File Synchronization with Syxaw in an Ad-hoc Network Work in Progress

Fuego Core Project
Tancred.Lindholm@hiit.fi

Contents

- Example synchronization scenario
- The Syxaw file synchronizer
  - Focus on how data is shared
- The ad-hoc networking environment
- Basic Syxaw + ad-hoc = Baseline Model
- The Baseline Model
- Problems of the Baseline Model
- The Specializable Base Model
- Syxaw and P2P
- Conculsions

Example Scenario

- Alice and Bob with laptops A and B cooperate on writing software KillerApp at work
- They normally synchronize their laptops with the company server S
- They both meet on bus on their way to Moscow, and decide to both do some work on KillerApp (what else is there to do... :))
- Wanting to track each others progress they need to synchronize
- Synchronizing via S is not possible
- Ideal case if A and B could talk to each other directly
- Alice and Bob use our synchronizer

The Syxaw File Synchronizer

- Synchronizes files and directories on different devices
- Works in the wireless environment with
  - high latency
  - relatively low/costly bandwidth
  - intermittent connectivity
- On devices with
  - data stored in files
  - limited computing abilities (cycles cost battery life)
  - ever-increasing storage capacity (>1Gb common soon)
- Next, we look at the way file Syxaw shares files, so we can understand what happens to this model in the ad-hoc world

Syxaw Share Model: Starting Locally

- Each device has some local name for its objects

- Keep these names, as they are already implemented, and frequently meaningful to the user (file names)
- UID = unique id for an object (in scope of a device)

Global Names: LID+UID=GUID

- Create a globally unique name (GUID) by prefixing the UID (local name) with a **location id** (LID)
- Trivial implementation: use DNS names for LIDs (better: use real host identities)

LID prefix

www.hiit.fi/0023
www.hiit.fi/1232
www.hiit.fi/4242
www.hiit.fi/6672
www.hiit.fi/3145
www.hiit.fi/4023
www.hiit.fi/4232

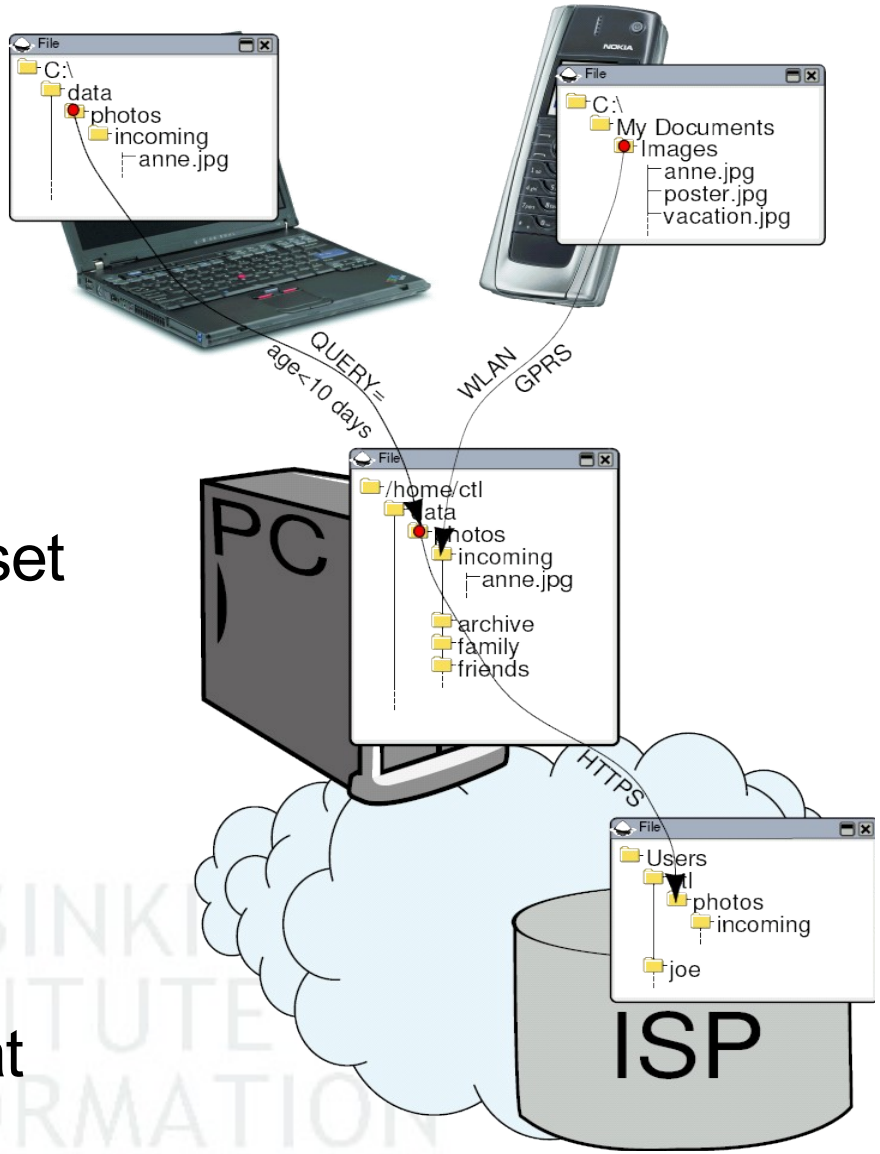
- location = unit sharing a space of consistent, named objects. May be a e.g. single device, or a clustered file server

