MAX

Advanced P2P with RTMFP: Tips and Tricks

Michael Thornburgh | Senior Computer Scientist

ballerina.addEventListener(Ballerina.ADAGIO,positionHandler)
}
Fl
Ps
Dw
CSS
CF
ID
import fl.transitions.*;
import fl.transitions.easing.*;
import flash.events.*;

RTMFP Refresher

- "`rtmfp:`" vs "`rtmp`:"
  - Add an "`f`"
- NetConnection, NetStream, NetGroup
- Client-server, 1:1 P2P, group P2P, IP multicast
- UDP vs TCP
  - But always congestion controlled!
- Full and partial reliability
  - Avoid retransmission to keep latency down
- Prioritization for real-time communication
  - Requires congestion control

RTMFP Refresher

- Communication requires a NetConnection
  - `var netConnection:NetConnection = new NetConnection;`
  - …
  - `netConnection.connect(“rtmfp://server.example.com/...”, ...);`
- RTMFP server performs P2P introduction, NAT/firewall traversal
- Communication (including P2P) halts if NetConnection is closed or connection to the server is lost
- Server options
  - FMES4+, on premise or hosted on AWS EC2 by the hour
  - Codename Cirrus service: http://labs.adobe.com/technologies/cirrus/

RTMFP Refresher – P2P modes

- One-to-one direct connections
  - Publish:
    - `var ns:NetStream = new NetStream(netConnection, NetStream.DIRECT_CONNECTIONS);`
    - `ns.publish(“stream”);`
  - Play:
    - `var ns:NetStream = new NetStream(netConnection, peerID);`
    - `ns.play(“stream”);`
- Peer ID is a unique 256-bit cryptographic pseudorandom identifier
  - Locally derived by each peer, not assigned by server
    - `netConnection.nearID`
  - Disappears when NetConnection closes

RTMFP Refresher – P2P modes

- Groups
  - Group specification string (“groupspec”)
    - Start with “G:” to distinguish from peer IDs
    - `GroupSpecifier` class helps you make groupspecs
  - Join
    - `var ns:NetStream = new NetStream(netConnection, groupspec);`
    - `var ng:NetGroup = new NetGroup(netConnection, groupspec);`
  - P2P permission dialog
    - Groups can use upload bandwidth for other members’ traffic
    - Must be accepted before groups will function
  - NetConnection with or without server

RTMFP Refresher – P2P Groups modes

- Multicast streaming (P2P, native IP, "fusion")
- Posting
- Directed Routing
- Object Replication

RTMFP Refresher – P2P Groups modes

- Multicast streaming (P2P, native IP, “fusion”)
  - `netStream.publish(“stream”);`
  - `netStream.play(“stream”);`
  - `NetGroup.MulticastStream.PublishNotify/.UnpublishNotify` events
- Posting
- Directed Routing
- Object Replication

RTMFP Refresher – P2P Groups modes

- Multicast streaming (P2P, native IP, “fusion”)
- Posting
  - `netGroup.post(object)`
  - `NetGroup.Posting.Notify` event
- Directed Routing
- Object Replication

RTMFP Refresher – P2P Groups modes

- Multicast streaming (P2P, native IP, "fusion")
- Posting
- Directed Routing
  - `netGroup.sendToNearest()/sendToNeighbor()/sendToAllNeighbors()`
  - `netGroup.receiveMode`
    - `NetGroupReceiveMode.EXACT` (default)
    - `NetGroupReceiveMode.NEAREST`
  - `NetGroup.SendTo.Notify` event
- Object Replication

RTMFP Refresher – P2P Groups modes

- Multicast streaming (P2P, native IP, "fusion")
- Posting
- Directed Routing
- Object Replication
  - `netGroup.addHaveObjects()/addWantObjects()`
  - `netGroup.removeHaveObjects()/removeWantObjects()`
  - `netGroup.writeRequestedObject()`
  - `netGroup.replicationStrategy` (lowest first, rarest first)
  - Events
    - `NetGroup.Replication.Request`
    - `NetGroup.Replication.Fetch.Result`
    - `NetGroup.Replication.Fetch.SendNotify`
    - `NetGroup.Replication.Fetch.Failed`

RTMFP Refresher

CONNECT. DISCOVER. INSPIRE.

