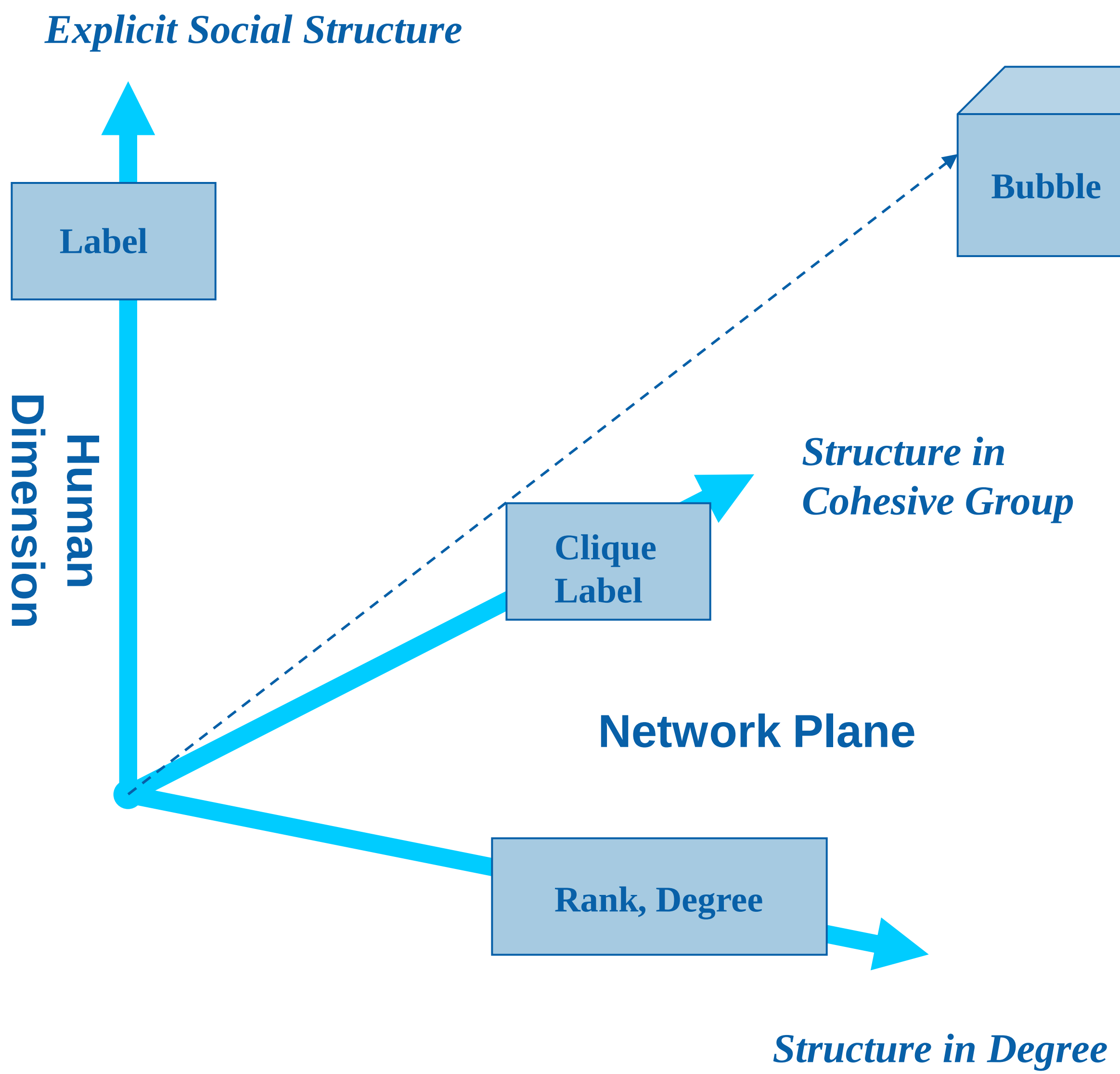


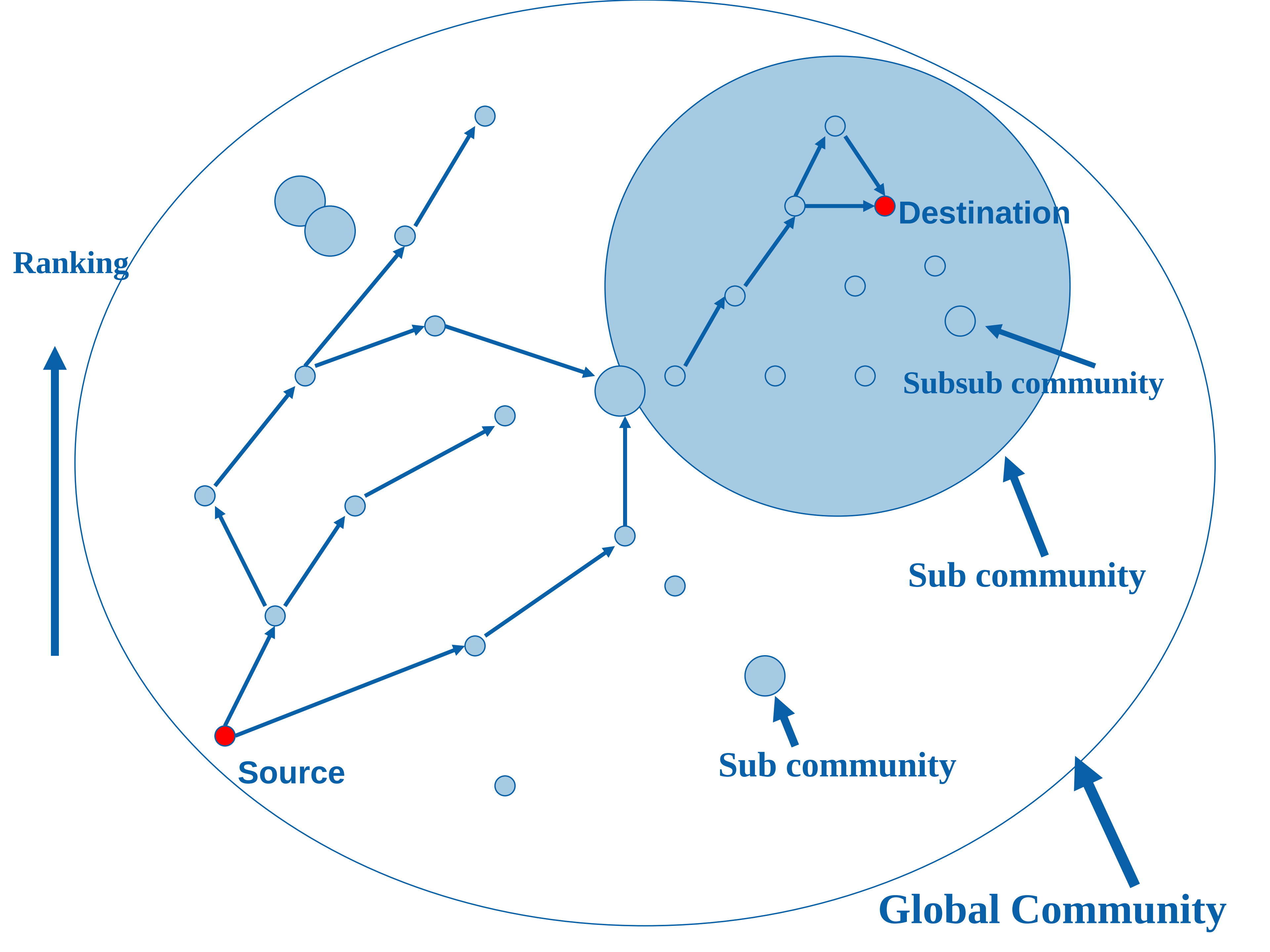
K=4

Human Hubs: Popularity

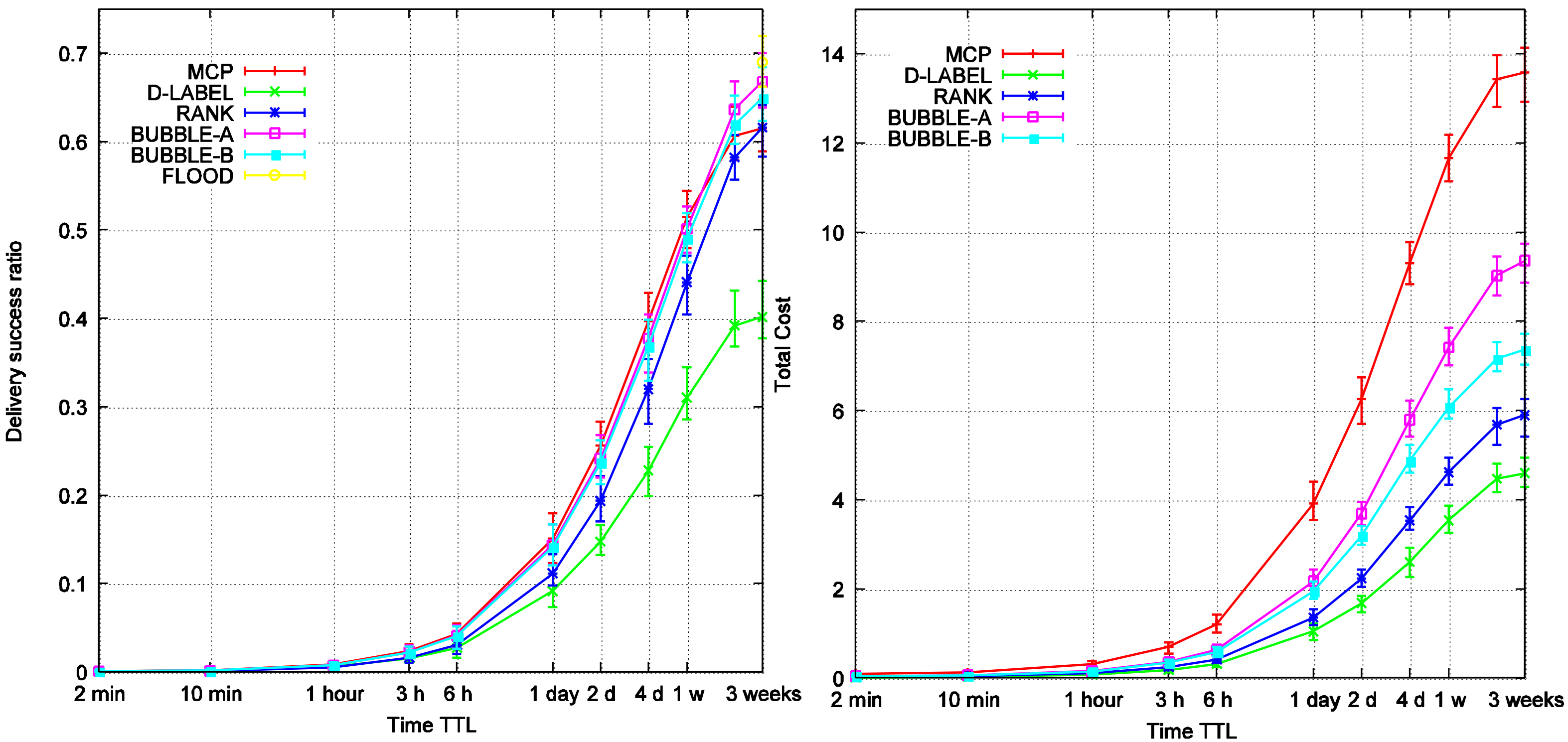