Pairing Objects with a *Synchronization Link*

- To synchronize = changes from one object flow to another.
- To indicate that two objects should synchronize, we create a **link** between these
- Example 1: A user wants to synchronize his home PC folder of photos with that on the PDA
  - Link the folders
  - Synchronize (link is persistent for later syncs)
- Example 2: A user wants to join a project workspace, receives a link to it in his email
  - Actuating the link causes a local folder linked to the workspace to be created

Example: Linked directory trees

- Links draw as 
- Photos are captured with a camera phone and synchronized to the PC
- Phone↔PC synchronization set up by linking `Images` to `incoming`
- The laptop links to the PC; a query is used to get photos <10 days old
- The PC syncs with an ISP that provides archival storage



Share Model Definitions

- Each link has two ends. From the POV of a location *L*
  - **branch**: the object on *L*
  - **base**: the object on some other device (it is the "base" of the object *L*)
  - The link is local knowledge to *L*
  - $b \rightarrow a$ = branch b linked to base a
- *L* initiates synchronization with its base
- E.g. phone book on PDA linked to company phone-book
  - To sync, initiate sync on the PDA. Local changes on PDA->company phone book and vice versa
- Object in the example Alice's A = a, Bob's B=b, server =s

Share Model Definitions

- To name the content of an object at a point in time we assign version numbers to the object, aka *commit* the object
- An object may act as both a base and a branch
  - photos folder on the PC:
    base of laptop, phone; branch of ISP
  - we expose different content and metadata depending on the role of the object = *facets*
  - local facet = acting as branch of a link
  - remote facet = acting as the base of a link
  - local and remote facet version numbers different
- Syxaw process running at a location = instance
- Illusion of single mutable object = singleton contract

Some Syxaw Features

- Sync protocol has only 3 operations on top of HTTP
  - No directory operations!
- Entails XML reconciliation framework
- Directory tree synchronization in XML
  - expressing directory trees as XML
  - reconciling using the XML reconciliation framework
- Download and upload operate on batches of objects, saving a lot of round-trip times
- Xebu “binary” serialization of XML
- HTTP has proved useful in the world of NATs, firewalls etc.

