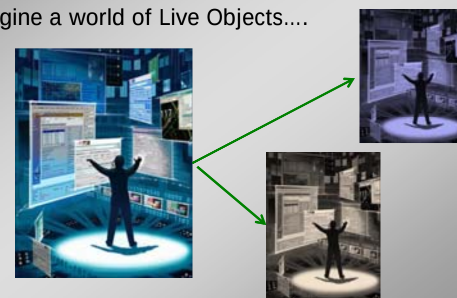
Live Objects

Krzys Ostrowski, Ken Birman, Danny Dolev
Cornell University, Hebrew University*

(*) Others are also involved in some aspects of this project… I'll mention them when their work arises…

Live Objects in an Active Web

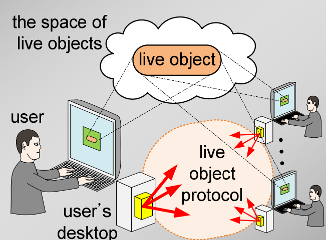
- Imagine a world of Live Objects….



- …. and an Active Web created with "drag and drop"

Live Objects in an Active Web

- Imagine a world of Live Objects….



- …. and an Active Web created with "drag and drop"

Live Objects in an Active Web

- User builds applications much like powerpoint
  - Drag things onto a "live document" or desktop
  - Customize them via a properties sheet
  - Then share the live document
- Opening a document "joins" a session
  - New instance can obtain a state checkpoint
  - All see every update…
  - Platform offers privacy, security, reliability properties

When would they be useful?

- Build a disaster response system…
  … in the field (with no programming needed!)
- Coordinated planning and plan execution
- Create role-playing simulations, games
- Integrate data from web services into databases, spreadsheets
- Visualize complex distributed state
- Track business processes, status of major projects, even state of an application

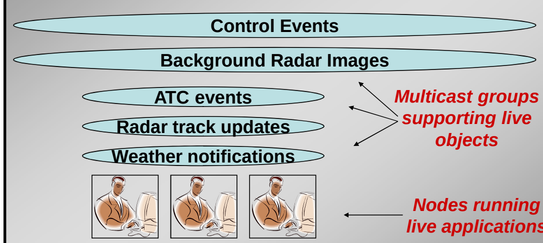
Big deal?

- We think so!
  - It is very hard to build distributed systems today. If non-programmers can do the job numbers of such applications will soar
  - Live objects are robust to the extent that our platform is able to offer properties such as security, privacy protection, fault-tolerance, stability
- Live objects might be a way to motivate users to adopt a trustworthy technology

The drag and drop world

- It needs a global namespace of objects
  - Video feeds, other data feeds, live maps, etc…
  - Our thinking: download them from a repository or (rarely) build new ones
- Users make heavy use of live documents, share other kinds of live objects
- And this gives rise to a world with
  - Lots of live traffic, huge numbers of live objects
  - Any given node may be "in" lots of object groups

Overlapping groups



… posing technical challenges

- How can we build a system that…
  - Can sustain high data rates in groups
  - Can scale to large numbers of overlapping groups
  - Can guarantee reliability and security properties
- Existing multicast systems can't solve these problems!

Existing technologies won't work…

Kind of technology	Why we rejected it
IP multicast, pt-to-pt TCP	*Too many IPMC addrs. Too many TCP streams*
Software group multicast solutions ("heavyweight")	*Protocols designed for just one group at a time; overheads soar. Instability in large deployments*
Lightweight groups	*Nodes get undesired traffic, data sent indirectly*
Publish-subscribe bus	*Unstable in large deployments, data sent indirectly*
Content-filtering event notification.	*Very expensive. Nodes see undesired traffic. High latency paths are common*
Peer-to-peer overlays	*Similar to content-filtering scenario*

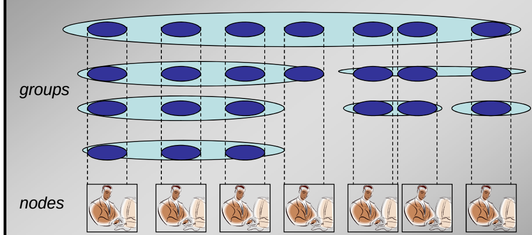
Steps to a new system!

