Monitor for a Multi-tree P2P Live Streaming System

Benjamin Schiller, P2P Networks, TU Darmstadt

Network Tech Talk Series / A213 / 28.03.2012

Motivation

- Distributed systems are hard to debug
- Understanding log output is a tedious job
- P2P-based live streaming systems
  - Overlays are constructed according to certain rules
  - Adaption of the overlay only based on local knowledge
- Goal: Monitor for existing live streaming system
  - Graphical representation of the overlay topologies
  - Collect statistics and display properties of the system
  - Influence behavior of nodes to a certain extend

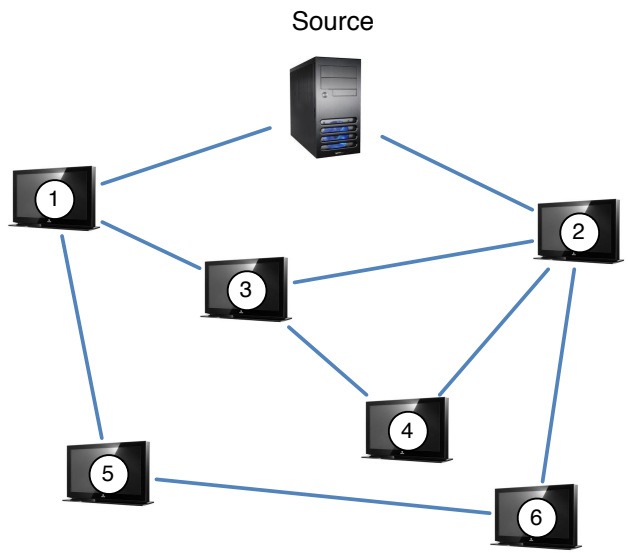
Outline

- Design of the Live Streaming System
- Implementation
- Demo
- Outlook

Design of the Live Streaming System

Considered Overlay Topologies

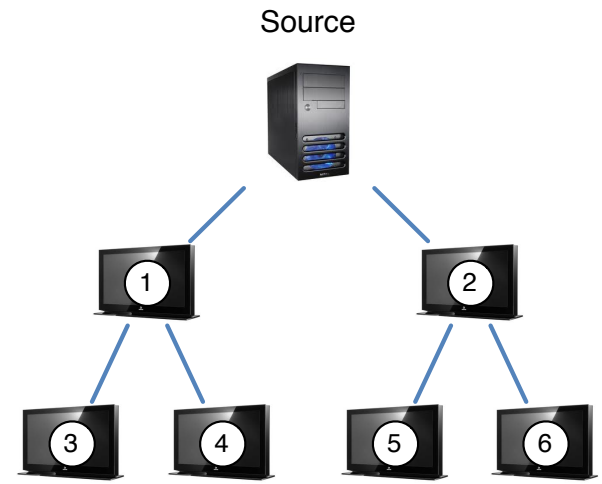
Mesh-based



(+) Many alternate routes, implicit load balancing
(+) High resilience to node failures and attacks
(+) Neighbor-selection based on different properties
(+) Good solution for VoD streaming

(-) Higher delays
(-) High maintenance overhead

Tree-based

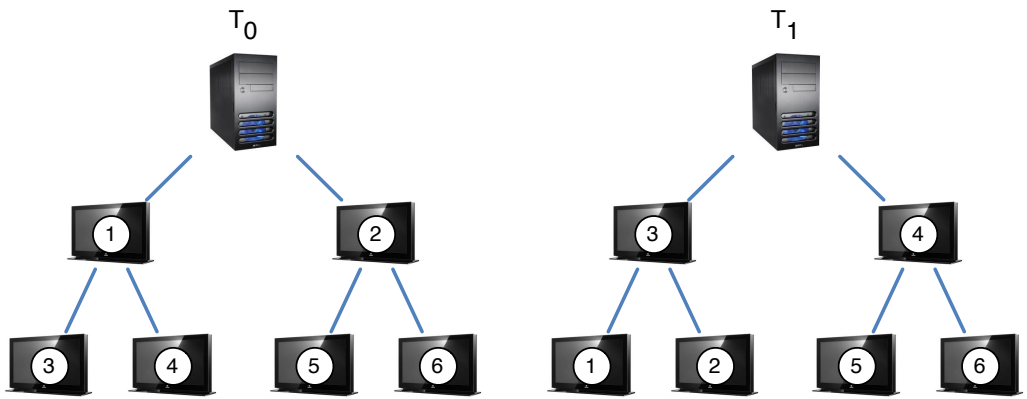


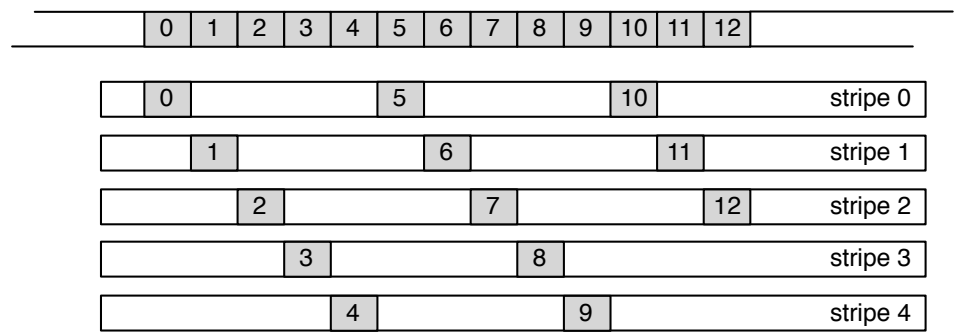
(+) Easy distribution of the media stream
(+) Simple overlay topology
(+) Good solution for live streaming