Discrete vs. Continuous Sync

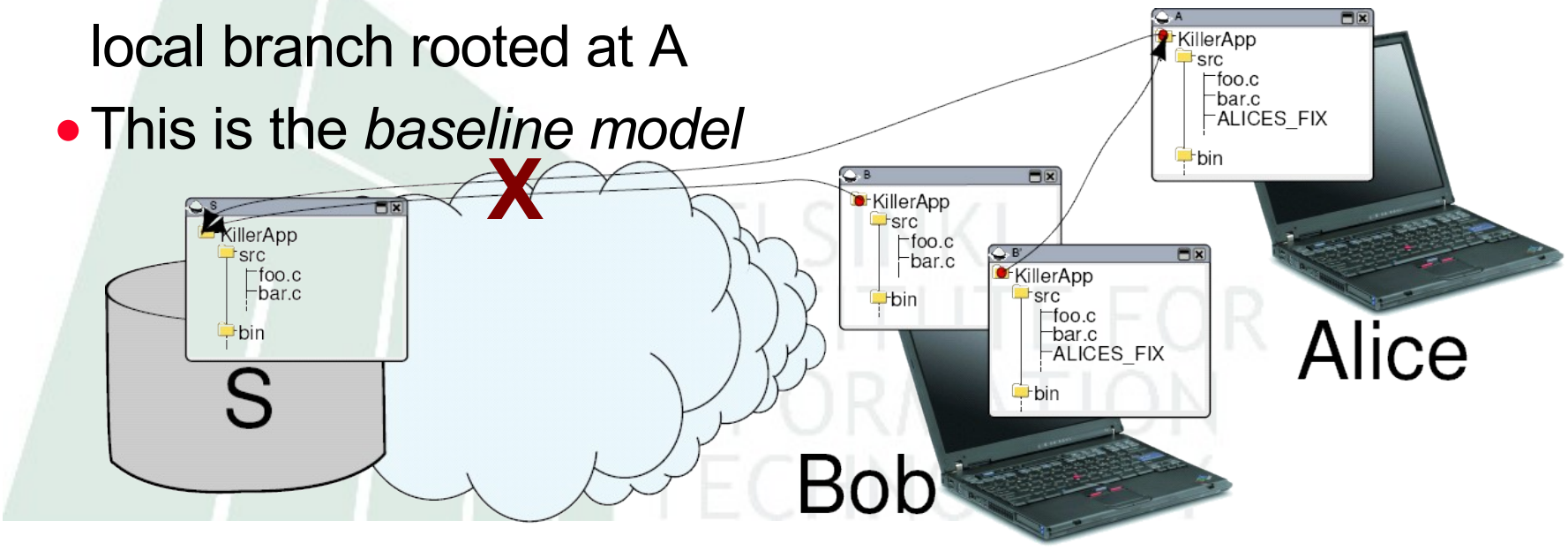
- Syxaw built on discrete synchronization runs. Why?
- Some points against continuous
  - Discrete allows control of when to sync = control of **cost** and **power**
  - Unnecessary data transfer at high price/power
  - Less efficient data transfer
  - Is continuous sync confusing in some cases?
- Some points for
  - Having to sync before you start = latency
  - You may be on an expensive network when you need data
  - Is explicit sync confusing in some cases?

The Ad-Hoc Environment

- **Newbie alert:** my knowledge of this is recent and limited, so I really appreciate expert comments
- Baseline: fixed infrastructure wireless network = FIWnet
- Durability, stability, connectivity, capacity < FIWnet
- Partitioning, in particular from FIWnet servers
- Changing network addresses - what is the identity of a peer?
- Consider multiple network transports. E.g. IR,IP,Bluetooth
- Service discovery
- Group formation, access control
- Heterogeneous execution environment
- Of lesser concern: radio, QoS, ad-hoc vs FIWnet power...