1. First, we'll look at group overlap and will show that we can simplify a system with overlap and focus on a single "cover set" with a regular, hierarchical overlap
2. Next, we'll design a simple fault-tolerance protocol for high-speed data delivery in such systems
3. We'll look at its performance (and arrive at surprising insights that greatly enhance scalability under stress)
4. Last, ask how our solution can be enhanced to address need for stronger reliability, security

Coping with Group Overlap

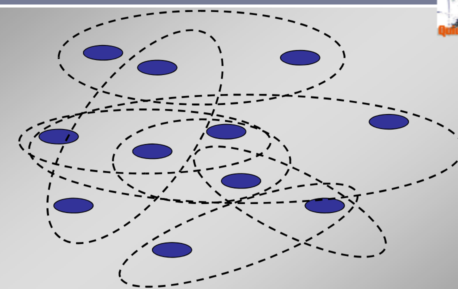
- In a nutshell:
  - Start by showing that even if groups overlap in an *irregular* way, we can "decompose" the structure into a collection of overlayed "cover sets"
  - Cover sets will have *regular* overlap
    - Clean, hierarchical inclusion
    - Other good properties

Regular Overlap



- Likely to arise in a data center that replicates services and automates layout of services on nodes

Live Objects ⇒ Irregular overlap



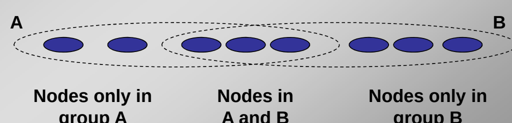
- Likely because users will have different interests…

Tiling an irregular overlap

- *Build some (small) number of regularly overlapped sets of groups ("cover sets") s.t.*
  - Each group is in one cover set
  - Cover sets are nicely hierarchical
  - Traffic is as concentrated as possible
- Seems hard: $O(2^G)$ possible cover sets
- In fact we've developed a surprisingly simple algorithm that works really well. Ymir Vigfusson has been helping us study this:

Algorithm in a nutshell

1. Remove tiny groups and collapse identical ones
2. Pick a big, busy group
   1. Look for another big, busy group with extensive overlap
   2. Given multiple candidates, take the one that creates the largest "regions of overlap"
3. Repeat within overlap regions (if large enough)

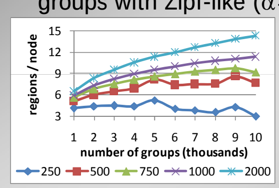


Why this works

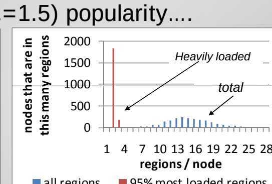
- … in general, it wouldn't work!
- But many studies suggest that groups would have power-law popularity distributions
  - Seen in studies of financial trading systems, RSS feeds
  - Explained by "preferential attachment" models
- In such cases the overlap has hidden structure… and the algorithm finds it!
- It also works exceptionally well for obvious cases such as exact overlap or hierarchical overlap

It works remarkably well!

- Lots of processes join 10% of thousands of groups with Zipf-like (α=1.5) popularity….



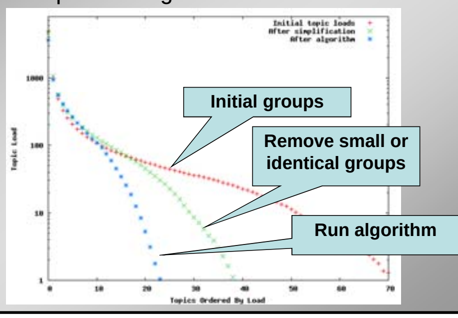
Nodes end up in very few regions (100:1 ratio…)



And even fewer "busy" regions (1000:1 ratio)!

