Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games

4th IEEE NIME 2008
Las Vegas, USA

Shun-Yun Hu (胡舜元)
syhu@yahoo.com

National Central University, Taiwan, R.O.C.
2008/01/12

ACN Laboratory

State of Online Games Today

- Massively Multiplayer Online Game (**MMOG**) is the fastest growing game genre
  - World of Warcraft (9 M subscribers, 500,000 online)
  - Second Life (10 M accounts, millions of transactions)

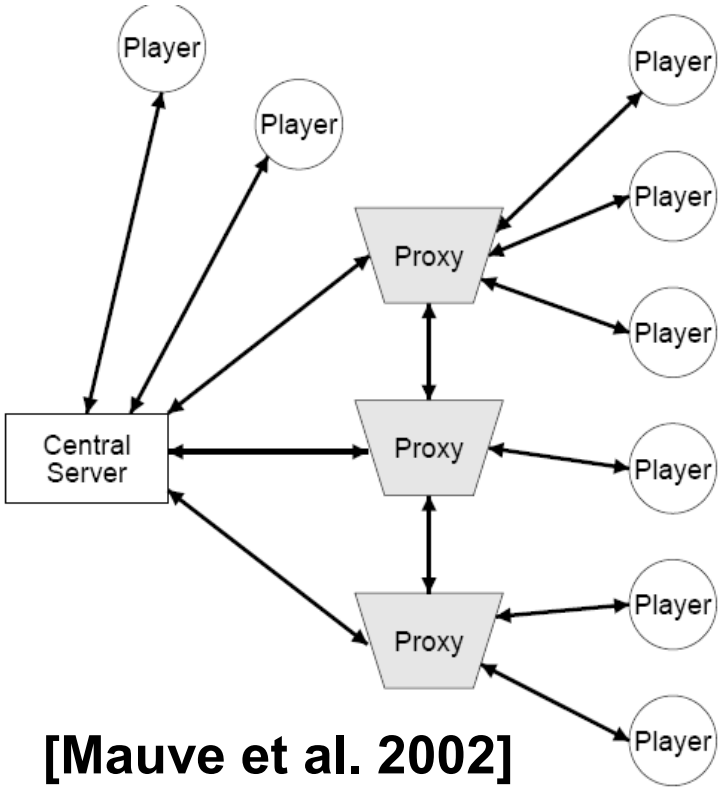
- Max. concurrent users in a world
  - MMORPG: 2,000 ~ 3,000
  - EVE Online: 30,000
  - Second Life: 30,000 ~ 45,000

Motivation

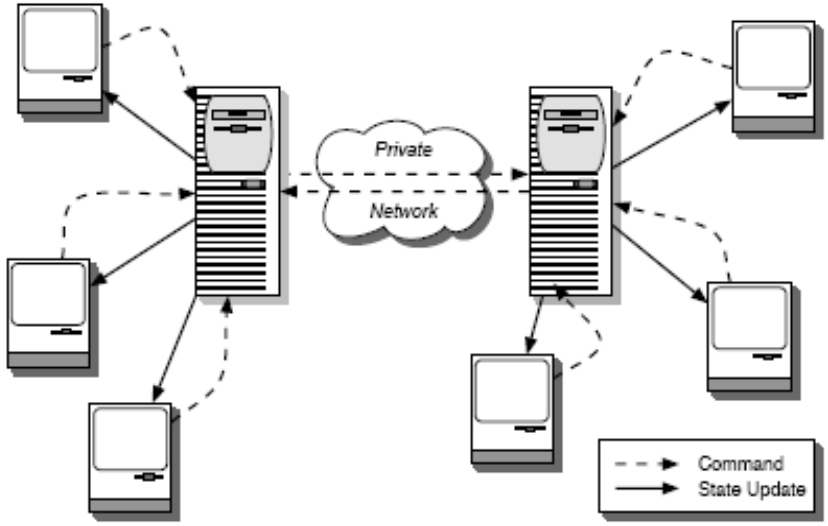
- Today's MMOGs face scalability limitations addressable by *peer-to-peer (P2P)* solutions
- **Goal:** A continuous seamless virtual world with ***millions*** of concurrent users
- Challenges:
  - Heterogeneity
  - Churn
  - Hacking

MMOG server clusters (1/3)

- Replication-based (proxy server & mirror servers)