Approach #1: The Baseline Model

- Let's address the scenario using the existing share model
- Assume we an ad-hoc transport between A and B
- Elect either of A or B to act as base, the other as branch. Here: A=base, B=branch
- Create share b' on B so that b'→a and synchronize b'
- Alice and Bob then work on a local branch rooted at A
- This is the *baseline model*



Baseline Issues: Multiple Transports

- *Call channel* = synchronous RPC with small headers + streamed data.
- Sync protocol currently on a HTTP (+TCP+IP) call channel
- We need a call channel implementation for e.g. Bluetooth
- What transport to use when contacting LID *x*?
- Solution 1: lookup mechanism
  - LID x $\rightarrow$ transport *t* + identity *y*
- Solution 2: LID encodes transport
  - LID t:y $\rightarrow$ transport *t* + identity *y*
  - Not really architecturally sound, but may be simpler

Baseline Issues: Identity

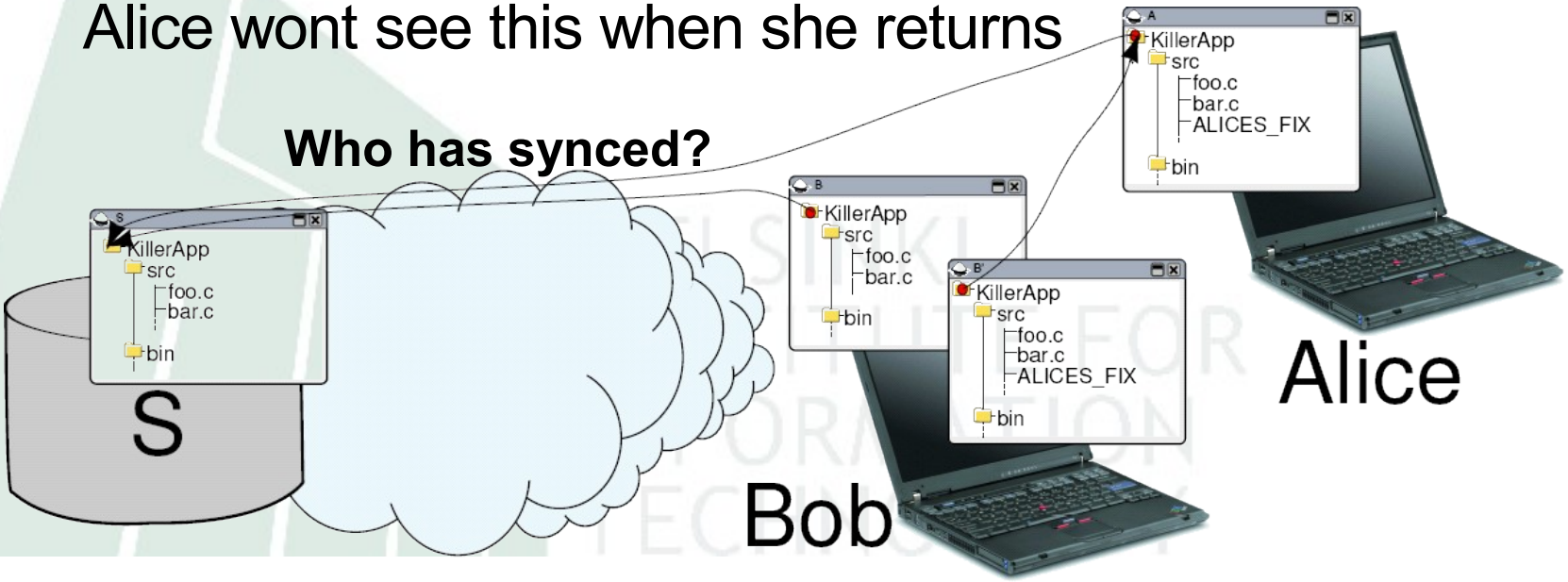
- How do we go from transport *t* + identity *y* to get a network address *n*?
- Solution 1: *y* encodes a persistent network address on *t*
  - E.g. MAC, which we then RARP to an IP
  - Persistent network address is a problem
- Solution 2: *y* encodes a persistent network-addressable identity on *t,* i.e. *"t=n"*
  - For IP, there is Host Identity Protocol (HIP) that does exactly this
  - .. or use Zeroconf to query IP by identity
- Solution 3: Syxaw identity lookup
  - Extend Syxaw with a service to lookup $y \rightarrow n$

