Erlang Media Server

Custom Protocols & DDoS Defense

mediaserver2:start_link()

- Old Highly-Threaded Java System
- Costly to Maintain
- Costly to Protect
- Group Capacity Limited to Single Server
- Commercial DDoS Unsuitable for Media
- Complex Links to Presence Server

Objectives

- Single Group / Multiple Servers
- Custom DDoS Modules
- Simplicity
- Robustness
- Impartial to Hosting Solution
- Near Independence from Presence Server

-define (SOLUTION, “Erlang”)

- Easy Language Semantics
- Built-in Supervision
- Process Isolation
- Amazing Clustering Support
- BUT MOST OF ALL…

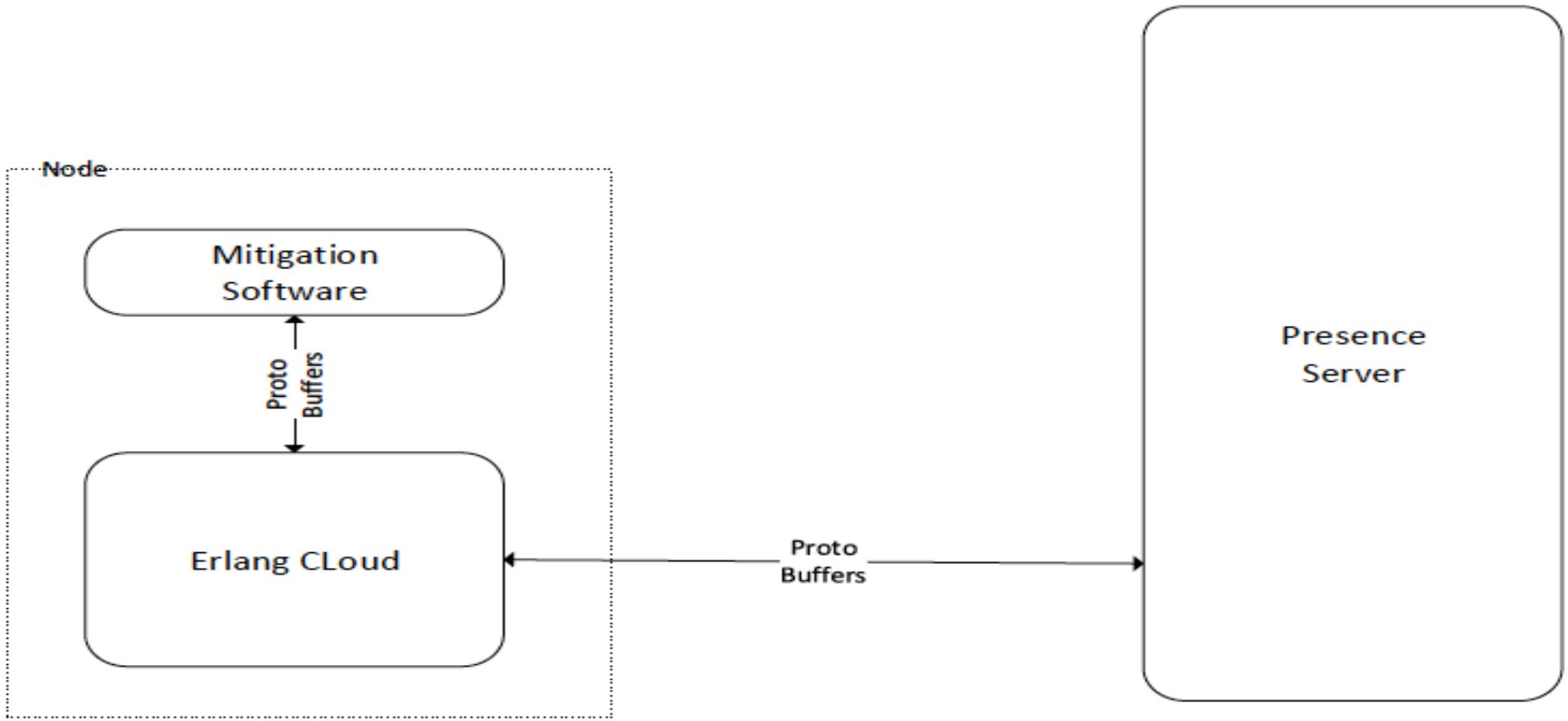
OTP!