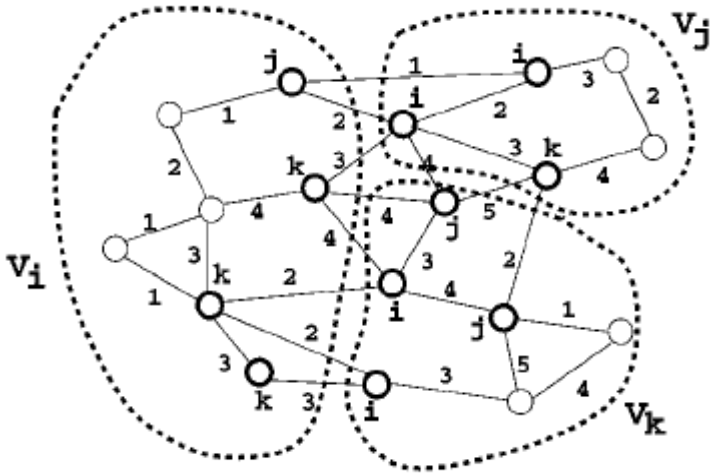
[Mauve et al. 2002]



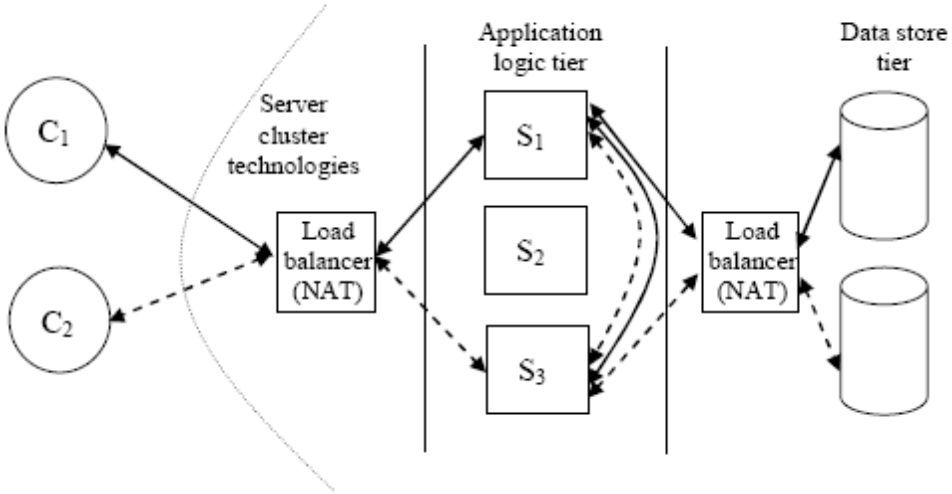
[Cronin et al. 2002]

MMOG server clusters (2/3)

- Object-based



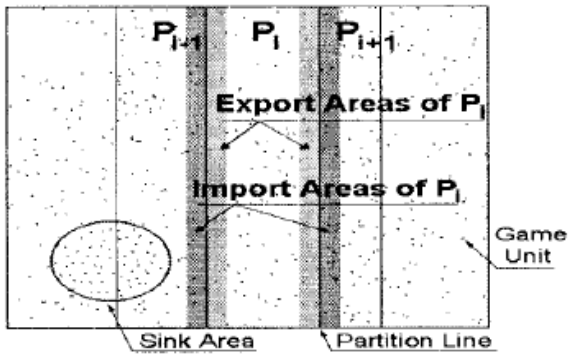
[Lui et al. 2002]

