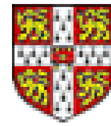


Network Architecture Research Considerations Or The Internet Conspiracy

Jon Crowcroft (University of Cambridge)

James Scott (xxx Research Cambridge)

Pan Hui University of Cambridge)

Christophe Diot (Thomson)

Mothy Roscoe (ETH)

Why we shouldn’t do network arch?

- Liberally borrowed slides from Mothy Roscoe (ex cambridge, sprint, intel, berkeley, now ETH)
  - Post-modernism?
  - No to FIND/GENI?
- Experience
  - Architectures emerge (ANSA/Herbert)
  - papers about them are usually Post hoc rationalisations
- First, general Arguments, Then specific (Haggle) example...

Why have an architecture?

- What does an Internet Architecture hope to achieve?
  - Interoperability across networks
  - Easier for applications to code to
  - Framework for providers to compete
- Does *any* Internet architecture really address these issues?

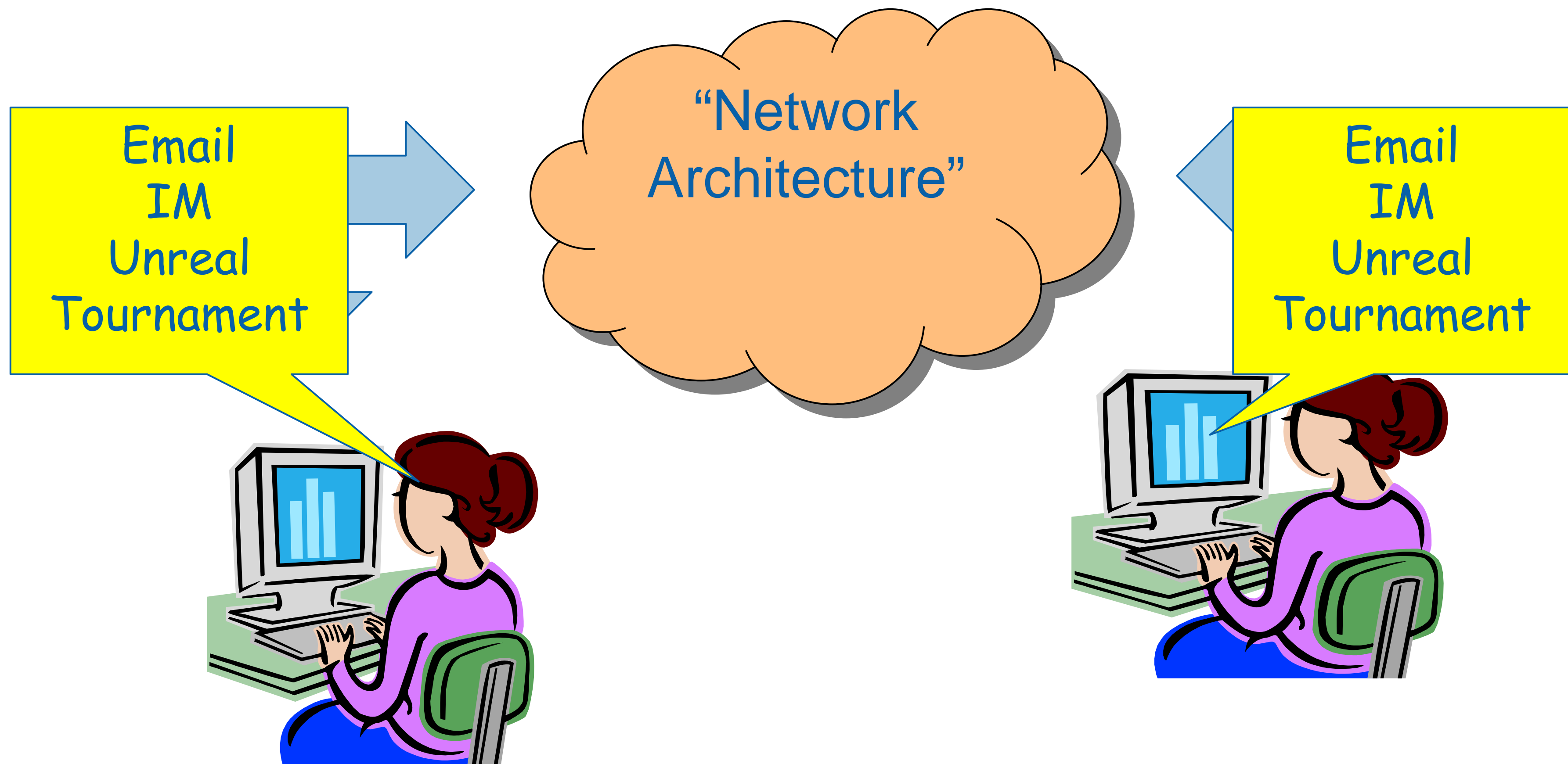
What has the “Internet Architecture” done for us lately?