- Full Use of All OTP Modules
- Mnesia
- Dialyzer
- Realtime Code Updates
- Multi-Node Support
- gen_event

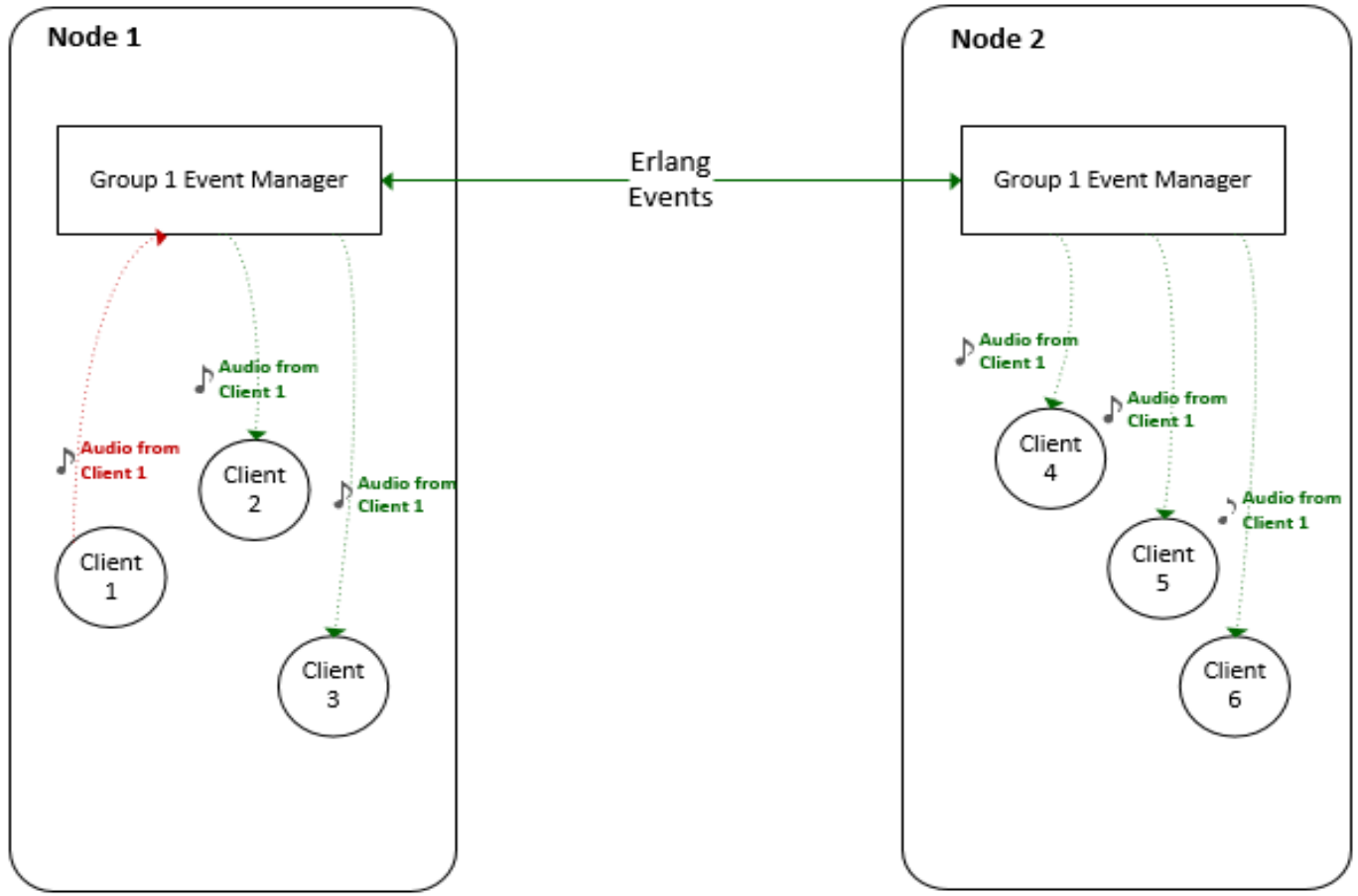
Basic Infrastructure Overview

- Clusters of 10 – 20 CentOS Boxes
- Clusters Multicast Group Voice to Clients Connected Anywhere
- Mnesia Maintains Client/Cluster Topology
- Google Protobuf Comms to Presence Server
- Google Protobuf Comms to DDoS Module on Each Host
- Multiple Simultaneous Clusters in Operation