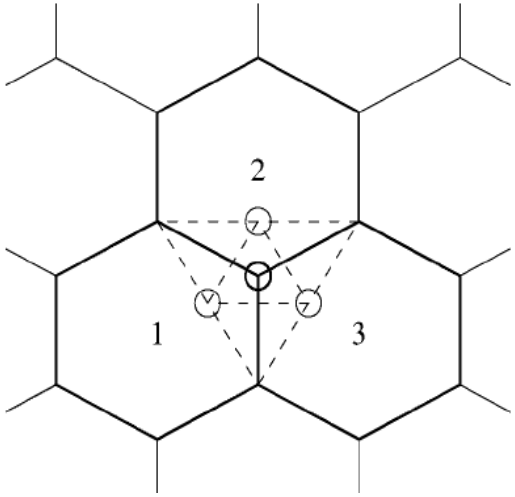


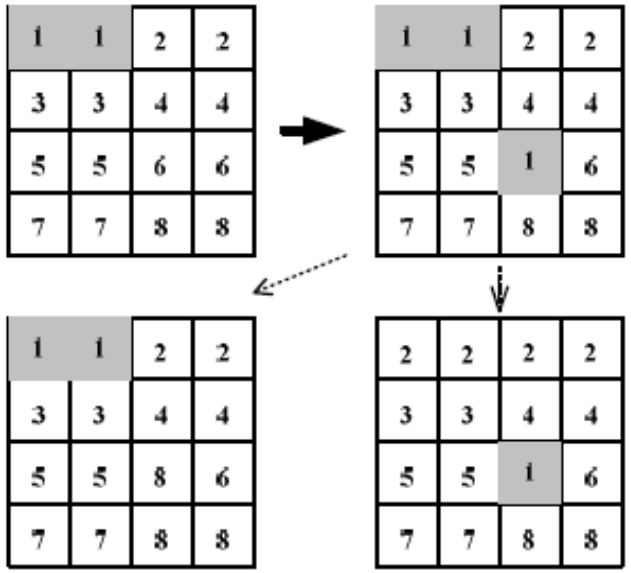
[Lu et al. 2006]

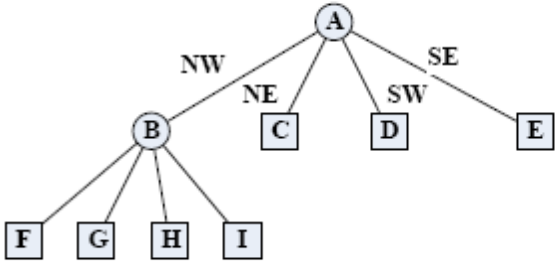
MMOG server clusters (3/3)

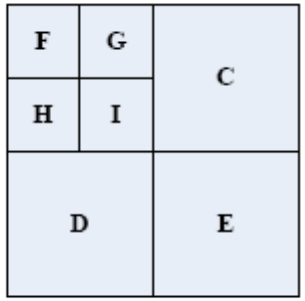
- Zone-based











ACN Laboratory

MMOG server cluster issues

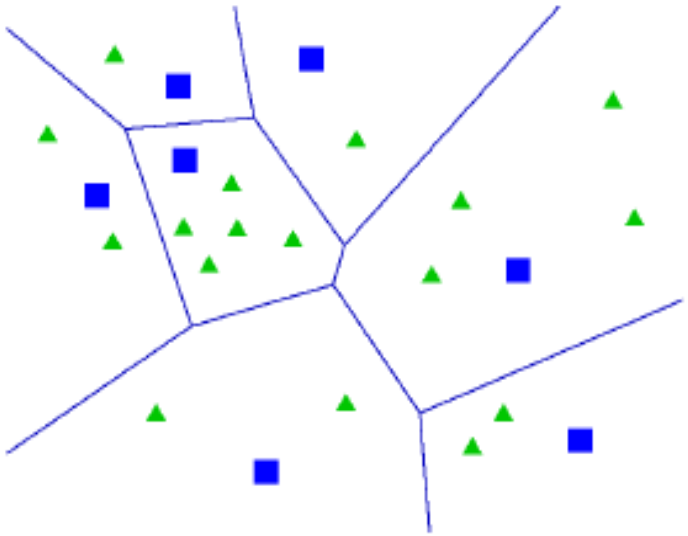
- Partitioning (static, dynamic)
- Load balancing (global vs. local)

- Main trade-off:
  - computation load vs. inter-server communication

- Main limitations:
  - Scalability (limited total resources)
  - Load balancing (high user density *hotspots*)