- For a detailed review, watch Matthew’s sessions from MAX 2008 & 2009

  http://tv.adobe.com/watch/max-2008-develop/future-of-communication-with-rtmfp-by-matthew-kaufman/

  http://tv.adobe.com/watch/max-2009-develop/p2p-on-the-flash-platform-with-rtmfp/

Serverless Mode

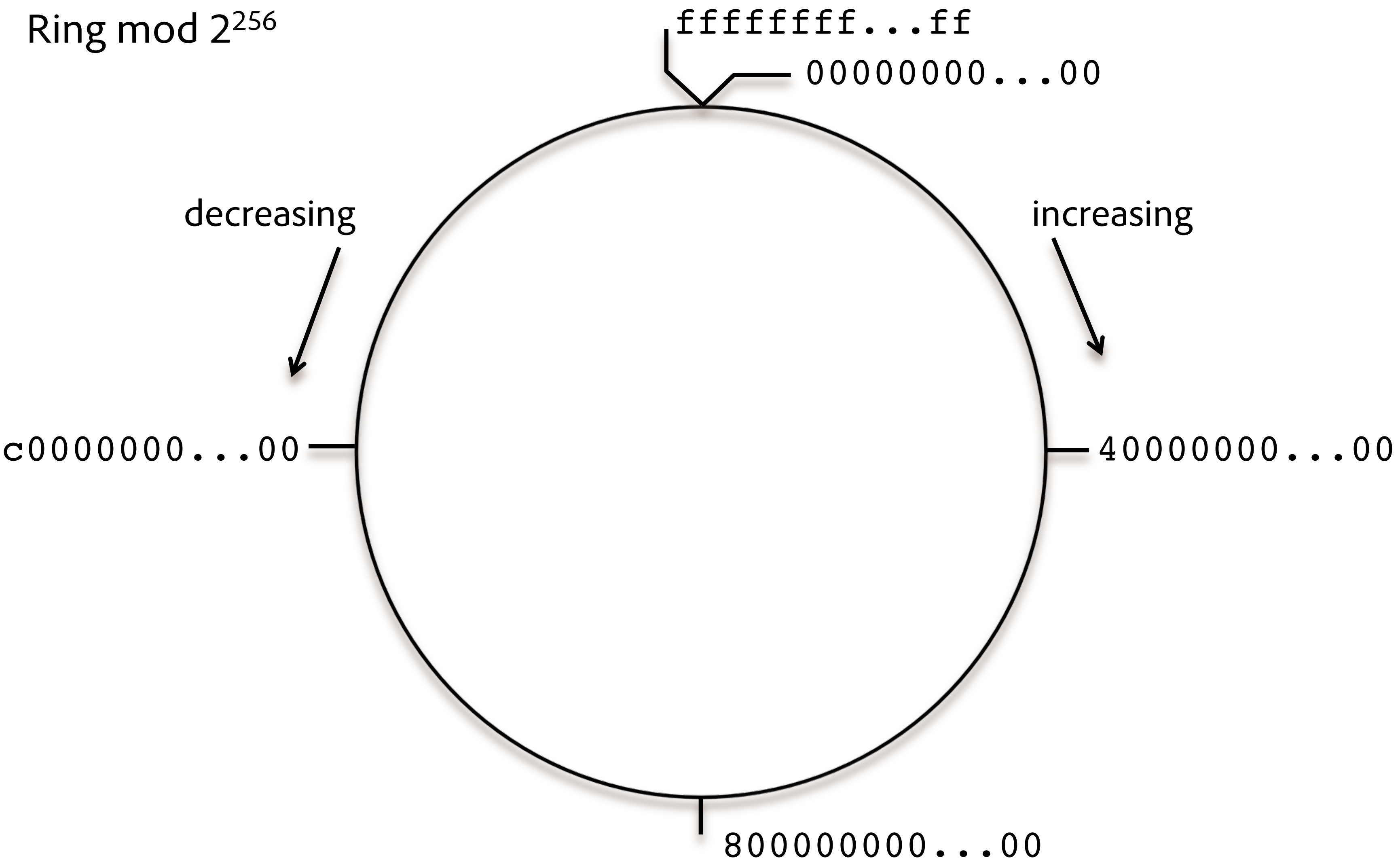
- `NetConnection.connect(“rtmfp:”)`
  - also used for receiving pure IP multicast streams
    - not talking about that today (not P2P ;)
- For Groups
  - Can use 1:1 NetStreams if a low-level session is already up (via groups)
    - as of Flash Player 11.0
    - use a 2-member group to guarantee
- Must use IP multicast to discover/establish peers
  - `GroupSpecifier.ipMulticastMemberUpdatesEnabled = true`
  - `GroupSpecifier.addIPMulticastAddress()`
  - LAN only (TTL/hop limit 1)