Baseline Issues: Discovery

- How do we discover Syxaw instances?
- Solution 1
  - Register Syxaw with a Service Discovery framework
  - Use the native protocol of the framework for discovery
- Solution 2
  - Let Syxaw implement own discovery protocol
- Synergy between Syxaw native instance-to-address lookup and discovery
  - Implementing only one of these makes little sense?

Problems of the Baseline Model

- Having to set up an additional share b' is awkward
- Due to local visibility of links and version numbers, change set tracking between s, a, b and b' won't work
  - b' can't synchronize with s when Bob is back
  - Bob can't tell is the changes of b' is already in b
  - If Bob manually copies changes to b and synchronizes, Alice wont see this when she returns



What is Needed to Fix this?

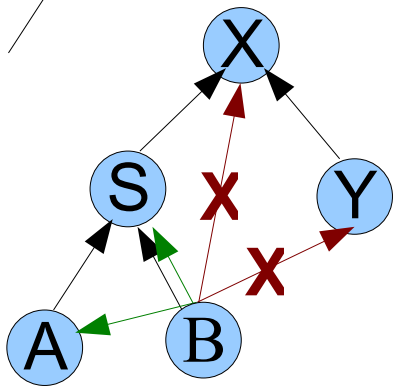
1. To eliminate the need for additional shares (such as b'), allow a link to be reset to another base = *branch switching*
2. It should be possible to detect which change sets have been included into an object
3. It should be possible to commit data on behalf of another location. This is a *proxy commit*
   - Avoid the strangeness of having changes but not being able to commit them

- We propose a model that adheres to these requirements, but in a restricted manner

= *the Specializable Base Model*

Approach #2: The Specializable Base Model

- Allow base switching along base-branch axis
  - Do not allow going beyond original base =
  - Do not allow switching between two branches
  - Seems versatile enough
  - Introduces useful simplicity (?)



- Track versions and change sets in this setting
  - we need a better “version number”
  - local version = LID+local number
  - version number = list of local versions in order of branching. E.g. (<s,1>,<a,5>)

The Specializable Base Model

- Committed change sets encoded in `joins` metadata field
- `joins` lists the most recent version committed from all branches, e.g.

```
joins=(<s,10>) (<s,9>,<a,8>)
```

- Problem: `joins` grows as number of branches grows, even after a branch is no longer used
  - Future work
- In practice, we have found these version numbers and `joins` useful
  - Future work: what is it that makes them useful?

Syxaw and P2P

- P2P is popular, perhaps even overhyped
- Can it offer Syxaw anything in an ad-hoc environment?
- P2P decentralization can lower the probability of being partitioned from the base
- Also, we hope that P2P features like scalability and robustness can be “borrowed”
- P2P in the ad-hoc environment seems to be in its infancy
  - Comments on this?

Syxaw and P2P

- Two approaches: beneath or above the singelton object
- Beneath
  - P2P system offers distribution of logically centralized, mutable object
  - Hard/impossible to implement in practice
- Above
  - P2P offers publishing of object versions into the ad-hoc cloud
  - Syxaw instances can then download newer versions using P2P, provided they have no local changes
  - Committing changes still over HTTP $\rightarrow$ avoid difficult decentralized change protocols

In Conclusion

- Syxaw needs changes to work well in an ad-hoc networking environment
- Aspects we examined
  - Multiple transports
  - Network Identity
  - Instance Discovery
  - Versioning model
  - Use of P2P

Thank You!

Open Source at

http://hoslab.cs.helsinki.fi

“Fuego core” project