20,000 Foot View



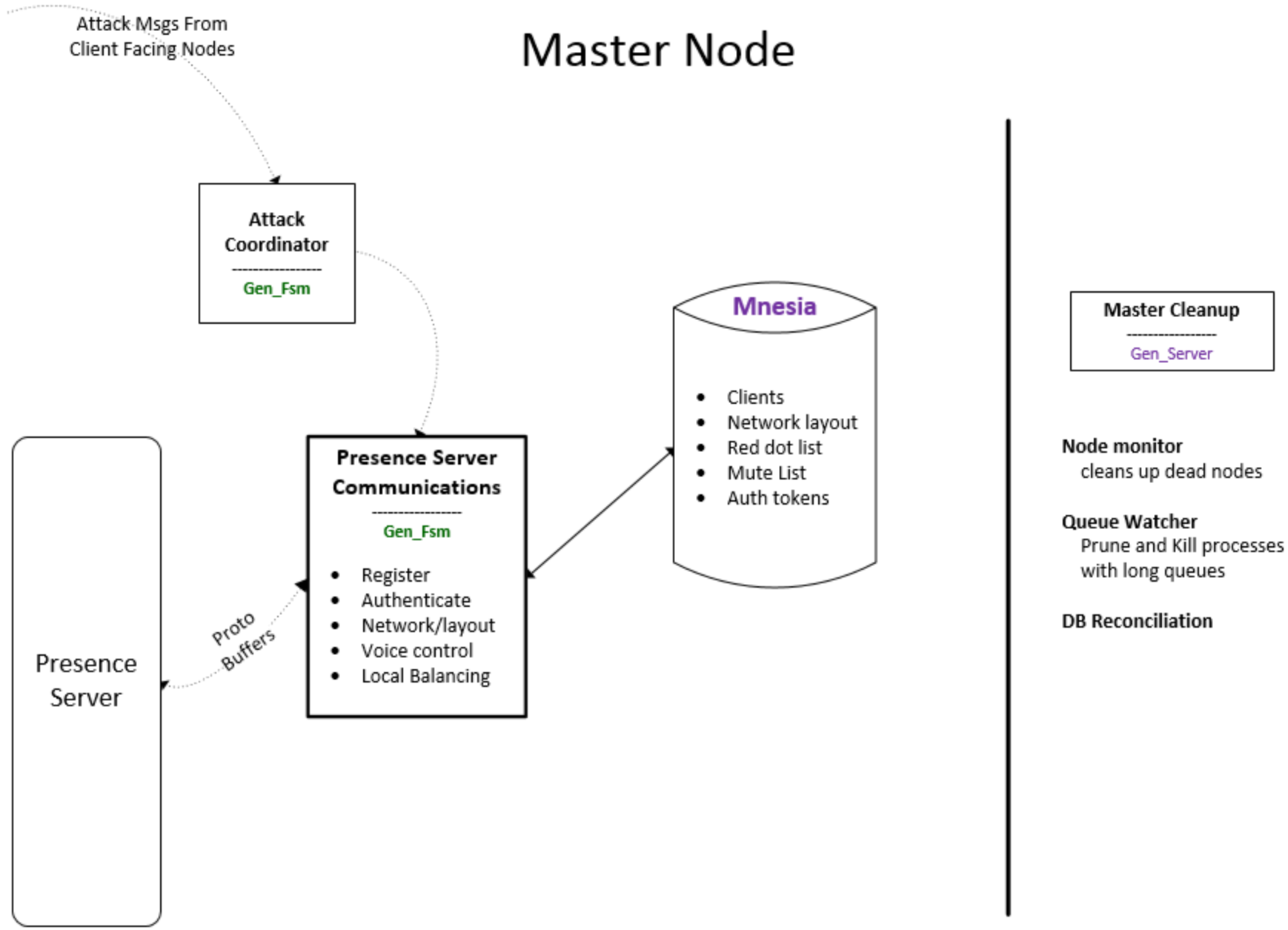
Packet Distribution



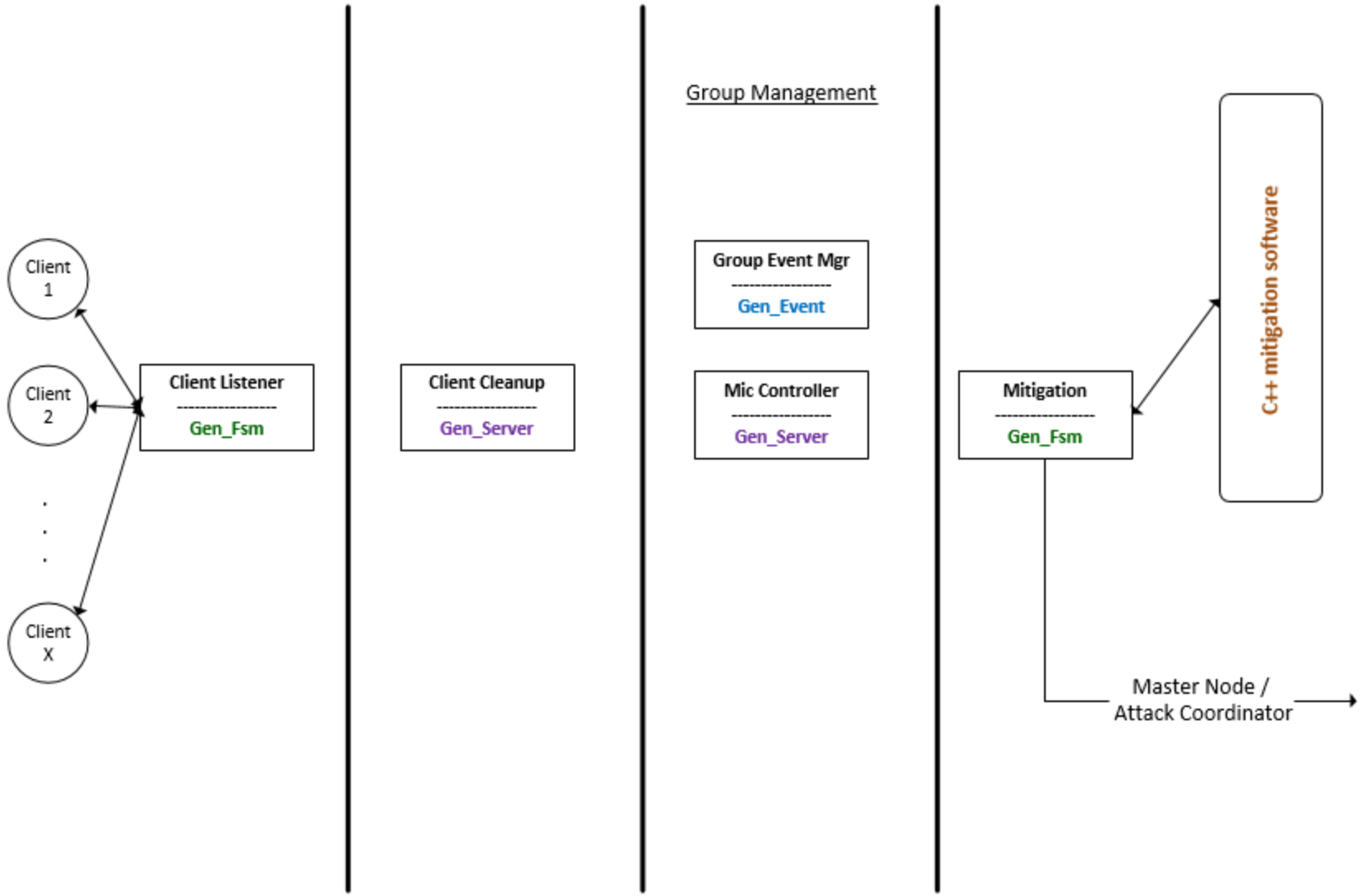
2 Types of Nodes

- Master
  - Communications to Java Presence Server
  - Stores Connection Topology in Mnesia
  - Coordinates Moving Clients From Attacks
- Client-Facing
  - gen_fsm for TCP Client Connections
  - gen_event for Packet Distribution Intra/Inter Node
  - Communications with Localhost DDoS Mitigation (C++)