Group Topology

- Group address is position on ring mod 2^{256}
  - SHA256 hash of peer ID (and other stuff)
    - and peer ID is itself a SHA256 hash of a cryptographic public key (and other stuff)
    - can’t directly choose or influence
    - evenly distributed
      - `NetGroup.estimatedMemberCount`
    - `NetGroup.convertPeerIDToGroupAddress()`
  - represented as 64-digit hex strings (256 bits),
    `0000000000000000000000000000000000000000000000000000000000000000` through
    `ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`

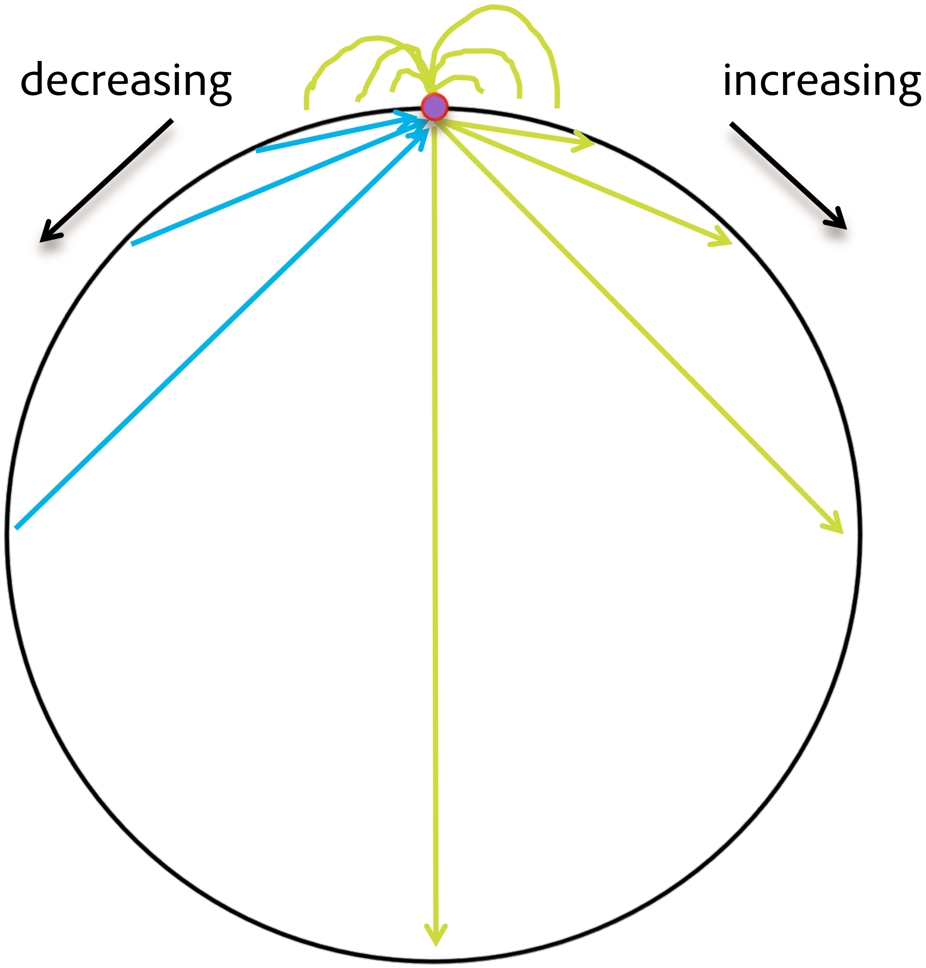
Group Topology

- Ring mod 2^{256}



Group Topology

- 3 immediate increasing
- 3 immediate decreasing
  - maintain the ring, full transitive connectivity
- Binary fractionation ½, ¼ …
- 6 low latency (lowest RTT)
- Binary fractionation target from other side
- 1 random every 10(ish) seconds



Group Topology

- Full mesh guaranteed* for groups 14 or smaller
  - 3 (inc) + 3 (dec) + 6 (fast) + 1 (rand) + 1 (self)
  - * NATs or firewalls may block some neighbor connections
    - usually not an issue for LAN communication
  - *High churn may delay convergence
    - Stale peer records may still seem worth trying for a while
    - Stale records purged after 5 minutes