Effect of different stages

- Each step of the algorithm "concentrates" load

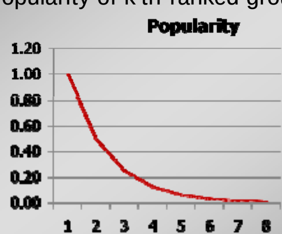


… but not always

- It works very poorly with "uniform random" topic popularity
- It works incredibly well with artificially generated power-law popularity of a type that might arise in some real systems, or with artificial group layouts (as seen in IBM Websphere)
- But the situation for human preferential attachment scenarios is unclear right now… we're studying it

Digression: Power Laws…

- Zipf: Popularity of k'th-ranked group ≈ $1/k^{\alpha}$



- A "law of nature"

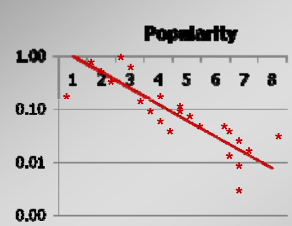
Zipf-like things

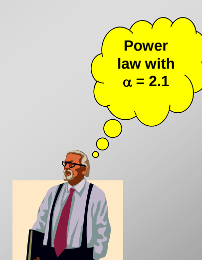
- Web page visitors, outlinks, inlinks
- File sizes
- Popularity and data rates for equity prices
- Network traffic from collections of clients
- Frequency of word use in natural language
- Income distribution in Western society
- … and many more things

Dangers of "common belief"

- Everyone knows that if something is Zipf-like, instances will look like power-law curves
- Reality? These models are just approximate…
  - With experimental data, try and extract statistically supported "model"
  - With groups, people plot log-log graphs (x axis is the topic popularity, ranked; y-axis counts subscribers)
  - Gives something that looks more or less like a straight line… with a lot of noise

Dangers of "common belief"





But…

- Much of the structure is in the noise
- Would our greedy algorithm work on "real world" data?
  - Hard to know: Live Objects aren't widely used in the real world yet
  - For some guesses of how the real world would look, the region-finding algorithm should work… for others, it might not… a mystery until we can get more data!

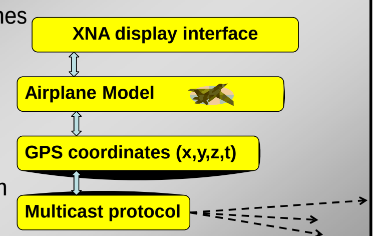
When in doubt…. Why not just build one and see how it does?

Building Our System

- First, build a live objects framework
  - Basically, a structure for composing components
  - Has a type system and a means of "activating" components. The actual components may not require code, but if they do, that code can be downloaded from remote sites
- User "opens" live documents or applications
  - … this triggers our runtime system, and it activates the objects
- The objects make use of communication streams that are themselves live objects

Example

- Even our airplanes were mashups
- Four objects (at least), with type-checked event channels connecting them
- Most apps will use a lot of objects…



When is an "X" an object?

- Given choice of implementing X or A+B…
  - Use one object if functionality is "contained"
  - Use two or more if there is a shared function and then a plug-in specialization function
- Idea is a bit like plug-and-play device drivers
- Enables us to send an object to a strange environment and then configure it on the fly to work properly in that particular setting

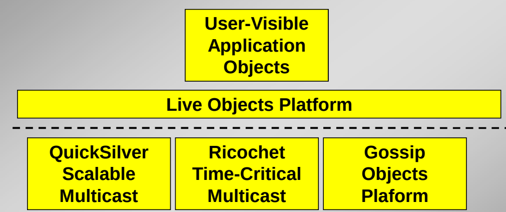
Type checking

- Live objects are type-checked
  - Each component exposes interfaces
  - Events travel on these, and have types
  - … types must match
- In addition, objects may constraint their peers
  - I expect this from my peer
  - I provide this to my peer
  - Here's a checker I would like to use
- Multiple opportunities for checking
  - Design time… mashup time… runtime

Reflection

- At runtime, can
  - Generate an interface: B's interface just for A
  - Substitute a new object: B' replaces B
  - Interpose an object: A+B becomes A+B'+B
- Tremendously flexible and powerful
  - But does raise some complicated security issues!

Overall architecture



So… why will it scale?

