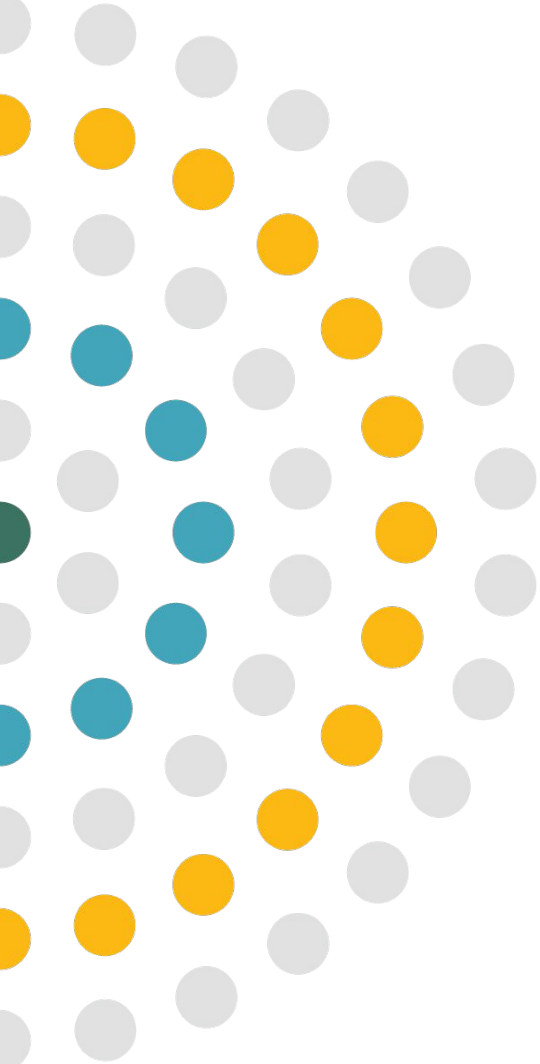


FDC³

Desktop Agent Bridging

Topologies vs. Discovery, Auth & Message Routing

Composition of a Desktop Agent Bridging proposal

A bridging proposal must include protocols that allow a DA to:

- **Locating** other agents to connect to
  - via configuration or discovery
- **Connecting** securely to other agents
  - & re-establish connections if they drop for any reason
- **Interacting** with other agents across the connection
  - & handle failures, timeouts etc.

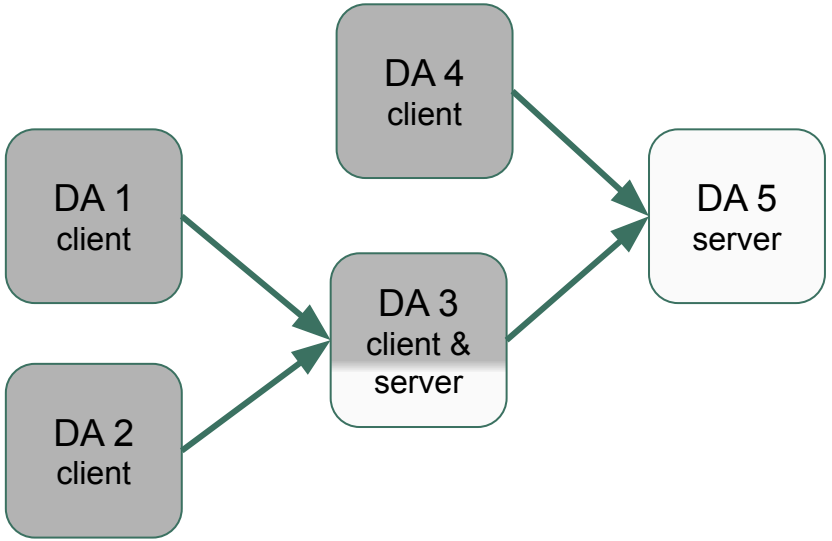
The **topology** of the connected Desktop Agents affects the complexity of the solution for each aspect…

The different topologies we could use

Clients & Servers

Each Desktop Agent:

- Implements Client behavior
  - Forward requests to servers,
  - Await responses from servers
  - Receive requests from servers
- May implement Server behavior
  - Receive requests from clients
  - Route requests to clients
  - Await responses from clients
  - Route responses to client

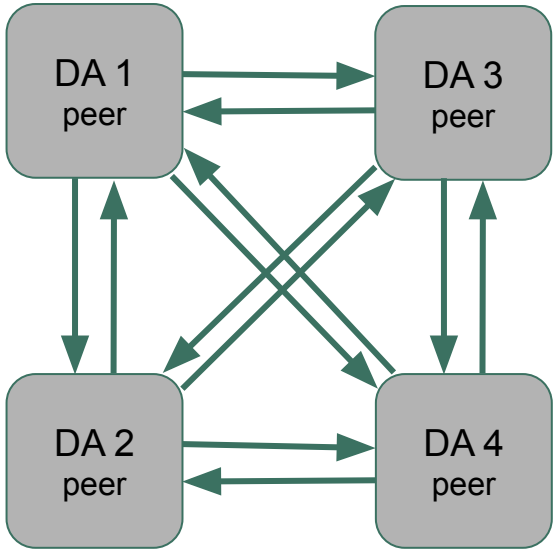


The different topologies we could use

Peer-to-Peer

Each Desktop Agent:

- Implements 'peer' behavior:
  - Forward requests to peers
  - Await responses from peers
  - Receive requests from peers



The different topologies we could use

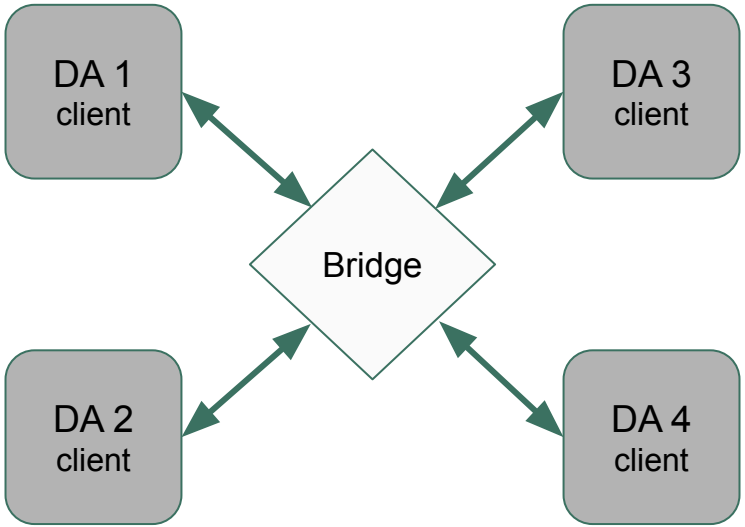
Standalone bridge

Each Desktop Agent:

- Implements Client behavior
  - Forward requests to bridge,
  - Await response from bridge
  - Receive requests from bridge

The Bridge (not a Desktop Agent):

- Implements 'Server' behavior:
  - Receive requests from clients
  - Route requests to clients
  - Route responses to client



How the topology affects: Locating

Clients & Servers

💔
- Discovery
  via a Service registry
  - Polling for discovery
  - Topology might change

  OR
- Config (per agent)
  - Polling for connections

💛
- Multiple roles to implement/pick from

💛
- Uses S websockets/ports
  - Port conflicts

Peer-to-Peer

💛
- Discovery
  via a Service registry
  - Polling for discovery / Return connections

  OR
- Config (per agent)
  - Polling for connections / Return connections

💚
- Single role to implement

💔
- Uses N websockets/ports
  - Port conflicts

Standalone bridge

💚
- Discovery
  via known port

  OR
- Config (all same)

💚
- Single role to implement

💚
- Uses 1 websocket/port

How the topology affects: Connecting

Clients & Servers

- 💛 Authentication:
  - Clients => servers OR
  - Creds exchanged OR
  - SSO (hard to standardise)
- 💔 Servers are multiple points of failure
  - restart = failure
  - topology might change on reconnect
- 💔 DA IDs applied by servers, clashes possible

Peer-to-Peer

- 💛 Authentication:
  - Credentials exchanged
  - Requires configuration: every DA against the others (many sets of creds) OR
  - SSO (hard to standardise)
- 💚 No individual point of failure
- 💛 DAs set own names, clashes possible

Standalone bridge

- 💚 Authentication
  - Clients auth against bridge
  - Allows for simpler auth schemes (e.g. access keys)
- 💛 Bridge is single point of failure
- 💚 Simple system service less likely to fail/restart
- 💚 Bridge sets DA names (name can be requested)

How the topology affects: Interacting

Clients & Servers

- 💔 Complex message propagation logic
  - Loops and alternate routes
  - Need message paths
- 💔 Multiple roles for DAs to implement
  - can change at runtime
- 💛 Moderate to implement multi-machine
- 💔 Hard to implement ACLs

Peer-to-Peer

- 💚 Simple message propagation
  - Only need source/destination
  - Aggregate responses
- 💚 Single role for DAs to implement
- 💔 Hard to implement multi-machine
- 💚 Easy to implement ACLs

Standalone bridge

- 💚 Simple message propagation
  - Only need source/destination
  - Aggregate responses (in bridge OR DAs)
- 💚 Single for role DAs to implement
- 💛 Bridge to be implemented
- 💚 Easy to implement multi-machine and ACLs

Summary

Scoring 💔 = 0 💛 = 1 💚 = 2

Clients & Servers

- Locate:
  1 💔 2 💛
- Connect:
  2 💔 1 💛
- Interact:
  3 💔 1 💛

Score = 4/20

Peer-to-Peer

- Locate:
  1 💔 1 💛 1 💚
- Connect:
  2 💛 1 💚
- Interact:
  3 💔 1 💚

Score = 9/20

Standalone bridge

- Locate:
  3 💚
- Connect:
  0-1 💛 2-3 💚
- Interact:
  1 💛 3 💚

Score = 18.5/20

The case for a standalone bridge

- We probably don't need complex topologies on the desktop
  - message paths become trivial
- Uses less websockets/ports
- Simpler to configure than peer-to-peer
- Easier to handle multi-machine use-cases
- Can move some of the message routing complexity to a bridge implementation
- Bridge implementation can be relatively simple, it is responsible for:
  - Name assignments
  - Message routing
  - Maybe channel state or aggregating responses