Forwarding Scheme Design Space




Destination
Ranking
Subsub community
Sub community
Sub community
Source
Global Community

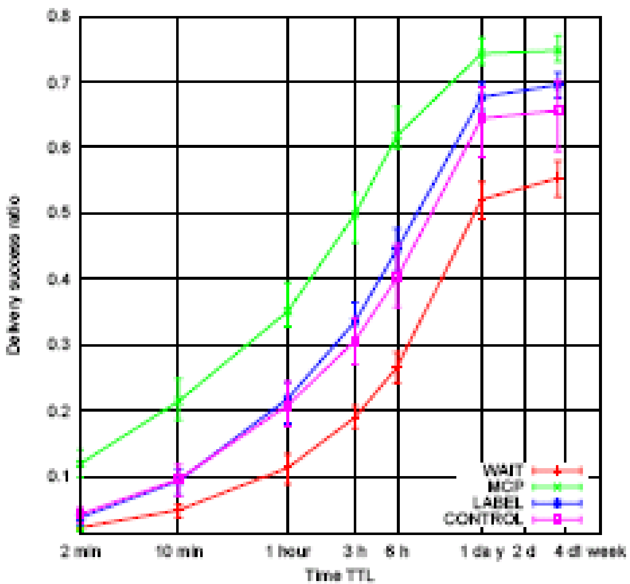