- Interoperability:
  - No – not really☺
- Uniform API:
  - Bad thing: hides useful features of the underlying network – c.f. cross layer optimisations and other oxymorons
- Provider framework:
  - Has any tier-1 ISP ever made significant profit from offering IP service?

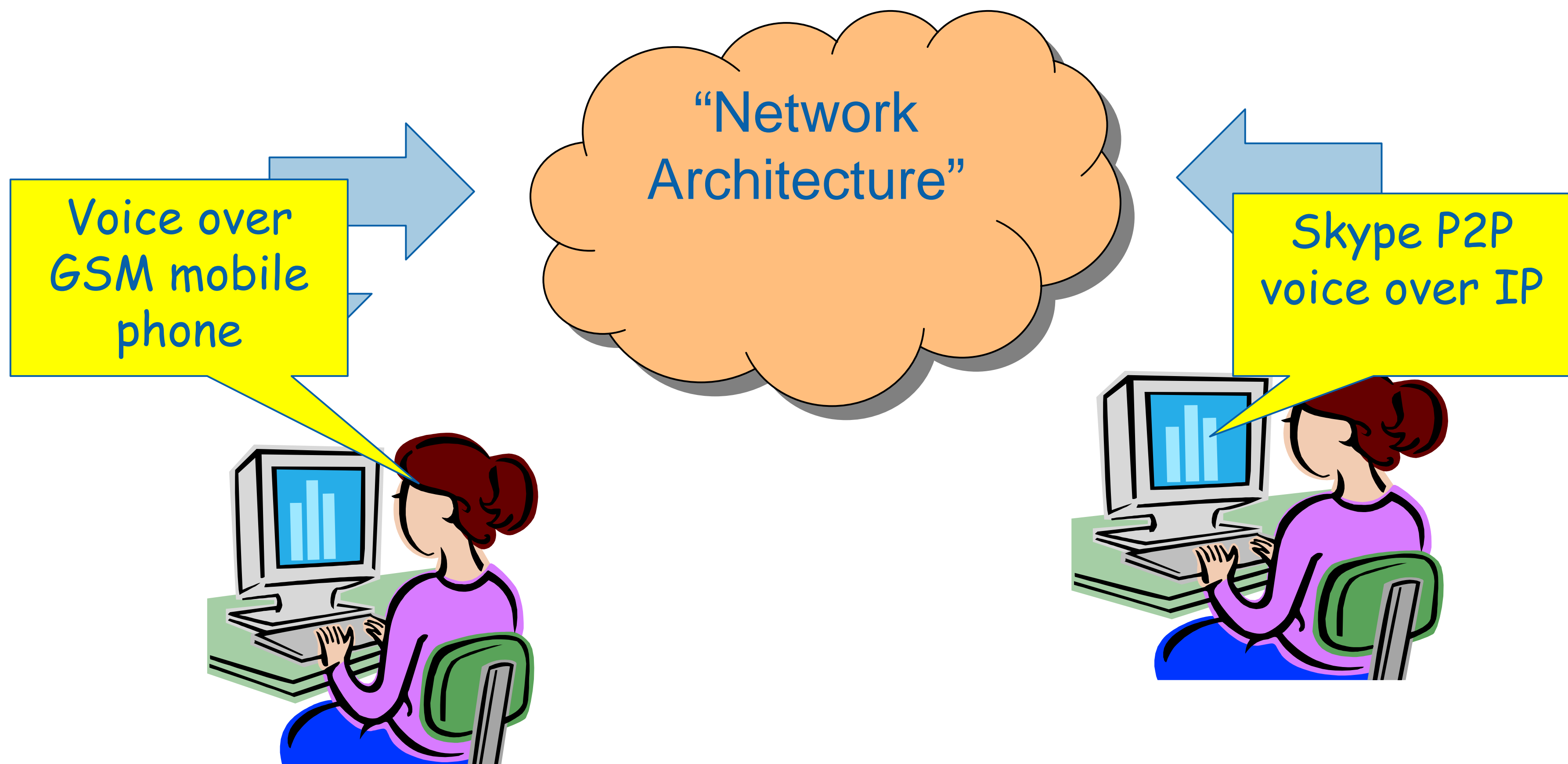
What has the "Internet Architecture" done for us lately?