(-) Unbalanced load distribution
(-) Vulnerabilities to attacks and failures

Used Overlay Topology

Multi-tree-based

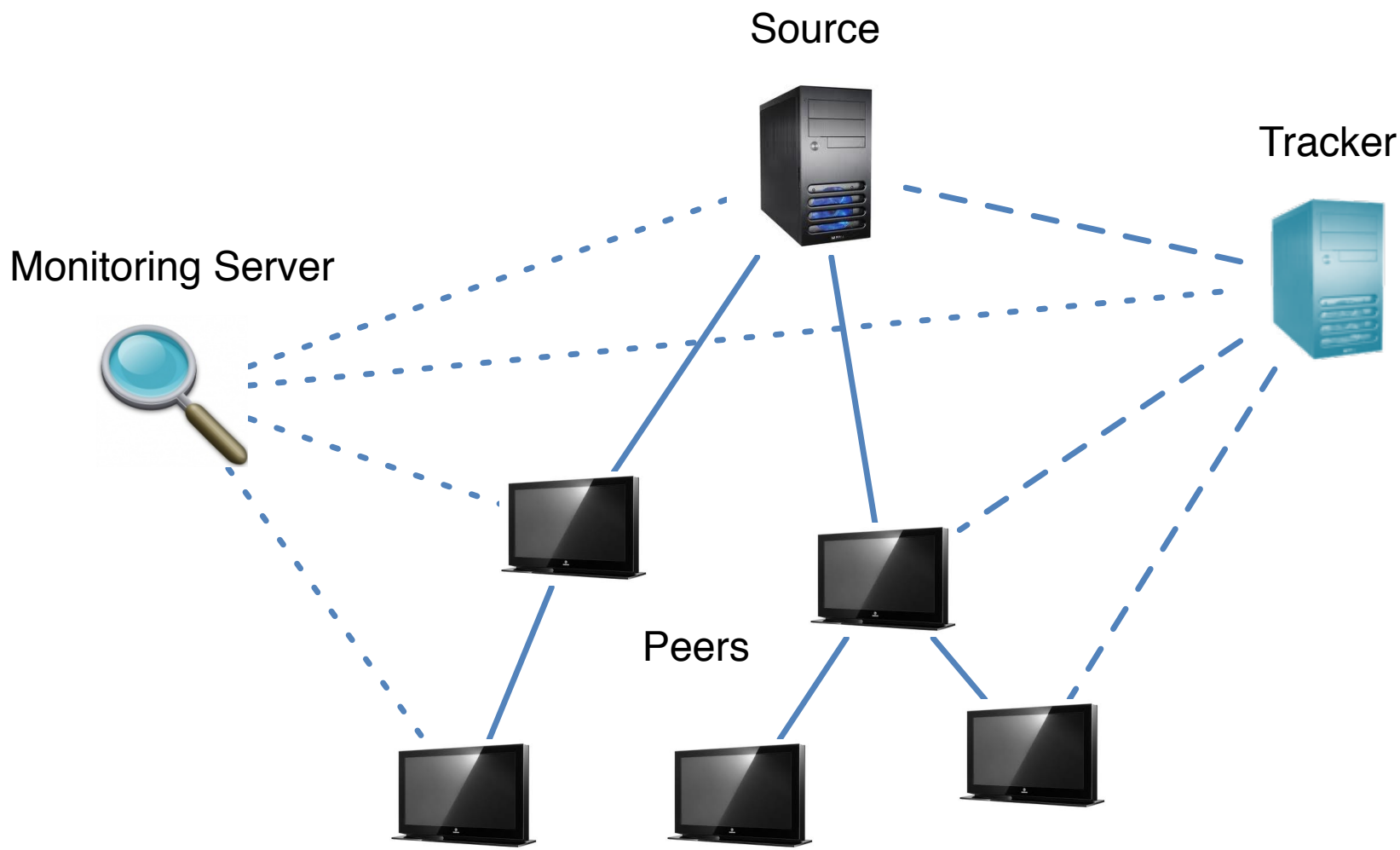




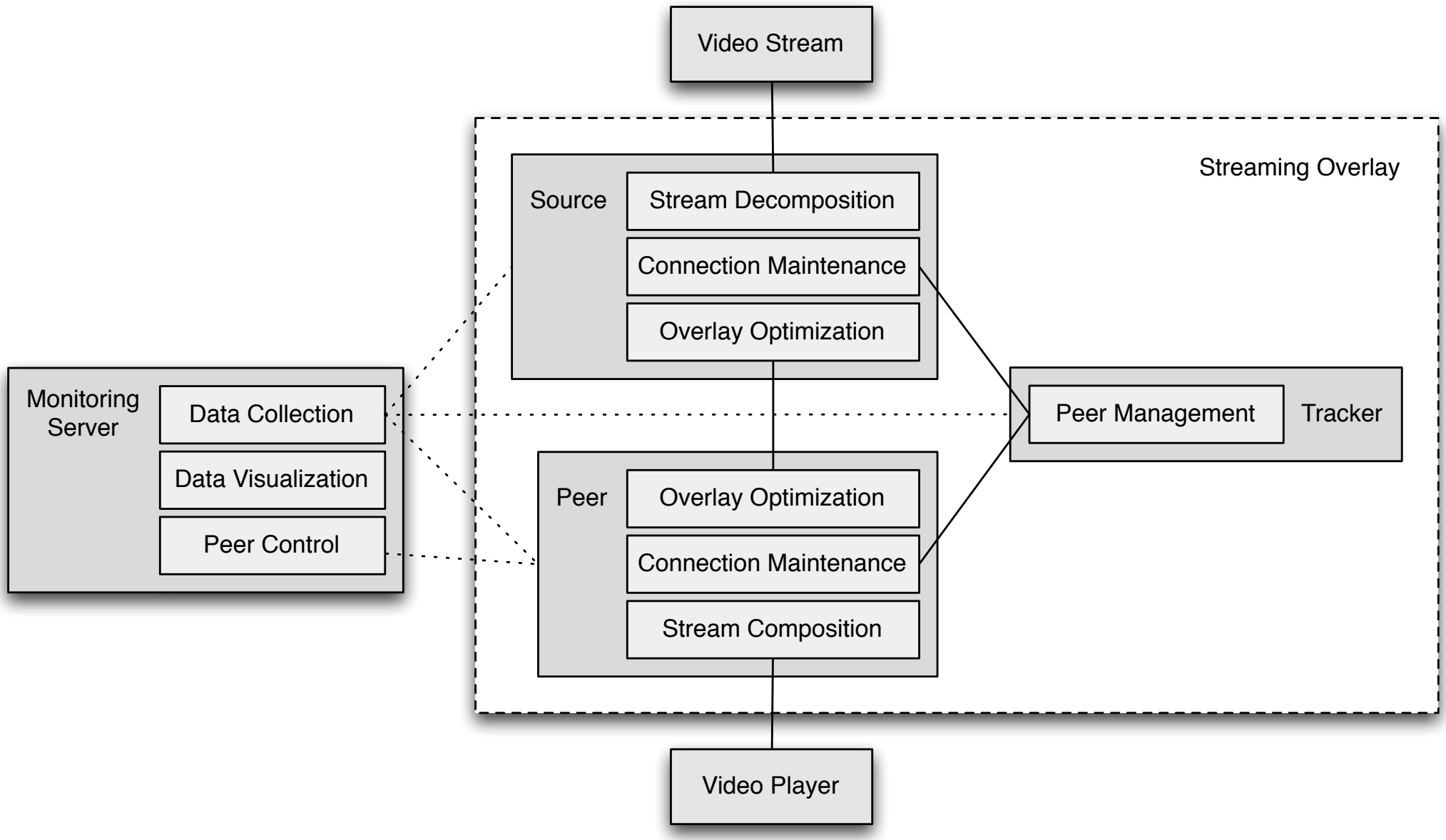
(+) Failures in single stripe can be tolerated
(+) Higher resilience against failures / attacks
(+) Better load distribution amongst end-hosts