Use affiliation+hubs to fwd inter+intra cliques



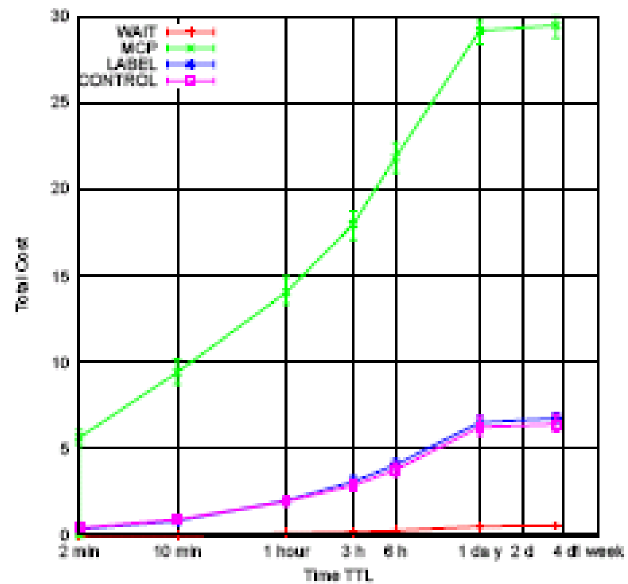
3c Connectivity&Routing 3 - Community Detection