- It *used to* enable rapid innovation
- Claim: lack of attention to value flow & economics was a Good Thing!
  - High commercial value blunts innovation (c.f. other industries)
  - Disruption is bad for business
- Idea: goal of networking research should be to enable surprising things

One (purist?) view of network architecture

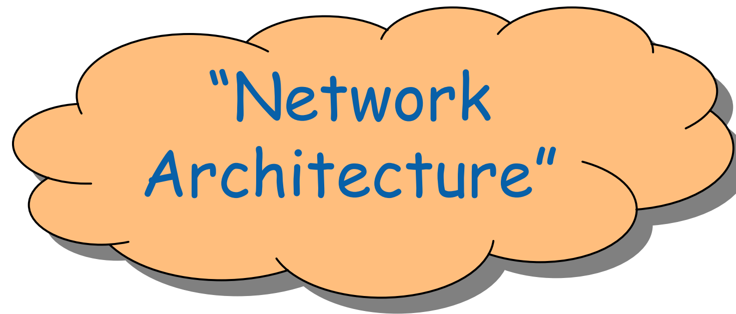


One (pluralist) view of network architecture



Different (seperatist/corrosive/layered) view of network architecture

"Systems people"



"Networking people"

A third way…

- Sidestep the whole purist/pluralist debate
- Avoid an architecture completely
  - It's unnecessary
  - It doesn't address any problems
  - It may be counterproductive
- Let me give a detailed worked example from the Haggle Project :- IST-4-027918-IP

The alternative

- There is no architecture
  - Just a bunch of computers with links
  - Removes “semantic bottleneck” from application writers
- Embrace “radically heterogeneous networking”
  - Two end-systems should be able to communicate even if they have *nothing* in common: protocols, address realms, semantics.

Surely something must be common?

- Of course: but it is only *the application*.
  - $\Longrightarrow$ that what is common is that which is application-specific

- Even DTN has single overlay architecture
  - (bundle routing)

By way of example: Haggle: a framework for Networking Mobile Users

- Wireline Networks: Solid
- MANET: Liquid
- Haggle: Gas
- Manage Phase Changes between these – reality of wireless is that we need to move from
  - Gas -> Solid
  - Gas -> Liquid
- More often than we previously realized

Motivation: Mobile users currently have a very bad experience of networking

- Applications do not work without infrastructure
- Local connectivity is plentiful (WiFi, Bluetooth, etc), but very hard to use transparently
- E.g. messaging/file transfer to others in this room?
- E.g. If I had used a modem to get some cached web content (e.g. news) earlier, and you wanted to access it, how can we share it?