Voronoi State Management

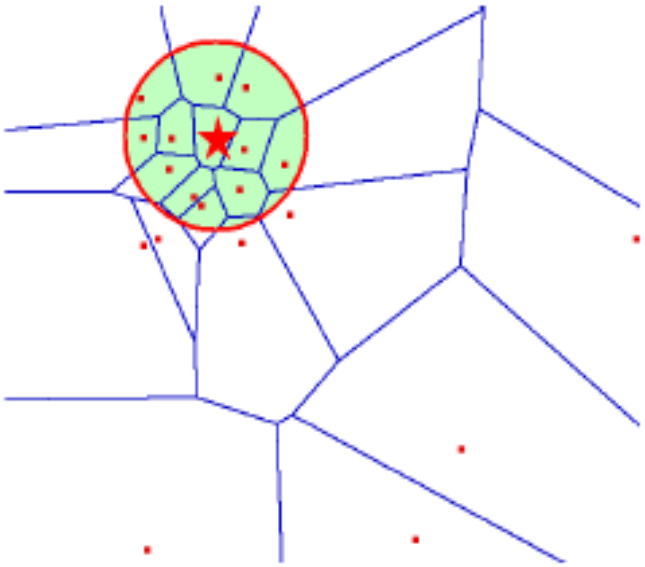
- Assumption: states stored in objects with (x,y)
- Initial idea:
  - Let game states be managed by *all* clients
  - Two roles for each client: *peers* & *arbitrators*
  - i.e. Voronoi partitioning

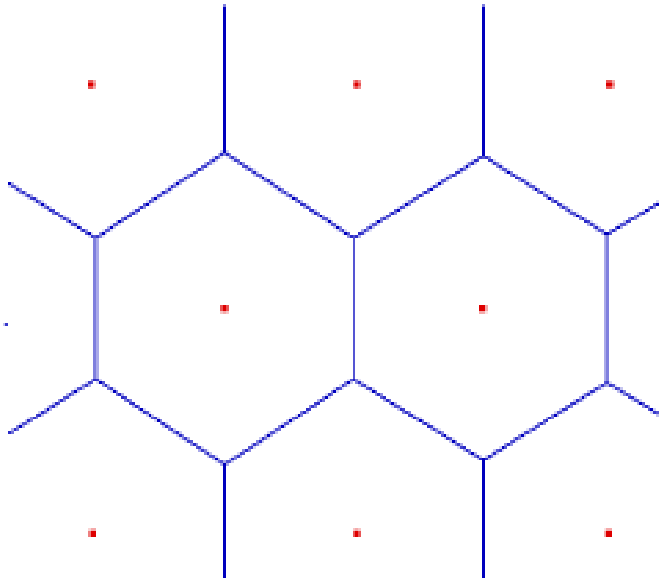


- Three problems:
  - $O(n^2)$ connections at hotspots
  - Some cells have large sizes
  - Constant ownership transfer

VSM: basic ideas

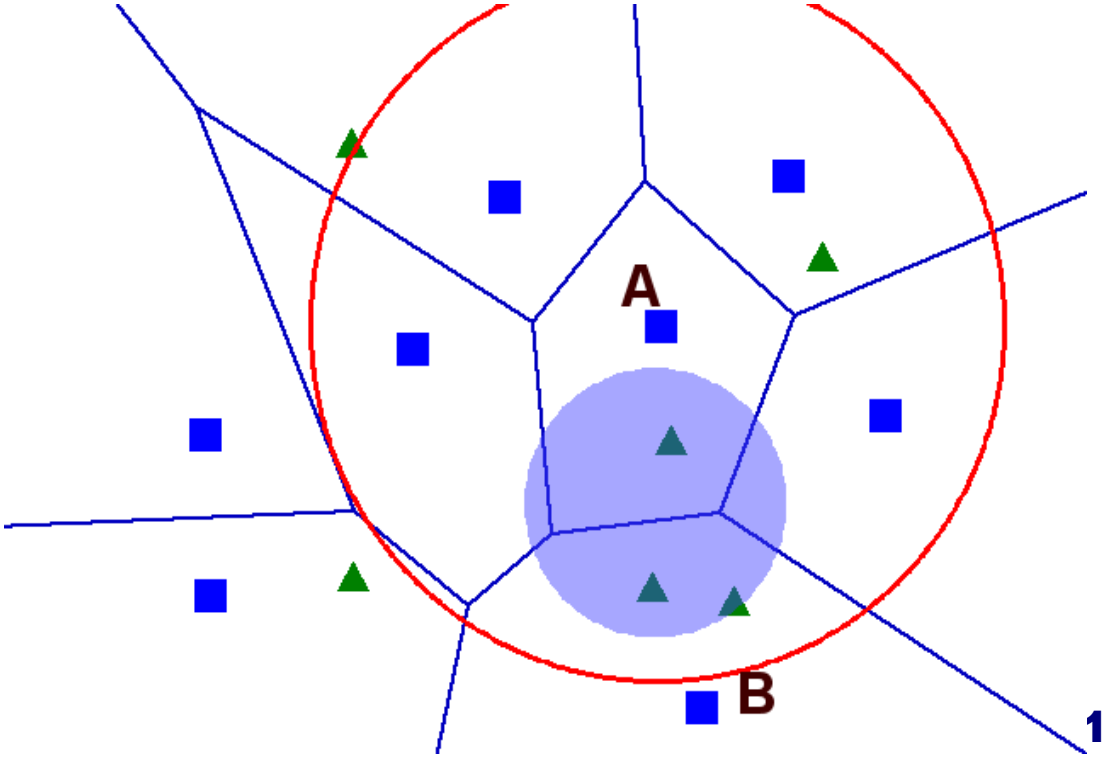
- Connection overload → *Aggregators* clustering
- Large cell-size → *Virtual peers*
- Constant transfers → Explicit ownership transfer