Community improves forwarding

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



(a) Delivery



(b) Cost

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

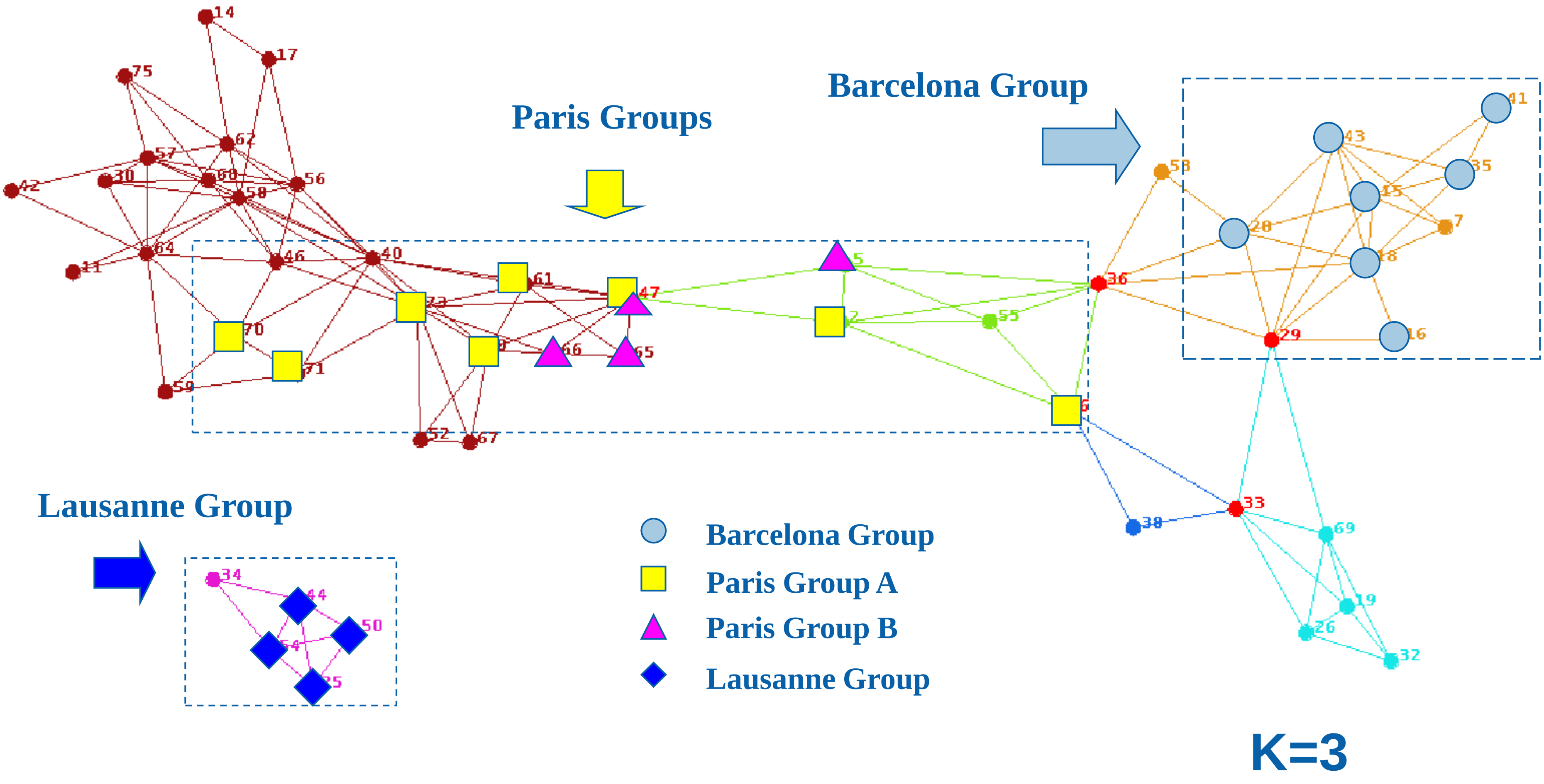
- 2-clique (connected components)
- 3-clique (overlapping triangles)