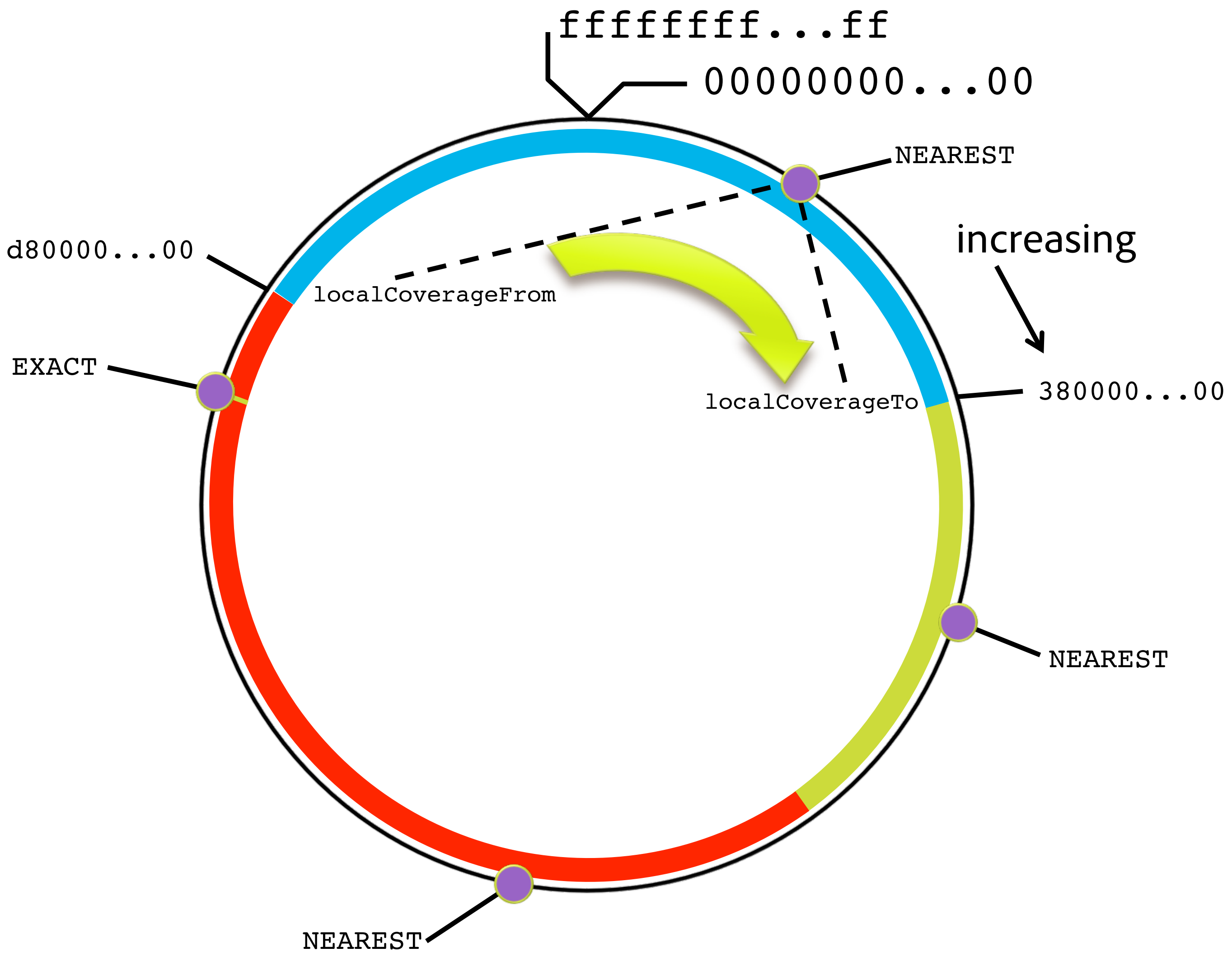
Directed Routing

- `GroupSpecifier.routingEnabled = true`
- Send a message directly to one or more peers in a group
- Reliable, in-order delivery mode

Local Coverage

- `NetGroup.receiveMode`
  - `EXACT` (default): `localCoverageFrom` = `localCoverageTo` = group address
  - `NEAREST`: `localCoverageFrom` to `localCoverageTo` inclusive
    - measured in increasing direction on ring
    - depends on your neighbors' addresses and receive modes
    - coverage ranges for `NEAREST`-mode peers in a properly connected group are contiguous and non-overlapping
    - `EXACT` peers may be point holes in other peers' `NEAREST` ranges

Local Coverage



Automatic Distributed Election

- Consistently select O(1) subset of members for special responsibilities
  - Example: Retrieve file from server and seed via Object Replication
- Local coverage can be used to implement
  - Example: peer covering "`0000...00`" is "elected"
  - Easy to check with string comparisons, 256-bit math not needed ☺
- Gotcha when electing more than one peer:
  - Postings must have unique byte serialization
  - Consistent serialization will avoid duplicates in the network
    - Use `Array`, avoid `Object`