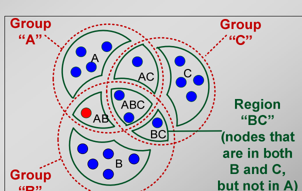
- Many dimensions that matter
  - Lots of live objects on one machine, maybe using multicore
  - Lots of machines using lots of objects
- In remainder of talk focus on multicast scaling…

Building QSM

- Given an "enterprise" (for now, LAN-based)
  - Build a "map" of the nodes in the system
  - … annotated by the live objects running on each
- Feed this into our cover set algorithm… it will output a set of covers
- Each node instantiates QSM to build the needed communication infrastructure for those covers

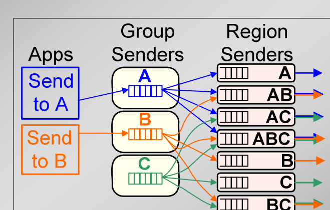
Building QSM

- Given a regular cover set, break it into regions of identical group membership
- Assign each region its own IP multicast address



Building QSM

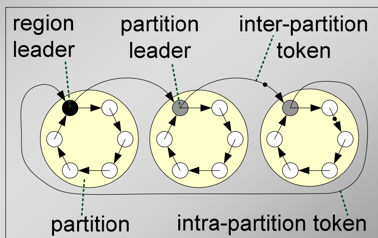
- To send to a group, multicast to regions it spans



- If possible, aggregate traffic into each region

Building QSM

- A hierarchical recovery architecture recovers from message loss without overloading sender

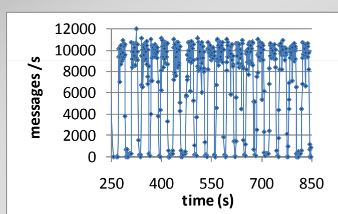


… memory footprint: a key issue

- At high data rates, performance is dominated by the reliability protocol
- Its latency turns out to be a function of
  1. **Ring size and hierarchy depth,**
  2. **CPU loads in QSM,**
  3. **Memory footprint of QSM (!!)**
- This third factor was crucial… it turned out to determine the other two!
- QSM has a new "memory minimizing" design

… oscillatory behavior

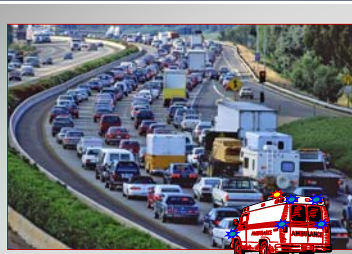
- We also struggled with a form of thrashing



Overcoming oscillatory behavior

- Essence of the problem:
  - Some message gets dropped
  - But the recovery packet is delayed by other data
  - By the time the it arrives… a huge backload forms
  - The repair event triggers a surge overload… causing more loss. The system begins to oscillate
- A form of priority inversion!

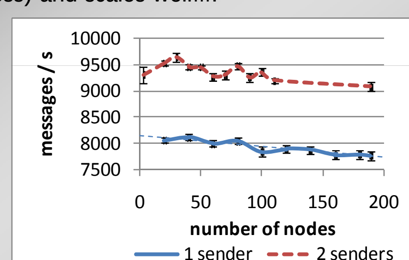
Overcoming oscillatory behavior



- Solution mimics emergency vehicles on a crowded roadway: pull over and let them past!

The bottom line? It works!

- QSM sustains high data rates (even under stress) and scales well….

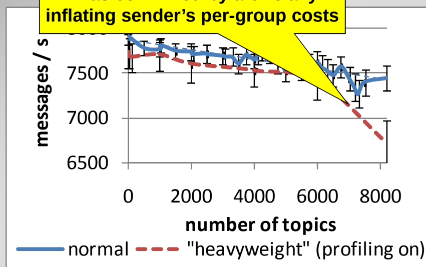


The bottom line? It works!

- ... a ps (topics)

Scalability limited by memory & CPU loads at the sender

... as confirmed by artificially inflating sender's per-group costs



What next?

- Live objects in WAN settings... with enriched language support for extensions

Configuration Mgt. Svc

Port to Linux

Gossip Objects Platform

PPLive/LO

Properties Framework

Learning more

http://liveobjects.cs.cornell.edu