Overlapping feature

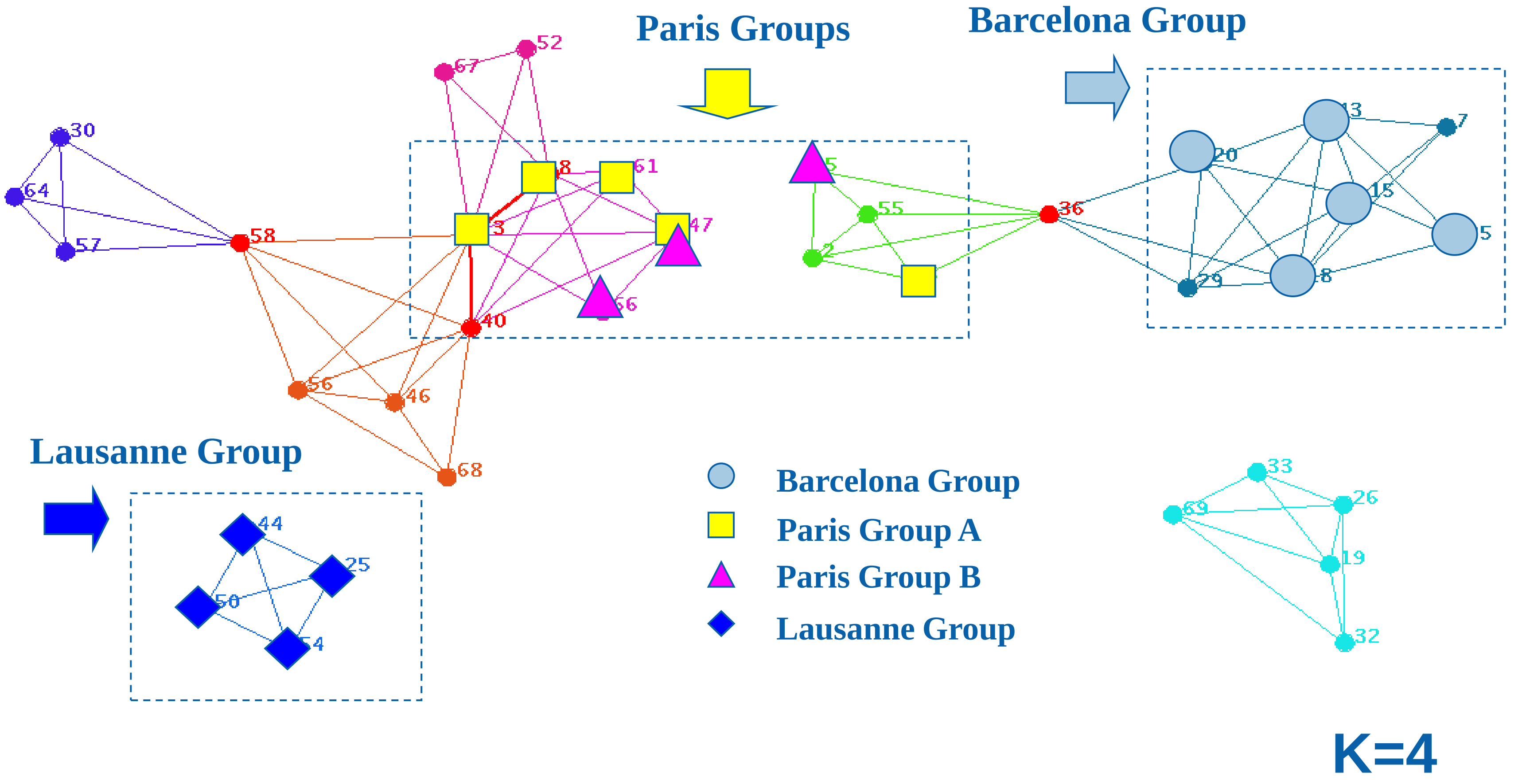
Percolation threshold

$$p_c (k)= 1/[(k-1)N]^\wedge(1/(k-1))$$

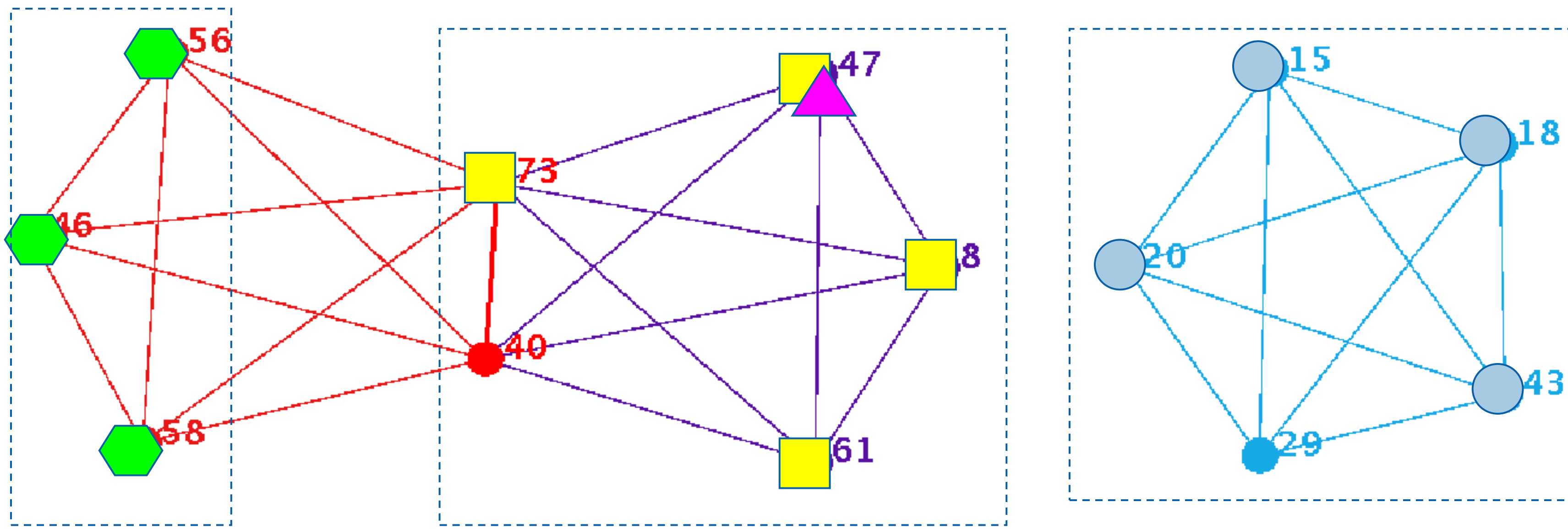
K-clique Communities in Infocom06 Dataset



K-clique Communities in Infocom06 Dataset



K-clique Communities in Infocom06 Dataset



Italian

Paris Group A (French)

Paris Group B (French)

Barcelona Group (Spanish)

K=5

Weighted network analysis (WNA)

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

Experimental dataset	Infocom06	Cambridge	Reality	Infocom05
Q_{max}	0.2280	0.4227	0.5682	0.3039
Max. Community Size	13	18	23	13
No. Communities	4	2	8	4
Avg. Community Size	8.000	16.500	9.875	6.5