(-) More complex structure than single-tree ALM
(-) Additional maintenance overhead

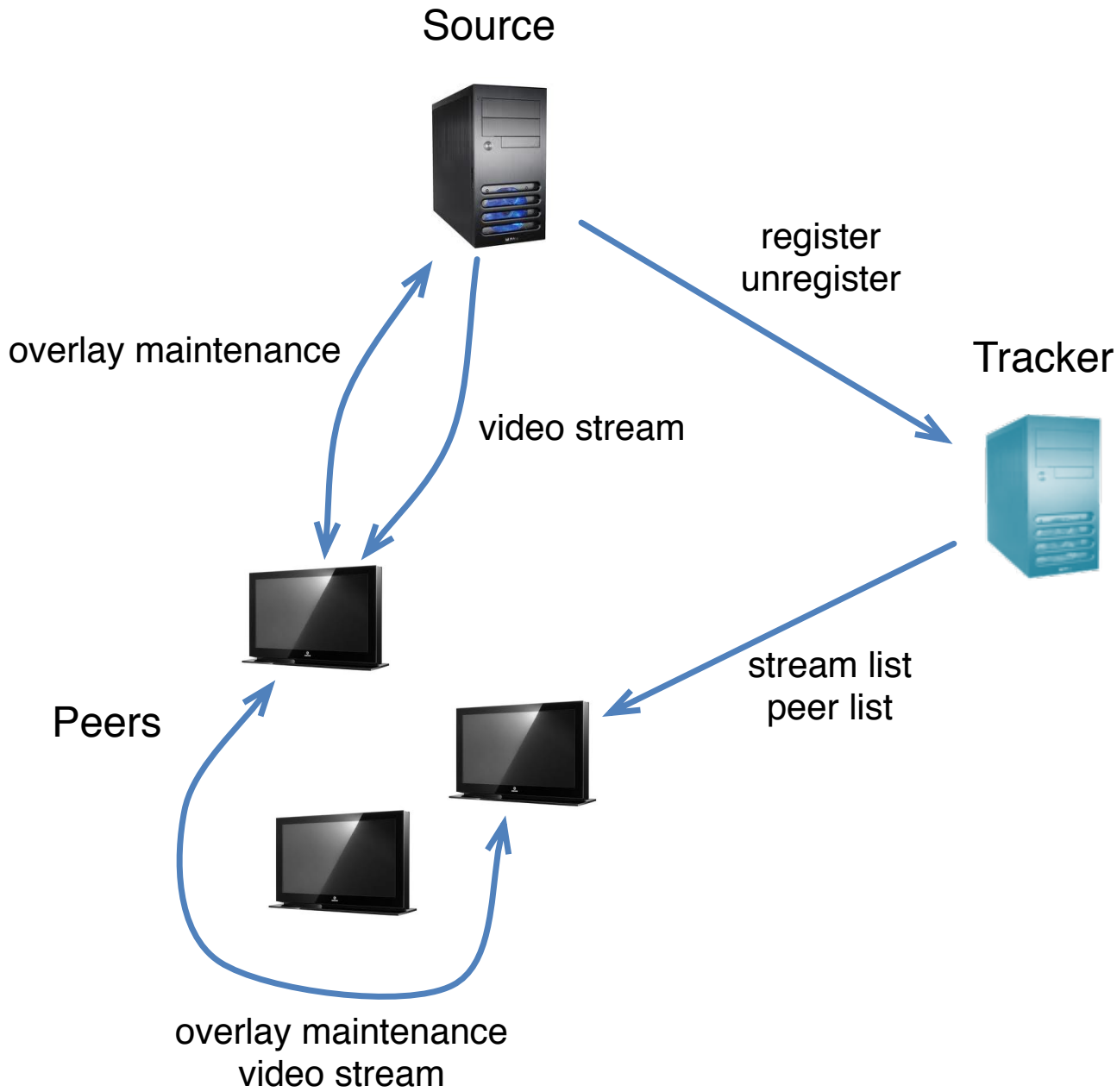
Components



Architecture



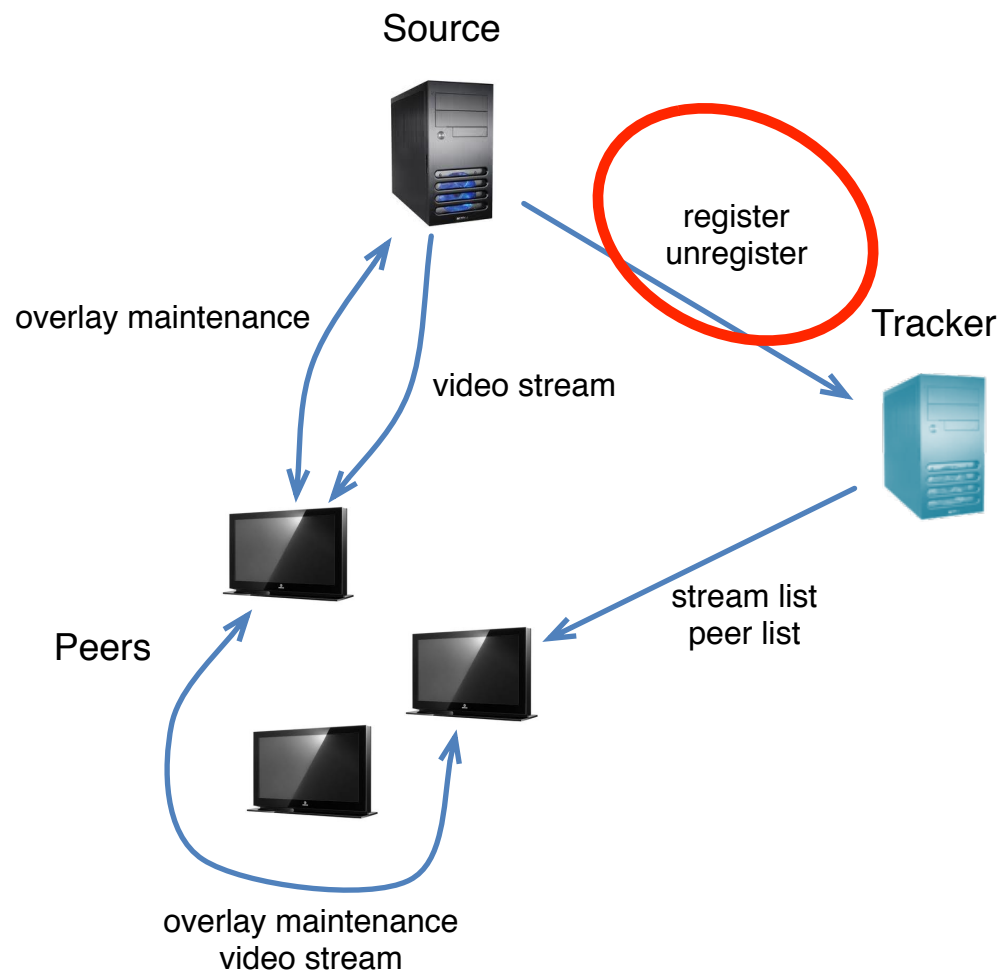
Message Flow



Source <=> Tracker

- ALIVE

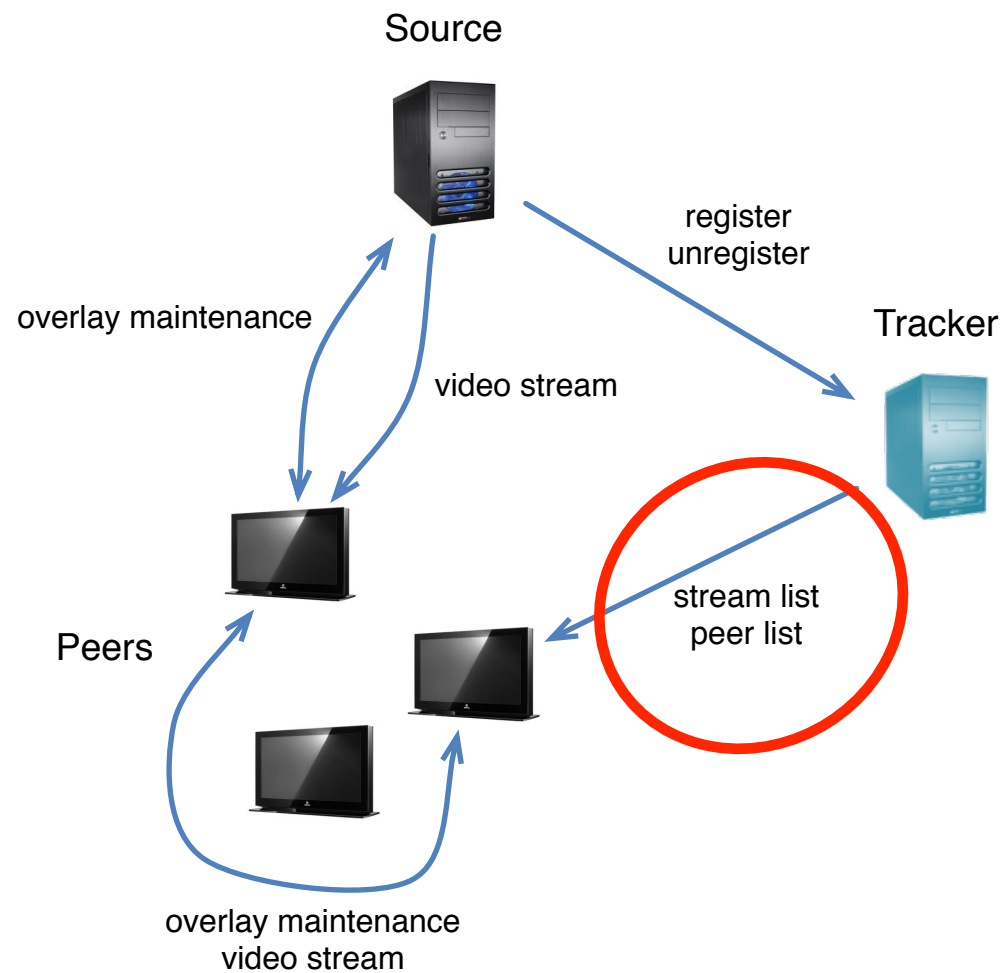