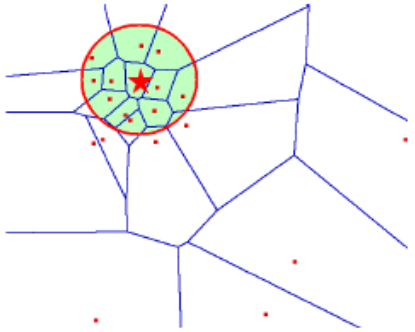
VSM: Consistency control

- *managing arbitrator* receives and processes events
- Events are forwarded if necessary
- Resulting updates are sent to affected arbitrators



VSM: Load balancing

- Traditional: high-capacity nodes first, then adjust
- VSM: low-capacity nodes first, then cluster

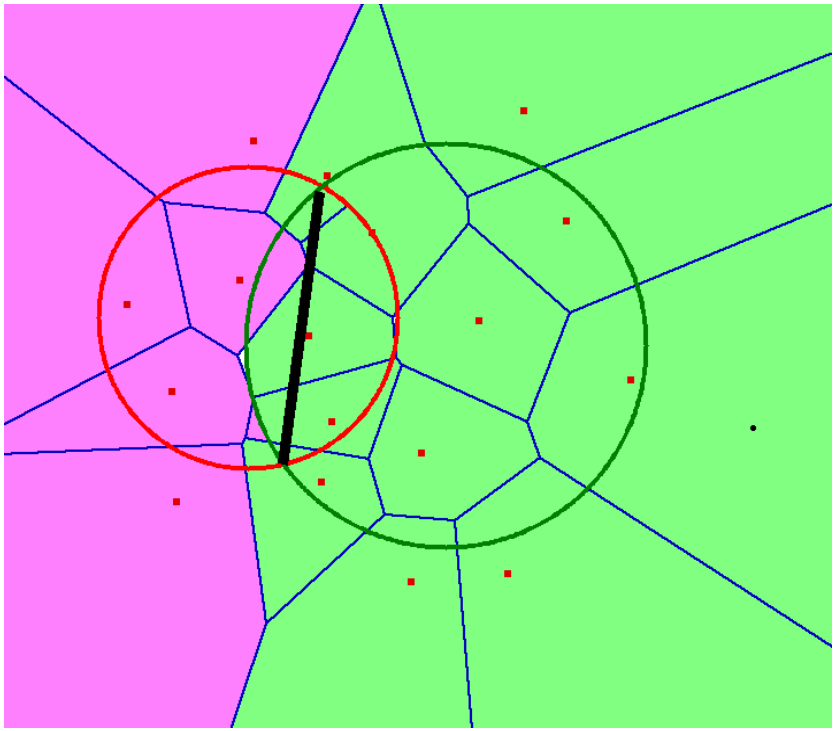


- Assume known load detections
- *Overload, underload* defined

- Overload: ask gateway for *aggregator*, submit control
- Underload: disintegrate, release control

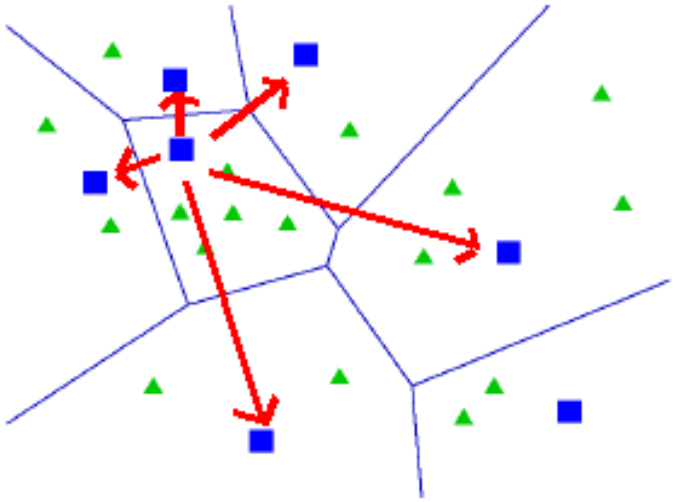
VSM: Load balancing (2)