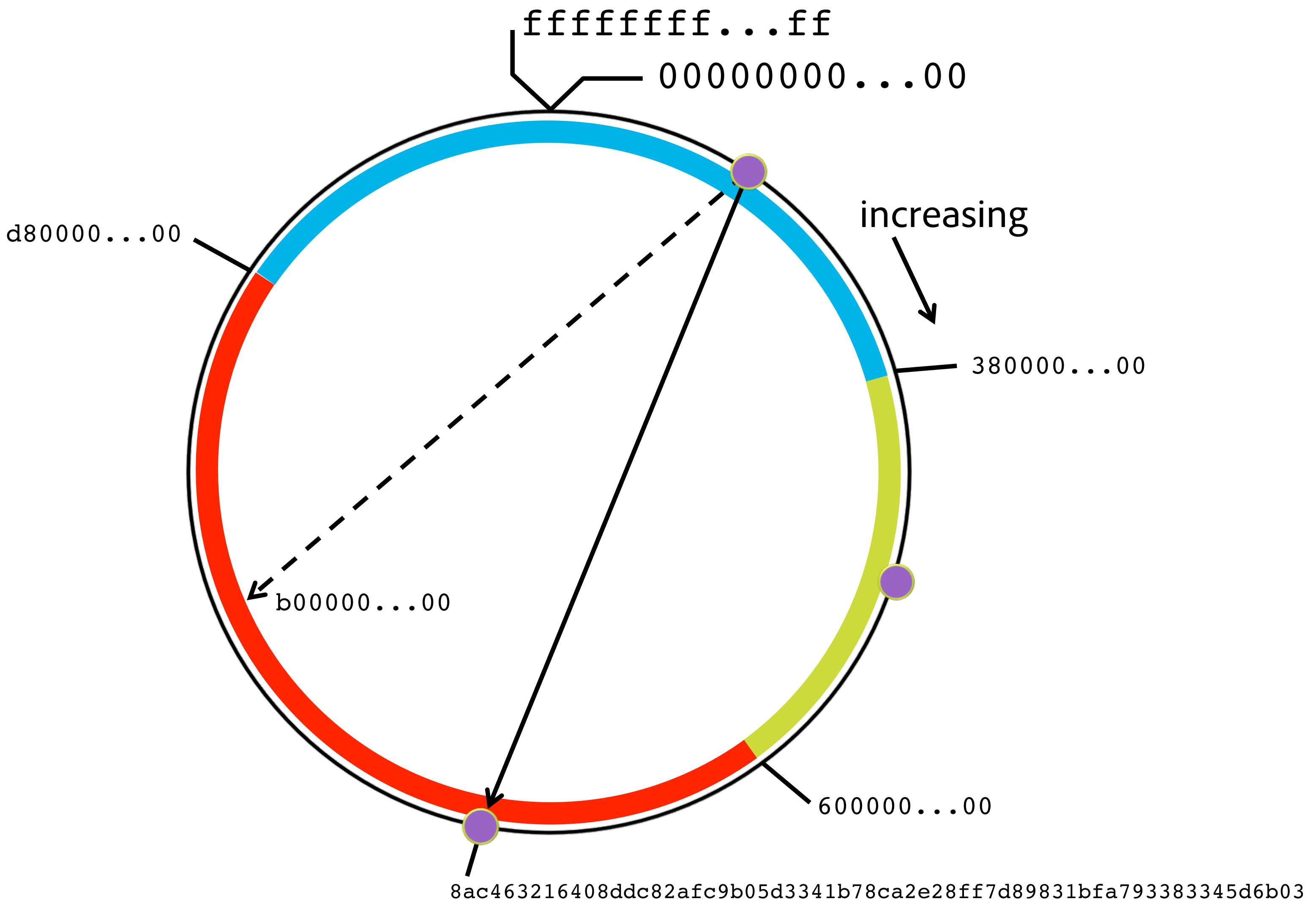
Automatic Distributed Election

DEMO

Send to Nearest

- `NetGroup.sendToNearest()` sends a message one hop to the **neighbor** (or self) numerically nearest to the target group address
  - `EXACT` peers are only nearest to their own group address
  - Each peer is nearest to its own group address
  - Recursive operation can route a message through the group hop-by-hop to the node that is actually closest
  - If target address is in self's local coverage, message is sent to self with `fromLocal=true`
  - Can build a DHT this way
    - Worst-case diameter is $\log_4 N$
    - Often better

Send To Nearest



Send To Nearest

- Too many `EXACT` peers can also disrupt `NEAREST` routing
  - Ring-adjacent `NEAREST` peers must be neighbors with each other to calculate consistent non-overlapping local coverage ranges
  - A peer may over-estimate its local coverage, causing a message to not be routed to the actual nearest peer

Send To Neighbor

- `NetGroup.sendToNeighbor()` sends a message along the ring
  - `NetGroupSendMode.NEXT_INCREASING, .NEXT_DECREASING`
- Correct topology means self→`NEXT_INCREASING`→`NEXT_DECREASING` == self→`NEXT_DECREASING`→`NEXT_INCREASING` == self
  - Could traverse entire group hop-by-hop
- Incorrect topology means spurs in the ring
  - Potentially inconsistent recursive routing behavior
  - Usually caused by NAT or firewalls blocking a neighbor connection

Send To Neighbor

- Adjacency scales as O(1)
- Convenient when you need someone else to do something for you
  - Example: announce when you leave the group, since you can't do it
- When all members' receive modes are `NEAREST`, `NetGroup.LocalCoverage.Notify` event can indicate an adjacent neighbor has changed
  - `localCoverageTo` changing indicates `NEXT_INCREASING` is different
  - `localCoverageFrom` changing indicates `NEXT_DECREASING` is different

Send To Neighbor

CONNECT. DISCOVER. INSPIRE.