- REGISTER_STREAM
- UNREGISTER_STREAM



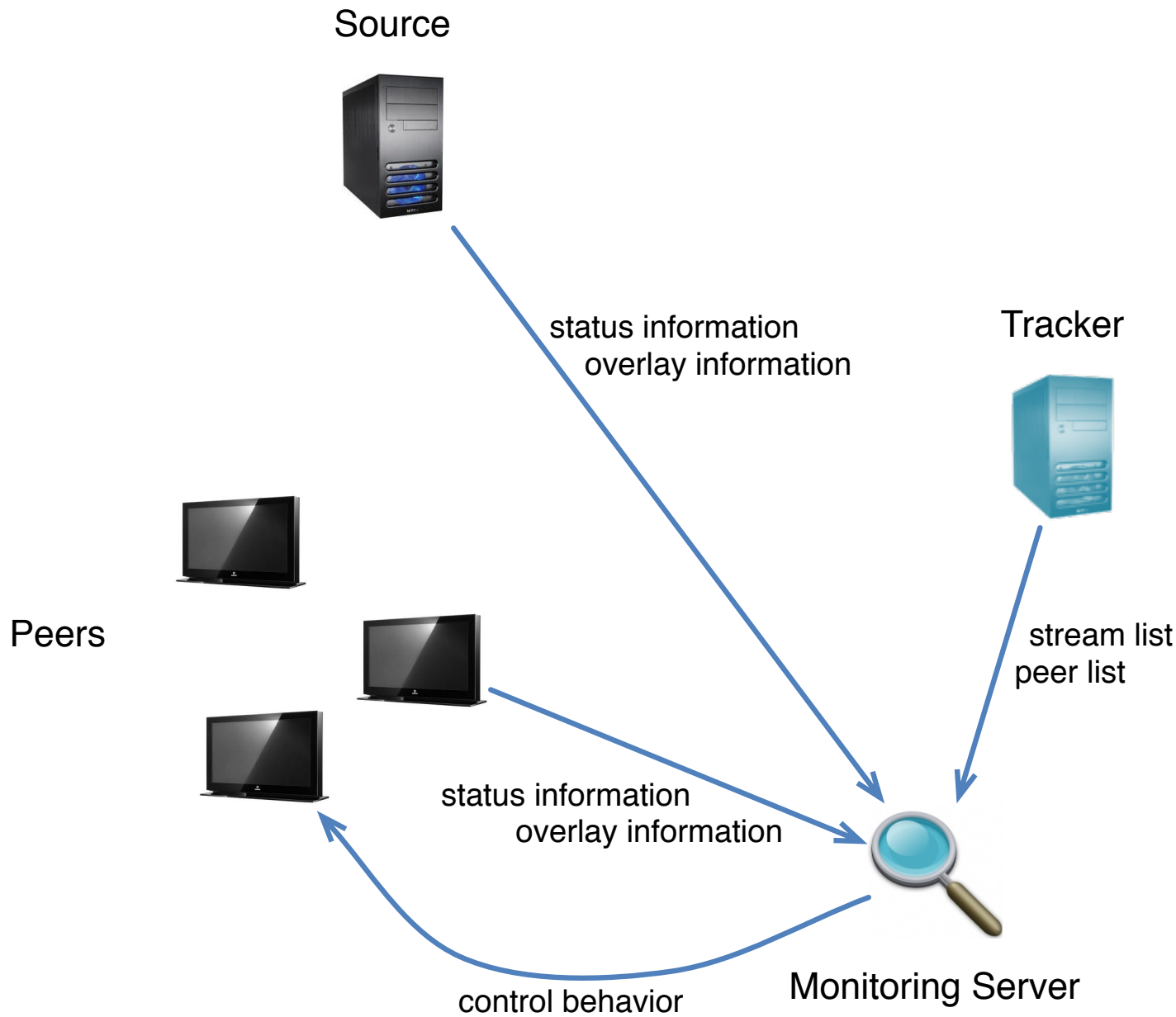
Tracker <=> Peer

- P_ALIVE
- P_LEAVE

- GET_STREAMS
- GET_PEER_LIST

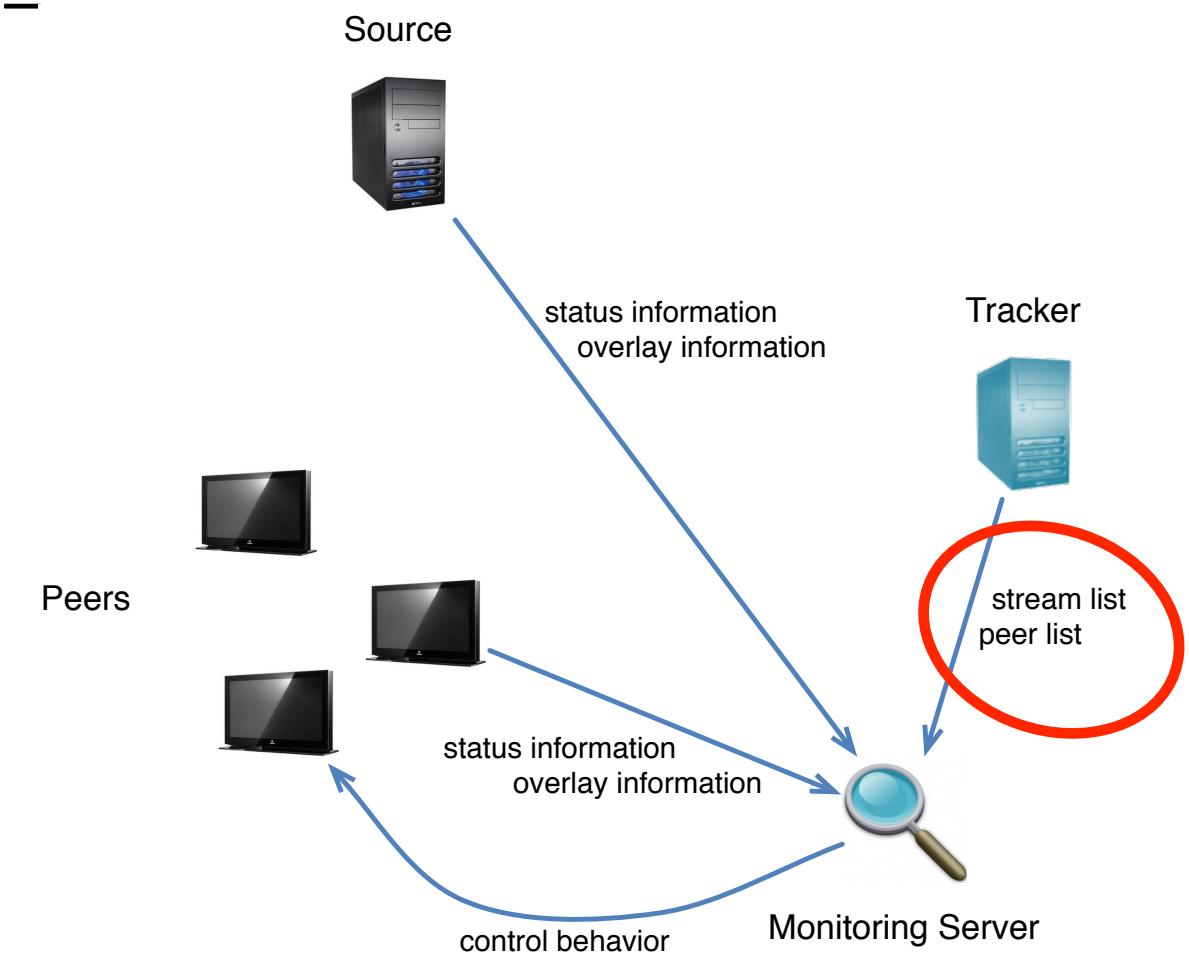


Monitoring Server



Monitor <=> Tracker

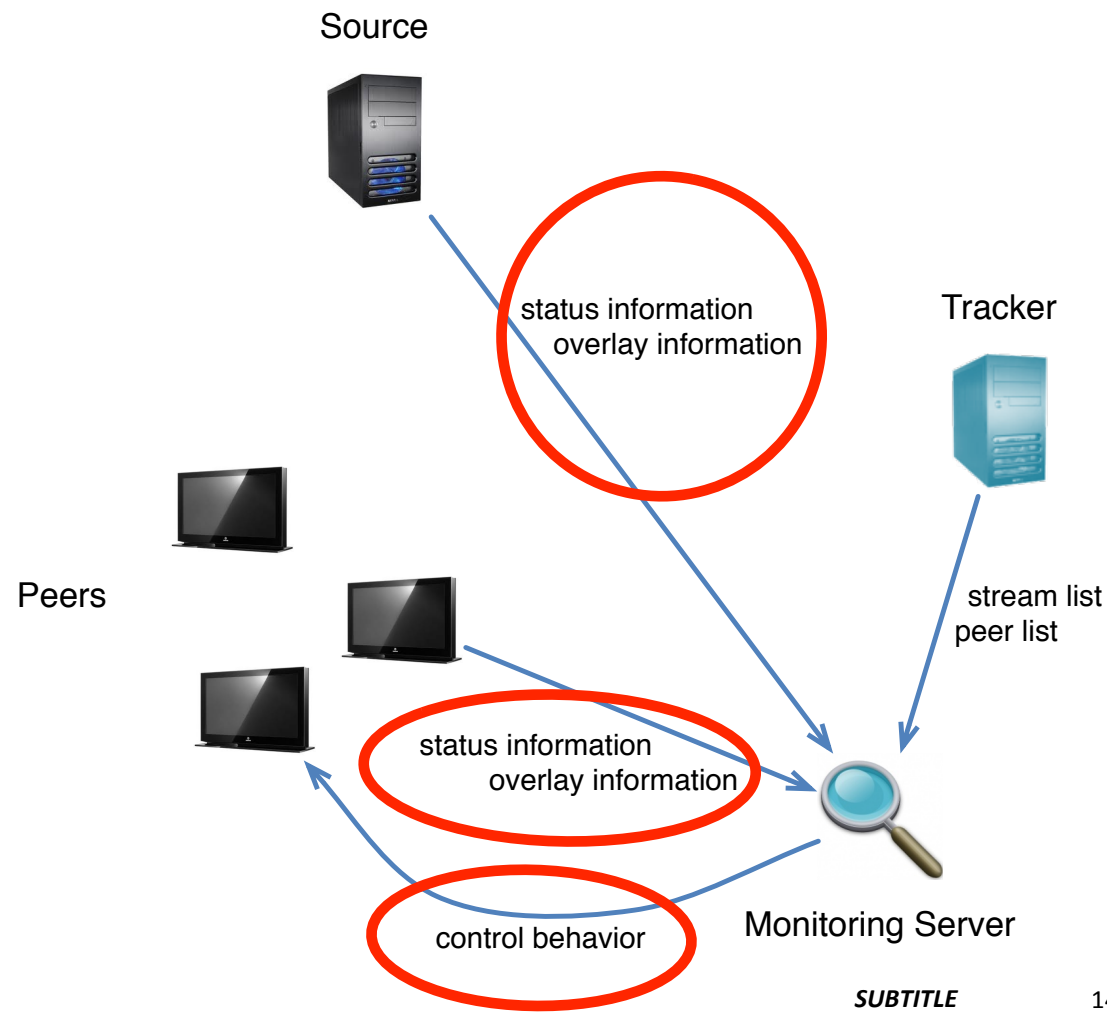