No. Community Nodes	32	33	73	26
Total No. of Nodes	78	36	97	37

Distributed Community Detection

SIMPLE, K-CLIQUE, MODULARITY

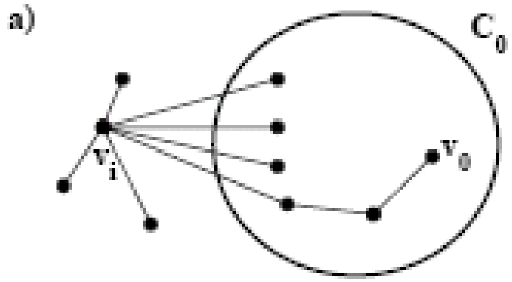
Terminology : Familiar Set (*F*), Local Community (*C*)

Update and exchange local information during encounter

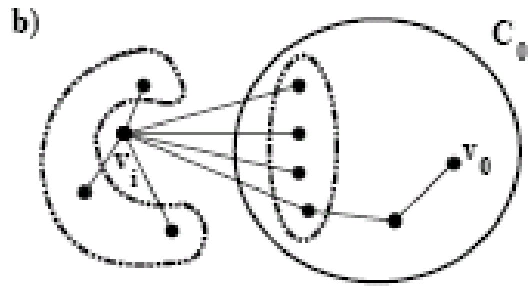
Build up Familiar Set and Local Community

- *CommunityAccept*( ), *MergeCommunities*( )

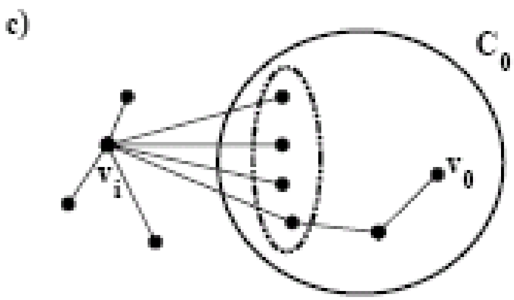
SIMPLE



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 =

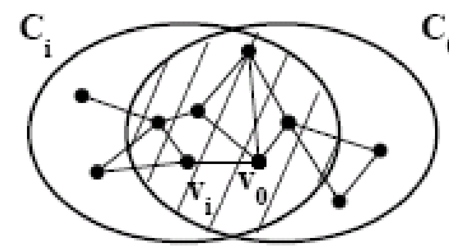


Then we count the number of vertices in both v_i's familiar set and also in the local community of v_0 =

d)