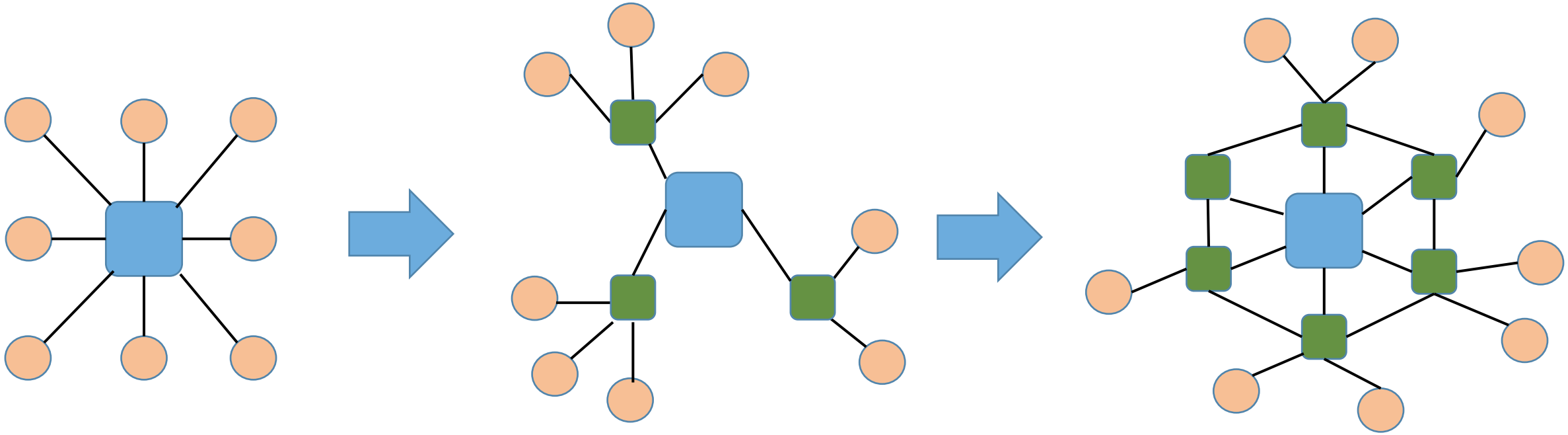
Master Node


Attack Msgs From
Client Facing Nodes
Attack
Coordinator
Gen_Fsm
Mnesia
Clients
Network layout
Red dot list
Mute List
Auth tokens
Master Cleanup
Gen_Server
Presence Server
Communications
Gen_Fsm
Register
Authenticate
Network/layout
Voice control
Local Balancing
Proto
Buffers
Presence
Server
Node monitor
cleans up dead nodes
Queue Watcher
Prune and Kill processes
with long queues
DB Reconciliation

Client Facing Node



DDoS Mitigation



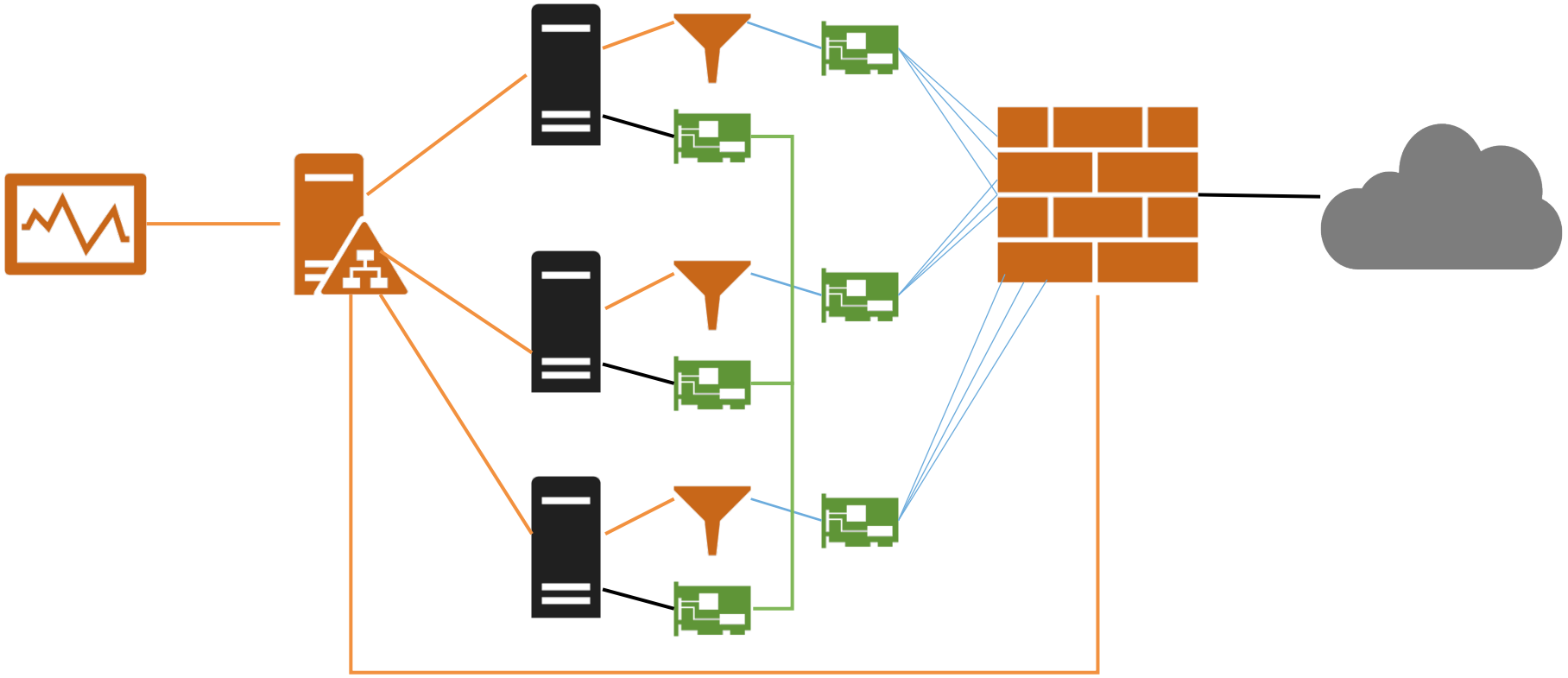
Standard approach leads to failure. Kicking single server causes whole group go down or at least kicks all the consumers of specific publisher. Makes attackers easier to target / distract the conversation => more angry users.