- *Sphere of control* adjustable
- More than one aggregator → choose nearest



VSM: Fault tolerance

- Regular arbitrator:
  - Pick *backup arbitrator*, backup states
  - Backup transfers ownership to *enclosing arbitrators*
- Aggregators:
  - Pick *backup aggregators*
  - Take over original if failed
  - Choose new backup



Discussions

- Consistency → existing update-based
- Responsiveness → most events in 2 to 3 hops
- Load balancing → dynamic aggregation
- Reliability → backup nodes

- Persistency
- Security

Conclusion

- VSM utilizes
  - Voronoi partitioning
  - Existing consistency control
  - Clustering & superpeers (heterogeneity)
  - Backup nodes (churn)
- Future ideas
  - Aggregators move with nodes
  - Separate management of dynamic / static objects

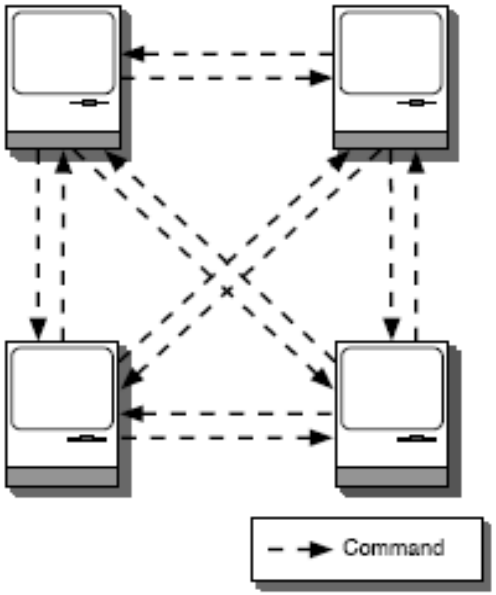
VSM: Consistency control

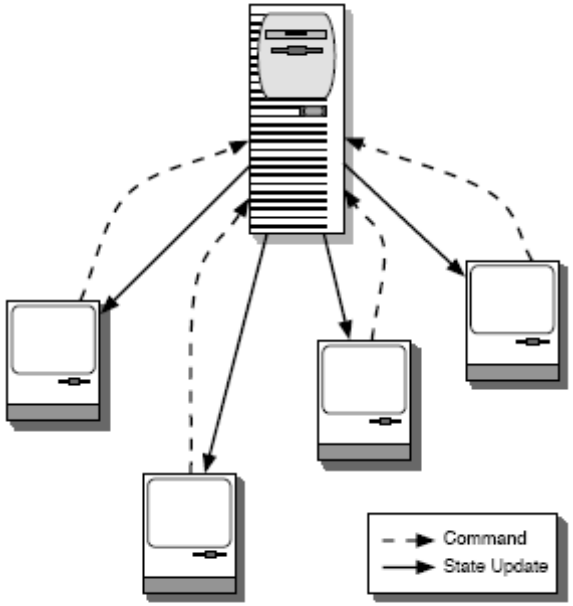
- Update-based
- Events sent to *managing arbitrator*
- *managing arbitrator* decides whether to forward
- Each arbitrator makes own decisions
- Send updates

- Attribute-level locks for transaction update
  - A arbitrator notifies B arbitrator (get lock)
  - A modifies states
  - B modifies states, release lock
  - A receives confirm, transaction done

Networking Models for Games

Point-to-Point vs. Client-server





ex. RTS

ex. FPS, MMOG

[Cronin et al. 2002]

Consistency Models for Games

- Event-based (often with point-to-point)
  - *Events* sent to *all* nodes
  - Nodes advance logical time together
  - Same states + same event executions

- Update-based (often with client-server)
  - *Events* sent to server node *only*
  - Server advances logical time
  - Server states + client synchronization via *updates*