And we admit v_0 to C_0 iff

$> \times \lambda$



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

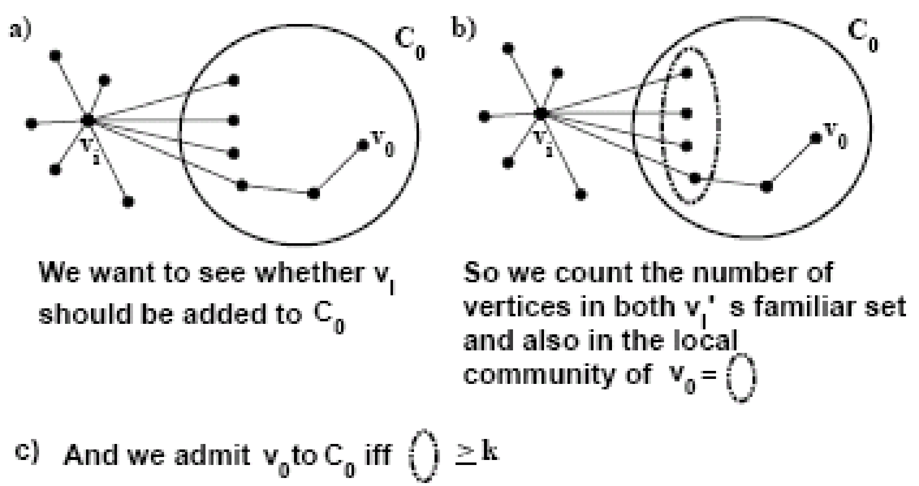
MergeCommunities (C_o, C_i)

CommunityAccept (v_i)

K-CLIQUE

CommunityAccept (v_i) : $|F_i \cap C_0| \geq k-1$

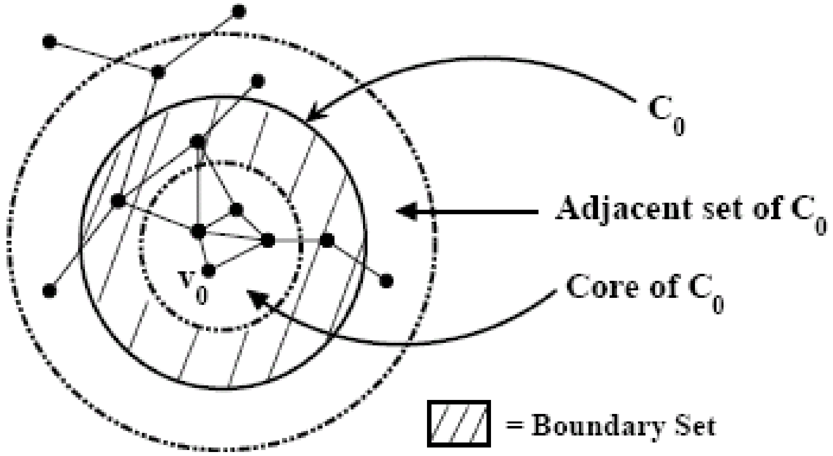
MergeCommunities(C_o, C_i): $|\tilde{F}_j \cap C_0| \geq k-1$



CommunityAccept (v_i)

MODULARITY

Boundary Set $$B_0 = \{v_i \mid (v_i \in C_0)\ and\ ((F_i \setminus C_0) \neq \emptyset)\}$$



Local Modularity $$R_0 = \frac{I}{|T|}$$

Measure of the sharpness of local community

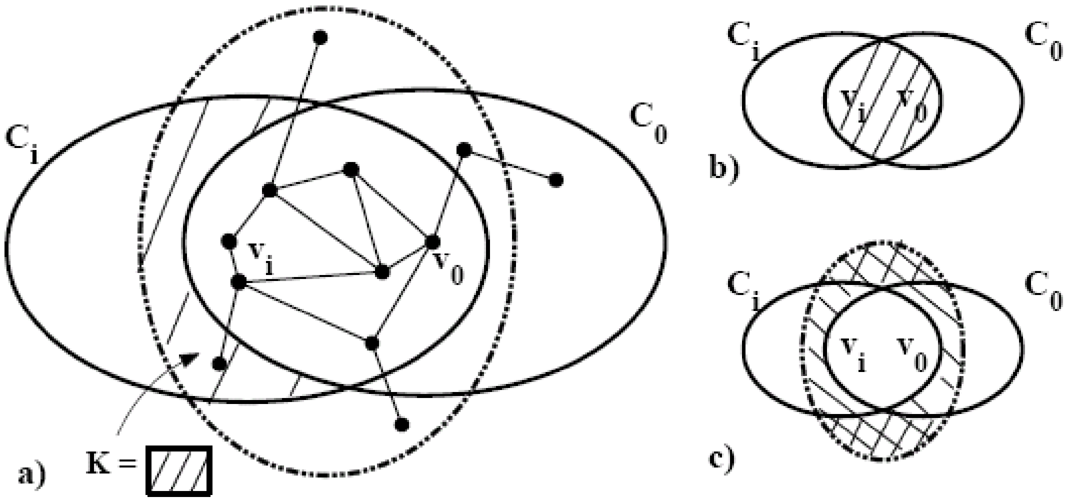
MODULARITY

CommunityAccept (v_i) :

$$(F_i \subseteq C_0 \; and \; B_0 \neq \emptyset) \quad \text{or} \quad \Delta R_0^i > 0$$