DEMO

Send To All Neighbors

- `NetGroup.sendToAllNeighbors()`
  - Sends to all directly connected neighbors, not to every member of the group
    - That's what `NetGroup.post()` is for
  - Sends to all neighbors at the same time
    - Could jam network if you have a lot of neighbors

- Demo shortly… ☺

Small Groups

- Small groups are fully meshed
  - Each member is a neighbor with every other member
    - Diameter 1
  - 14 or fewer members guarantees* a full mesh (see topology slides)
  - `NetGroup.sendToAllNeighbors()` reaches all* members directly
    - Much lower latency than posting
    - Load not shared by other peers
    - Lower overhead than posting for non-trivial messages
    - Ideal for N:N or where latency is important (like Real Time games)

Small Groups

- Neighbor disconnect means the peer is really gone
- Smallest useful group is 2 members
  - As of Flash Player 11.0, can use 1:1 `DIRECT_CONNECTIONS` NetStream even in serverless mode
- NetGroup.sendToNearest() to exact address for easy 1:1 messaging

Large Groups

- Large groups are not fully meshed
  - Each member will not be a direct neighbor with every other member
  - Probability of stable full mesh drops rapidly above 14 members
    - Very unlikely above about N=20, probability 0 at N=33
- `NetGroup.sendToAllNeighbors()` doesn't reach all members
- Use posting for N:N communication or for infrequent short messages few:N
  - Higher latency, possibly seconds or more to reach all members of a large group
  - Members share load
  - Will reach all members even when NAT/firewalls break correct routing topology

Large and Small Groups

CONNECT. DISCOVER. INSPIRE.

DEMO

Large Groups

- Useful as a lobby or discovery area, break-out to “small” groups
- Use `NetGroup.sendToAllNeighbors()` to reach O(log n) randomish subset
- Use multicast with NetStream.send() for continuous data from 1 (or a few) to all members
  - Possible jitter up to multicast window duration
  - Even a stable group takes tens of seconds to converge to low-latency “push” mode

Large Groups

- Recursive directed routing
  - Chances of NAT/firewall disruption of correct routing topology high
  - For DHT applications, use replication to increase chances of at least one working path
  - Routing diameter (worst case) $\log_4 N$, usually better
- Gossip approach for efficient (but potentially slow) information diffusion
  - Distributed presence
  - Diffusion diameter O(log N)
- Flooding diameter may approach O(log log N)

All Groups

- Full transitive connectivity regardless of size with high probability
  - Posting, flooding, gossip, object replication, multicast highly likely to reach all members

What About Multicast?

- Multicast is good at 1 (or few) –to-all distribution
  - Builds efficient spanning trees through group for low-latency push mode
  - `NetStream.send()` works in a multicast stream
- Multicast can have high delay/jitter
  - Pull mode at startup while trees are building
  - Peers joining and leaving can disrupt trees, pull mode while they rebuild
  - Tree building requires data to be flowing

What About Multicast?

- `NetStream.multicastWindowDuration`
  - Limits latency and jitter
  - Lower delay may mean lower reliability (more missed messages)
    - Much lower window durations will probably also require lowering the `multicastAvailabilityUpdatePeriod` and `multicastFetchPeriod`
      - Will increase protocol overhead
    - Set parameters before `NetStream.publish()`

- N:N Streaming
  - Full mesh of `DIRECT_CONNECTIONS` NetStreams is more efficient