The wireless networking research community has failed to support our end users

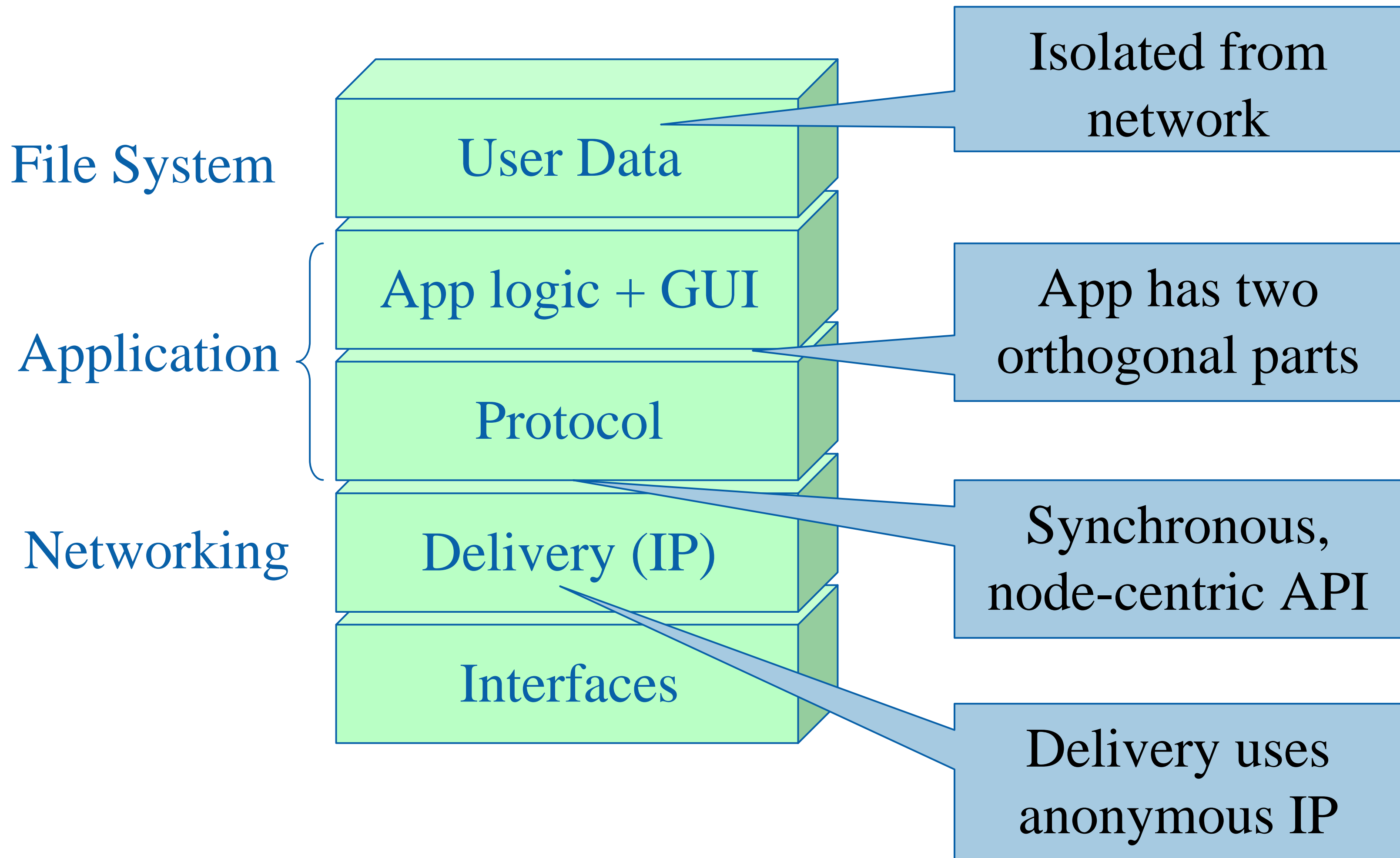
Our approach: Haggle

- Clean slate design of networking subsystem for mobile devices
- Haggle: a layerless, modular network architecture designed around the needs of mobile users
- Aims:
  - Allow applications to take advantage of all types of data transfer (neighborhood, infrastructure, mobility)
  - Allow networking endpoints to be specified by user-level naming schemes rather than node-specific network addresses,
  - Allow limited resources to be used efficiently by mobile devices, taking into account user-level priorities for tasks

Design principles for Haggle (not an architecture, no no no:-)

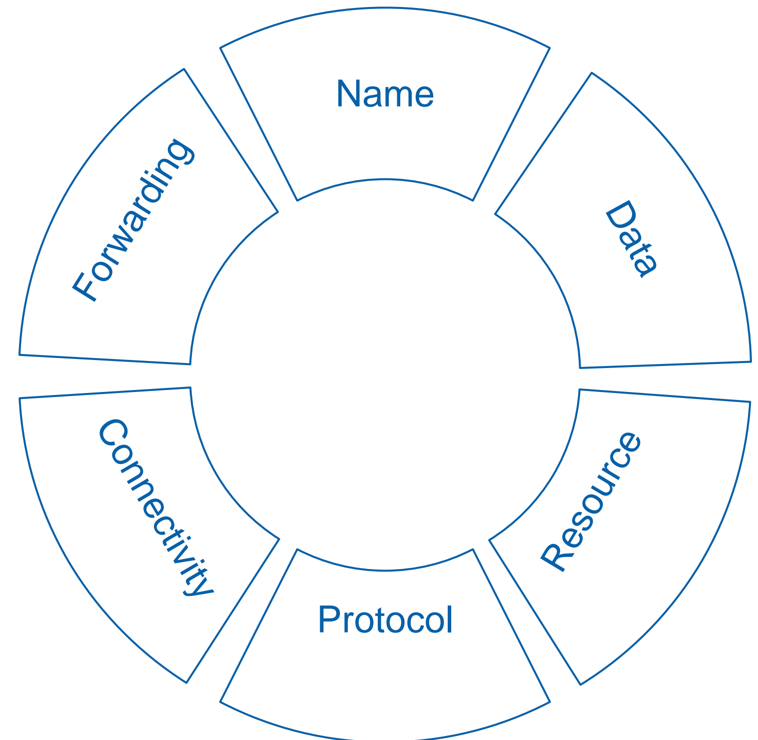
A. Forward using application layer information
    A. De-layer
B. Asynchronous operation
    A. Do not assume path
C. Empower intermediate nodes
    A. Do not separate host from router
D. All user data kept network-visible, and req/resp
    A. Tx==rx==store - all equal class functions
E. Exploit all data transfer methods
    A. Greed is good
F. Take advantage of brief connection opportunities
    A. Impatience is good
G. Empowered and informed resource management
    A. Know and tell what it costs