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

$$\tilde{F}_k \subseteq C_0 \quad \text{or} \quad \Delta R_0^k > 0$$



Results and Evaluations

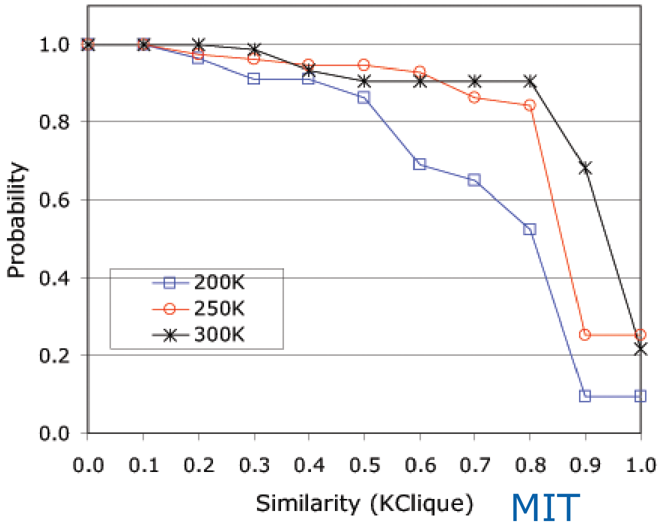
Data Set	SIMPLE	K-CLIQUE	MODULARITY
Reality	0.79/0.81	0.87	0.89
UCSD	0.47/0.56	0.55	0.65
Cambridge	0.85/0.85	0.85	0.87
Complexity	O(n)	$O(n^2)$	$O(n^4)/O(n^2k^2)$

Newman weighted analysis

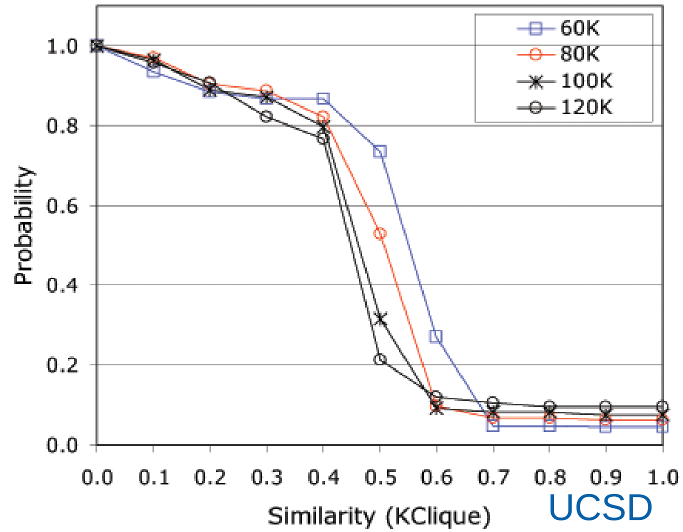
Palla et al, k-Clique

$$\sigma_{Jaccard} = \frac{|\Gamma_i \bigcap \Gamma_j|}{|\Gamma_i \bigcup \Gamma_j|}$$

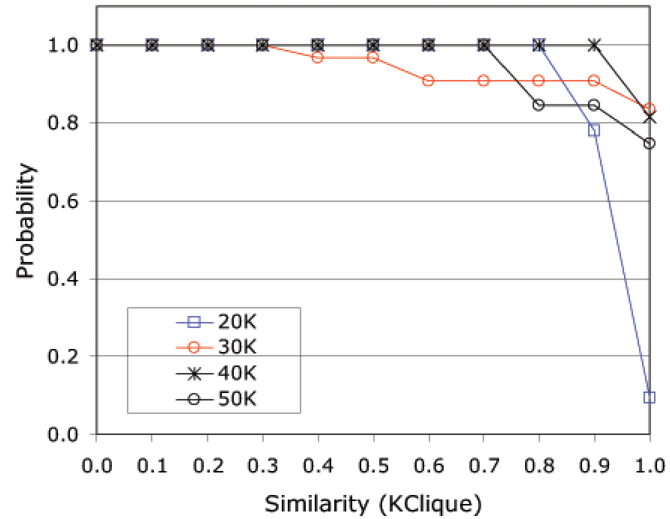
Results and Evaluations



MIT



UCSD



Distributions

of

Local Community Views

Outlook

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