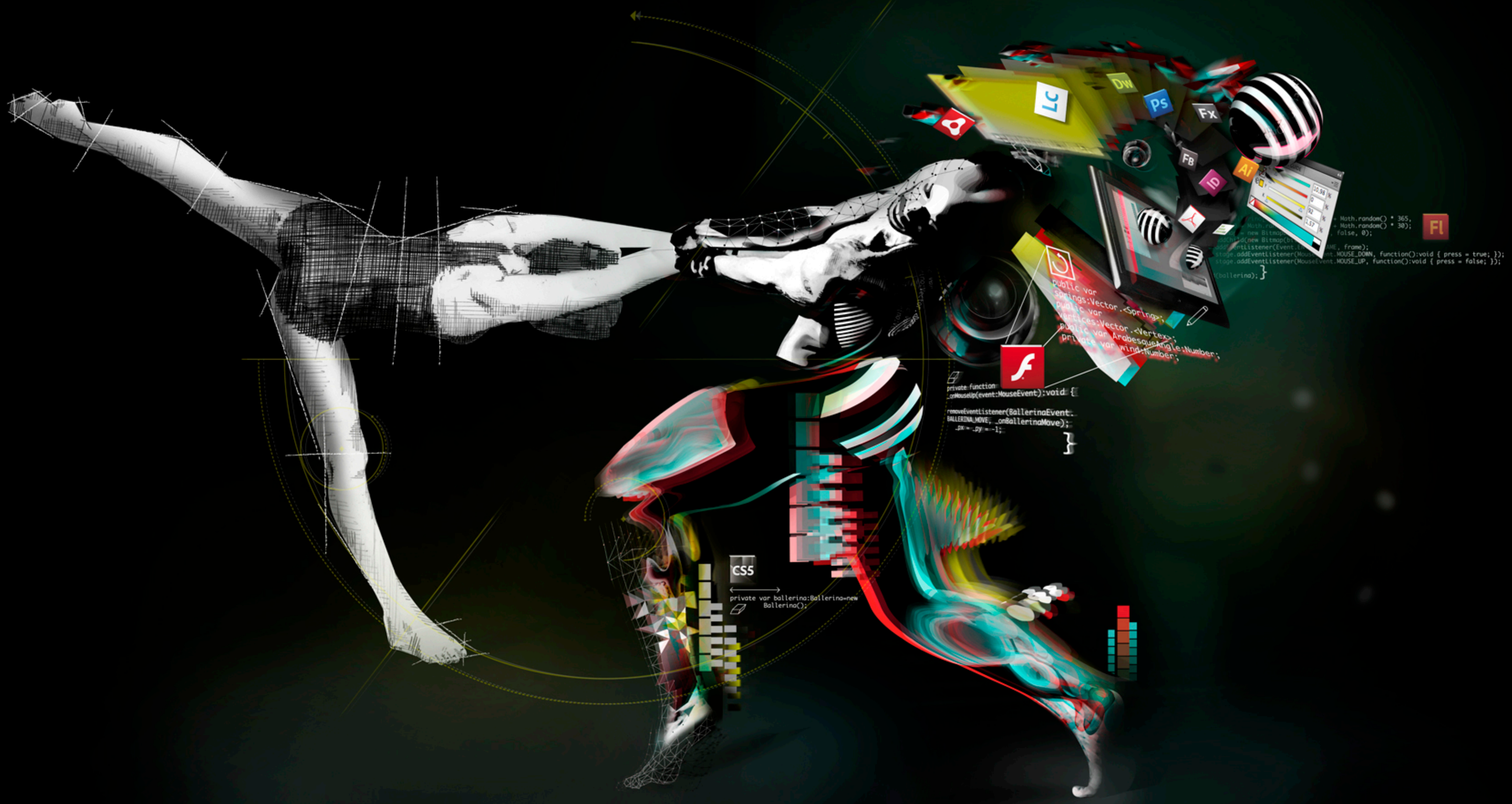
Q&A

CONNECT. DISCOVER. INSPIRE.

Q & A

MAX

CONNECT. DISCOVER. INSPIRE.



```
public var
springs:Vector.<Spring>;
public var
vertices:Vector.<Vertex>;
public var ArabesqueAngle:Number;
private var wind:Number;
```

```
private function
_onMouseUp(event:MouseEvent):void {

removeEventListener(BallerinaEvent.
BALLERINA_MOVE, _onBallerinaMove);
_px = _py = -1;
}
```

```
+ Math.random() * 365,
+ Math.random() * 30);
, false, 0);
frame);
stage.addEventListener(MouseEvent.MOUSE_DOWN, function():void { press = true; });
stage.addEventListener(MouseEvent.MOUSE_UP, function():void { press = false; });
(ballerina); }
```

```
private var ballerina:Ballerina=new
Ballerina();
```