- M_GET_STREAMS
- M_GET_ALL_PEERS
- M_GET_ALL_PEERS_FOR_STREAM



Monitor <=> Source / Peer

- GET_INFO
- GET_PARENTS *
- GET_CHILDREN

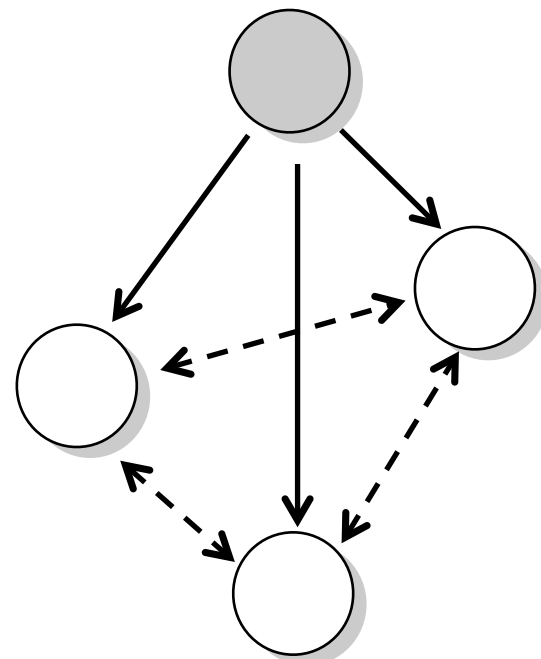
- SET_PROPERTIES
- TERM *
- KILL *



Local Overlay Optimizations

- Only parents optimize local situation

- Two possible operations
  1. Move down (forward child to another child)
  2. Move up (forward child to parent upon request)



1. Optimize in three steps
   - Select edge with highest cost
   - Select best alternative parent
   - Calculate gain and execute operation if gain > threshold