Current device software framework



Haggle Overview

- Clean-slate redesign of mobile node
- Spans MAC to application layers (inclusive), but is itself layerless – uses six “managers”
- API is asynchronous and data-centric
- Store-and-forward architecture with data persisting inside Haggle not separate file sys
- App-layer protocols (SMTP, HTTP, etc) moved into Haggle rather than apps themselves
- Naming/addressing done in graphs instead of stacks
- Resource management built-in



Data Objects (DOs)

- DO = set of attributes = {type, value} pairs
  - Exposing metadata facilitates search
- 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]

Protocols

- Protocols are methods for communicating with other nodes, including Haggle-specific protocols as well as legacy protocols (e.g. email)
- Example: protocol to send an ADU over an ad-hoc WiFi connection given a neighbour's MAC address
- Example: protocol using keywords as addresses via Google – e.g. satisfy a query for "world news"
- Legacy example: encapsulate an ADU in an email and send it using SMTP to an email address

Delivery using user-level names

- Plutarch, IPNL, RSIP etc etc etc
- Names come from:
  - Network, e.g. using neighbour discovery (12:AB:23:98:BE:FF)
  - Applications, e.g. Jon Crowcroft » jon.crowcroft@cl.cam.ac.uk
- Some names are also “addresses” i.e. data can be delivered to that name using one of the protocols available
- Delivery engine needs to:
  - Sense “nearby” addresses (e.g. Bluetooth inquiry gives MAC addresses, Internet connectivity means all email addresses are deliverable)
  - *Known-sender:* Map between ADU's destination name(s) and addresses of suitable next-hop nodes
  - *Known-recipient:* Determine suitable nearby nodes which may be sources or help find sources for requested data
  - Describe these potential transfers and their benefits to the resource manager

Resource management using tasks

- Resource management uses a list of tasks mainly provided by delivery engine
  - E.g. perform discovery on interface I
  - E.g. send ADU X to neighbour Y on interface Z
- Each task has an associated benefit and cost
  - Benefit is specified by task provider. May be time-dependent (i.e. using a pointer-to-function)
  - To get cost, resource use is estimated, and then the “cost” is a function of the resource use * the resource scarcity
  - Resource manager then schedules execution of tasks in order of highest benefit/cost ratio.

So where are we?

- Problem definition and some idea of solution
- NOT architecture:- framework for others to use(Sourceforge)
    - Others develop use cases
    - Architecture will emerge from common uses
- Other work we've done:
    - Measurement/analysis of human connectivity patterns
    - Measurement of in-motion wireless networking properties
- Ongoing work in European project
    - More measurement and analysis of possible data paths
    - Implementation of modular software framework
    - Addressing challenges of naming/addressing, forwarding, security/trust, usability, *legacy compatibility*

Questions?

- Thank you... ... ...

A. Forward using application layer information

- Use names meaningful to apps, e.g. human names, keywords for documents, parameters for content wanted (mime-type, etc)
- Delivery of data is accomplished by using the data itself to choose forwarding path, rather than artificial meaningless-to-the-user addresses such as IP.

