MERCURY: A Scalable Publish-Subscribe System for Internet Games

Ashwin R. Bharambe,
Sanjay Rao &
Srinivasan Seshan

Carnegie Mellon University

Current game architectures

- Distributed broadcast-based (e.g., DOOM)
  - Every update sent to all participants
- Advantages/disadvantages
  - \+ No central server
  - \- Waste of bandwidth
  - \- Synchronized game state – difficult for players to join at arbitrary times

Does not scale beyond small LANs

Current game architectures (cont.)

- Centralized client-server (e.g., Quake)
  - Every update sent to server who maintains "true" state
- Advantages/disadvantages
  - \+ Reduces overall bandwidth requirements
  - \+ State maintenance, cheat proofing much easier
  - \- Bottleneck for computation and bandwidth
  - \- Single point of failure

"Most online games get stuck at about 6,000 players per server, and players on one server can't talk to those on another, reducing the appeal of an online-gaming community." - www.redherring.com

Ideal architecture

Scalability

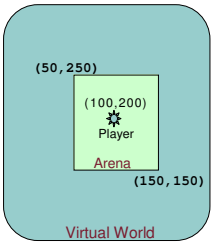
- No hot-spot in the system in terms of:
  - Number of packets routed or received
  - Computational load
- Most efficient use of available bandwidth
  - Every player only receives "relevant" updates

Outline

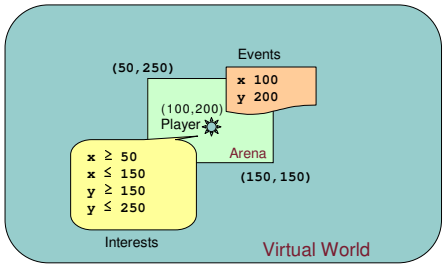
- Scalable game design
- Publish-subscribe systems
- Architecture of MERCURY – a distributed publish-subscribe system
- Preliminary evaluation
  - Scalability
  - Performance
- Future work

Bandwidth efficiency

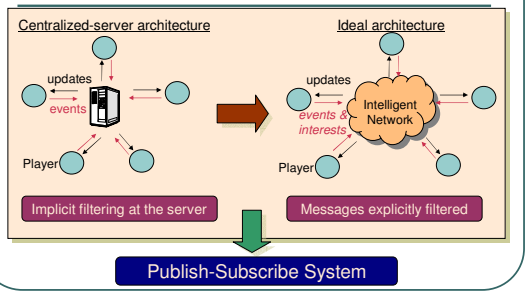
- Events happening in "arena"
  - Need updates of visible or audible entities only
- Other relevant game information
  - Various scores
  - Enemies, teammates
  - Details about other terrain, etc.



Modeling a first person shooting game



Goal architecture



What is publish-subscribe ?

- Publishers produce events or *publications*
- Subscribers register their interests via *subscriptions*
- Network performs routing such that
  - Publications "meet" subscriptions
  - Publications delivered to appropriate subscribers

Critical components

- Subscription language
  - Subjects vs. attribute/values
  - Exact matches vs. regular expressions?
- Routing mechanism
  - Where are subscriptions stored in the system?
  - How are publications routed so that they "meet" subscriptions?

Related systems

- Scribe, Herald
  - Scalable, but –
  - Restricted subscription language
- Siena, Gryphon
  - Flexible subscription language, but –
  - Poor scalability due to message flooding

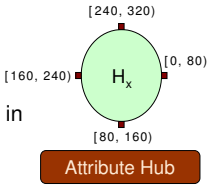
Delicate balance between expressiveness of language and scalability of routing

MERCURY: subscription language

- SQL-like
- Type, attribute name, operator, value
  - Example: int x ≤ 200
- Attribute-values are sortable
- Sufficient for modeling games –
  - Game arenas
  - Player statistics, etc.

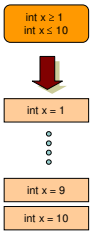
MERCURY: routing protocol

- Each node responsible for range of attribute values
- For each attribute, nodes arranged into circle
- Each node compares value in message to his range; and routes along the circle
- Why not use hashing ?



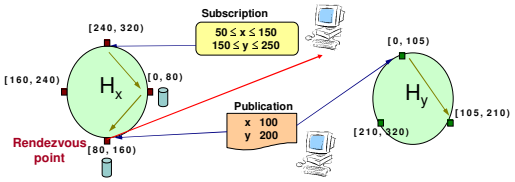
Why not use hashing ?

- Hashing is good for exact matches
- Want to support range queries
- Possible approach
  - Hash each value in the range
  - Problems
    - Can only be used for discrete-valued attributes
    - Too many subscriptions



Routing illustrated

- Send subscription to any one attribute hub
- Send publications to all attribute hubs



Evaluation metrics

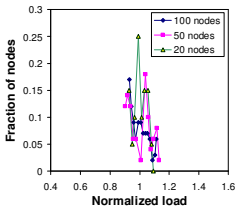
- Scalability metric → load
  - Number of publications routed by a node
  - Averaged over time
- Performance metric → publication delivery delay
  - Time between sending of a publication and its receipt by all subscribers
  - Averaged over all subscribers of a publication

Simulation workload

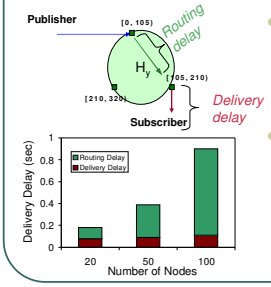
- Modeling of a FPS game
  - Virtual world = square
  - Subscriptions = rectangles around current positions
- Mobility models from ns-2 for modeling player movements

Results: scalability

- Ideal case
  - Every node routes approx. equal #msgs
  - Normalized load is delta function at 1
- Observations
  - Small load fluctuations (± 12%) → good load balancing



Results: performance



- Linear Scaling, however, magnitude is high (~900ms)
  - High number of Mercury level hops
  - *O(n)* worst case!
- Solution: maintain exponentially spaced pointers on the circle
  - Can bring down delay to *O(log n)* hops

Conclusions

- Subscription language expressive enough for games
- Completely decentralized architecture
- Scalability
  - Avoids flooding of subscriptions and publications – reduces network traffic
  - Distributes publications and subscriptions throughout the network – prevents swamping

Future Work

- Performance
  - Simulation shows *publication delivery delay* scales linearly
  - Need much better delay values (~300-400 ms) for real-time game play
  - Cached pointers and network aware placement of nodes → delay competitive with centralized systems
- Realistic workloads
  - Current load balancing depends on workload
  - Introduce BOTs into Quake → collect traces
- Currently building a proof-of-concept system
  - Quake-II+ prototype which uses MERCURY

State maintenance

- Traditional pub-sub = filter
  - No notion of an underlying persistent state
- Game has associated state
- Every publication
  - Is matched against subscriptions and routed
  - Acts as a write event on the underlying database
  - Easily supported by Mercury by writing it at the rendezvous point(s)