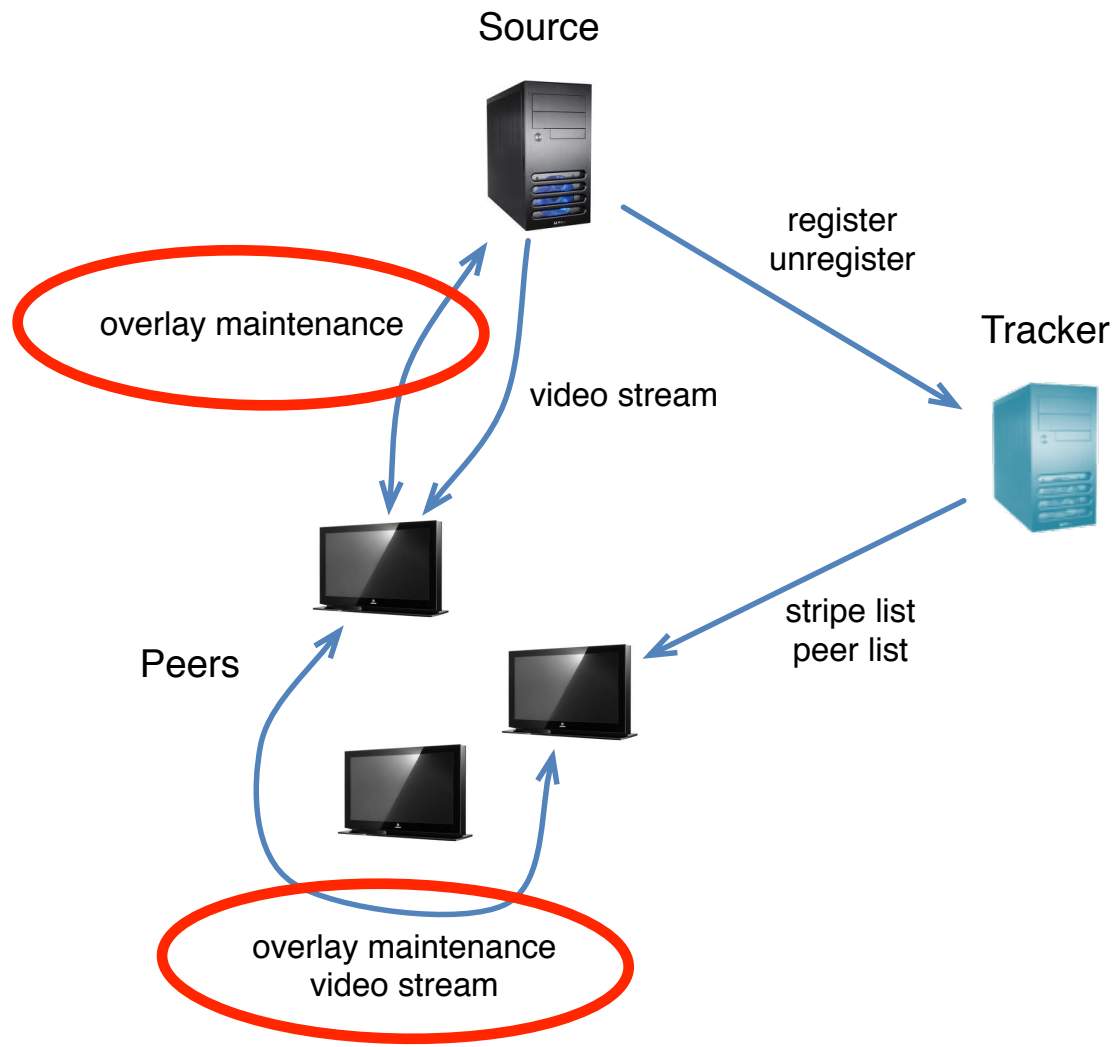
2. Bandwidth available?
   - Request new successor from neighbors