Distribute everything but don't forget about internal network. Spreading too much may cause overflow and degradation of quality.

Let it fail – don't rely on a stable connection, make it reconnect quick (aggressive from both server and client side). Lots of work on the client side too. Can't really go for HTTP model (every request is a different connection) but at least try to emulate that.

- Geographic and content-related issues. Asian, middle east attacks, specific regions. Adult content leads to more attacks, network-specific attacks (room wars for a top place).
  - Separating offensive regions to different nodes.
  - Offload less attacked services separately with less resources involved.
  - Learn on how to detect affected services and move them to the quarantine.

- Can’t really solve link saturation on a host itself. Even 10gbps attack is nothing these days.
  - Look for provider with huge uplinks (300, 600gbps or bigger) and an ability for controlling access rules on the edge of a network.
  - Cutting bad traffic upstream is the only solution.



Make them search. Having multiple servers (hundreds, thousands?) is costly.

- Assigning even /28 network multiples endpoints with x16 rate.
- Use random ports – otherwise it would be easy to find all other addresses on the same subnet.
- Use all addresses / all ports all the time. Make them deal with thousands of address/port combinations.

Implement early detection and monitoring

- System level network interface monitoring, kernel extensions.
- Detect known attack patterns early before it could really affect the OS networking level, cut the interface out ASAP.
- Build a network of monitoring agents, make them all communicate to warn each other and learn from others, make them all able to control the outside firewall / network edge.
- Constant monitoring, personal involvement. They never stop and are always looking for the new ways. Be able to modify all parts of the system on the fly without interruption.

Automated user feedback

- Collect as much user feedback as possible. Connection failures, latency, everything counts.
- Use it to change routes automatically.
- Automated blacklisting model. Don’t let bad endpoints stay up.

Next steps

- Get rid of “1 stream” = “1 connection” model. Make all connections share all the traffic at the same time.

Audio Cloud Features

- Streaming Group Audio Multicast
- Muting/”Red-Dotting” Support
- MultiChannel ActiveSpeaker Selection
- Multi-Codec Support (Reflector Server Only)
- Rate Controls and Auto-Reconnect
- Load-Balancing of Client Connections

Audio Deployment Types

- Multi VIP OVH cluster
- Traditional DC with dual/single network interface
- Amazon cluster
- Standalone single-node development mode
- Controlled by environmental variables

Cloud Stats

- 1 Cluster, 10 Servers, 50 Thousand Concurrent
- 2 + gigs/ sec of Audio Traffic
- Scalability? Nearly Infinite
  - Prototypical parallel system. Good Horizontal Scalability
  - N clouds can be expanded to N+1, 1 Cloud Connection/Presence Server
- Uptime is Excellent (No Recent Outages)
- Reliability is Outstanding
- Flexibility is Interesting…

Lessons Learned

- Erlang is Easy to Learn
- OTP Differentiates Erlang
- Dialyzer Is Key for Static Analysis
- Code Changes Do NOT Seem to Perturb the System (proc isolation)
- code:load_file (Still Learning but Huge +)
- Interlanguage support is key. Not one language solves it all
- Somehow It Just Works...