B. Asynchronous operation

- Apps can indicate network actions when natural to them; actual transfers can happen asynchronously when suitable connectivity occurs
- Late binding of user-level names to network-level addresses means that up-to-date context information can be used, e.g. dynamic IP address
- Support non-contemporaneous, store-and-forward connections

C. Empower intermediate nodes

- Much content is public/sharable – e.g. webpages
- Thus any intermediate node may also be a valid destination, e.g. it's user might also be interested in the webpage later
- Additionally, the intermediate node can be a source for that data too – e.g. it meets another node who is interested in the webpage

D. Message switching

- Message switching is useful for principles A,B,C
- Application layer forwarding information applies to whole messages
- With variability inherent in non-contemporaneous data paths, packet switching would result in lots of useless half-messages arriving, wasting bandwidth
- Intermediate nodes cannot gain benefits unless entire data units are made available to them

E. All user data kept network visible

- Data should not be stored privately by applications, but kept in the Haggle framework where it can be shared with other devices.
- Even your most personal data can be shared – with your other devices. Encryption can be used to prevent unauthorised parties snooping.
    - Painful data synchronisation systems (e.g. phone/PC) can be made obsolete!
- Public data is often popular and duplicated locally, so making this visible allows us to find more sources for it

F. Build request-response into the network

- User-level tasks are sometimes inherently two-way, e.g. a request for content and the response including the content
- Supporting this in the network framework itself rather than in apps allows all nodes to be used for the "turnaround point"
- I.e. all nodes can cache application data (even if they do not run any app that can understand it), and respond to a request for that data

G. Exploit all data transfer methods

- Different transfer methods have different properties
  - Synchronous (Bluetooth), asynchronous (SMS)
  - Zero cost (neighbourhood), cost-per-hour (WiFi hotspot), zero cost till monthly limit hit (SMS)
  - Physical-layer bandwidths, latencies, loss rates, etc
- A given transmission may be sent using multiple diverse paths, e.g. by email and later by Bluetooth directly.

H. Take advantage of brief connection opportunities

- Connection opportunities can be fleeting
  - E.g. driving past an AP or another car
- Must optimise transmissions to maximum user benefit
- N.B. current protocols such as Web, SMTP, are really bad at this (work to be presented in WMCSA 2006)

I. Empowered and informed resource management

- Haggle has the potential to use up all your device's resources and really piss you off
- It also has the potential to do resource management correctly, something today's devices don't do
  - Storage: your disk is not full, why not?
  - Networking: your WiFi interface has one queue, why?
  - Battery: why use static levels for "plenty" and "little"?
- Resource management must be put centre stage in Haggle

J. Use and integrate with existing application infrastructure where possible

- Incremental deployability if a Haggle node can interact with another node's legacy apps
- Can reuse existing (familiar and complex) apps via pretending to be a legacy protocol rather than having to push out new ones
- Leverage vast infrastructures that are already deployed – e.g. email servers, IM servers, the Google index, etc

Pocket Switched Networking (PSN): Scenario for Mobile Humans

- Study/define problem before attempting solution
- Pocket Switched Networking: the scenario that the mobile user of consumer IT apps faces every day
  - Humans carry one or more devices with them, each with wireless networking capabilities and storage
  - These devices experience *neighbourhood* (e.g. Bluetooth) and *infrastructure* (e.g. 802.11 AP) connection opportunities (with differing bandwidths, costs, etc)
  - Human *mobility* generates these opportunities as they move around with their normal mobility patterns.

Pocket Switched Networking: Application traffic

- In PSN, we can identify two classes of application traffic:

known-sender where one node needs to transfer data to a user-level destination (not a network-level address), e.g.:

  - Another user (who may own many nodes)
  - All users in a certain place/with a certain interest
  - Users with a certain role (e.g. "police"), etc.

known-recipient in which a device requires content of some sort, but it is irrelevant where the data comes from, e.g.:

  - Publicly distributed content such as "current news webpage"
  - Media files, e.g. "songs by the Scissor Sisters"
  - Locally generated information, e.g. traffic news

- For known-sender, there may be many recipients; for known-recipient, there may be many sources.

Why status quo (IP) does badly in PSN

- IP doesn't handle many-recipient well, and does not handle many-source at all
- IP's strict layering means infrastructure lookups to find endpoint addresses before data transmission begins
- TCP/IP's stream abstraction means that apps have to implement app-layer protocols; these rely on access to specific infrastructures
    - E.g. email user wants "message to James" but email client implements "message to IP address of MX DNS record of James's email domain"
- IP cannot handle non-contemporaneous connectivity, e.g. use of human mobility as data transfer opportunity
- Packet switching means that app-layer data is lost to the network; further exchanges of the same data means insertion of the data into the network by an application