Peer <=> Peer / Source

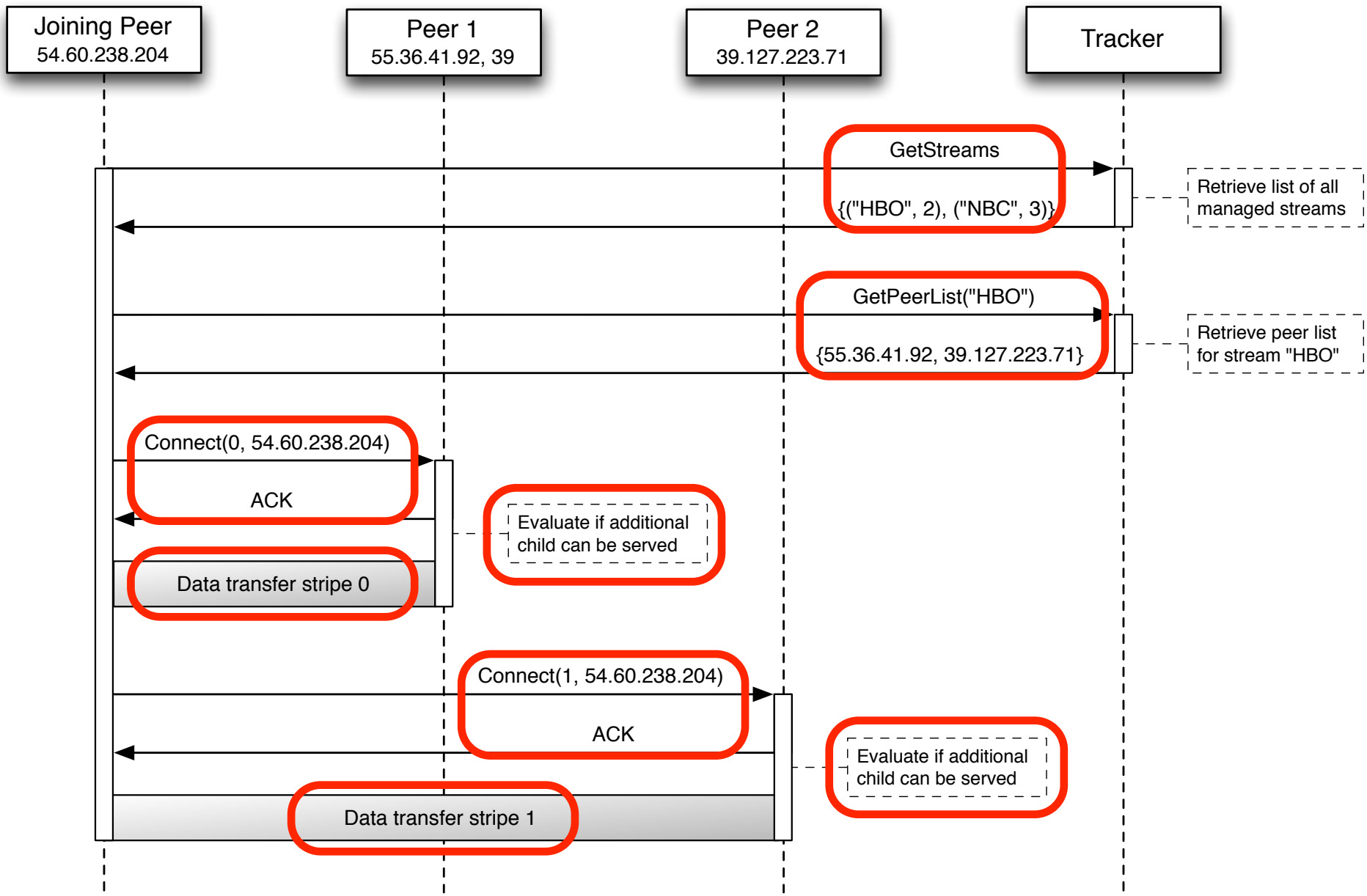
- ALIVE

- LEAVE
- LEAVE_STRIPE

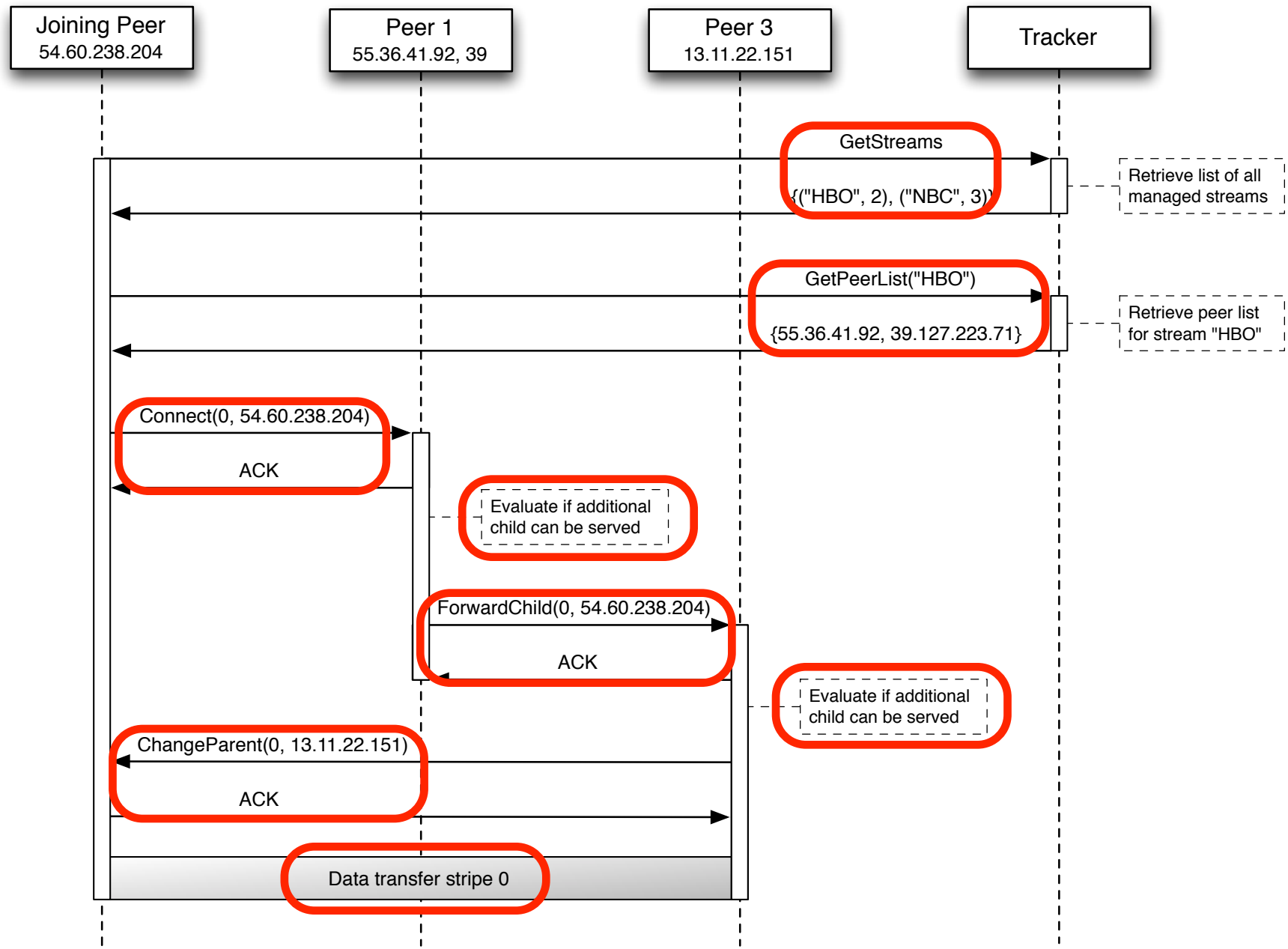
- CONNECT
- FORWARD_CHILD
- REQUEST_CHILD
- CHANGE_PARENT



Peer Join Procedure



Peer Join Procedure with Forward



Implementation

(Re-)Implementation

- Java version 1.6
- Android version 2.2.1
  - Planning to use version 4 (Ice Cream Sandwich) when available
- Messaging: Protocol Buffers
  - http://code.google.com/p/protobuf/
  - http://code.google.com/apis/protocolbuffers/docs/javatutorial.html

```
message RegisterStream {
  required string stream_identifier = 1;
  required Address maintenance_address = 2;
  required fixed32 stripes = 3;
}
```

```
message GetPeerList {
  required string stream_identifier = 1;
  required Address maintenance_address = 2;
}
```

```
message Connect {
  required string stream_identifier = 1;
  required fixed32 stripe_identifier = 2;
  required Address maintenance_address = 3;
  required Address data_address = 4;
}
```

DEMO

Demo Scenarios

- **test1.sh**
  - **5 streams, 5 peers each, 1-5 stripes, bw=(2/2)*stripes**
- *test2.sh*
  - *1 stream, 3 peers, 1 stripe, bw=(3/2)*
- **test3.sh**
  - **1 stream, 2 peers, 1 stripe, bw=(1/1)**
- *test4.sh*
  - *5 streams, 0 peers, 1-5 stripes, bw=(1/1)*stripes*
- *test5.sh*
  - *1 stream, 0 peers, 1 stripe, bw=(1/1)*
- *test6.sh*
  - *1 stream, 5 peers, 5 stripes, bw=(20/20)*

Next Steps

- Distribute RTP streams using overlay
- Display video at clients (Android as well!)
- Include AIMS interface in tracker
- Optimize overlay using AIMS location information
- Deploy software on Campus-Wide Testbed