CSx55: Distributed Systems
[Distributed Servers]

The premise of distributed servers
A hoard of servers hums in the back
A switch out at the forefront
    funneling requests one at a time
    balancing the load

Powered by TCP handoffs
the responses return as if
they spring from the switch
and not the hidden crowd of servers

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Can syntactic sugar obscure concurrency issues?
- How is throughput typically calculated?
- Will performance always be capped by the serial element?
- When to use a barrier vs. a latch?
- Where there a multiple queues of differing priorities, could we have more threads to service the highest-priority queue?

Topics covered in this lecture

- Thread architectures on the server-side
- State in Servers
- Distributed Servers
- TCP Handoffs
- Route optimizations using MIPv6

Thread-per-request architecture

- Worker thread is spawned for **each** incoming request
  - Worker thread *destroys itself* after processing request
- Advantages:
  - Threads do not contend for the shared work-queue
  - Throughput is potentially maximized
- Disadvantage
  - Overhead for thread creation and destruction operations

Thread-per-connection architecture

- Associates a thread per connection
- New worker thread created when a client makes a connection
  - Destroyed when client closes the connection
- Client may make many requests over the connection

Thread-per-object architecture

- Associate a thread with each remote object
- A separate thread receives requests and queues them
  - But there is a queue per-object

Thread-per-connection & Thread-per-object

- Advantages
  - Server benefits from lower thread management overheads compared to thread-per-request
- Disadvantages
  - Clients may be delayed when a worker thread has several outstanding requests, but another thread has no work to perform

Server Design Issues

Server Design Issues

- **Iterative** Servers
  - Handles request
  - Returns response to requesting client
- **Concurrent** Servers
  - Pass request to a separate thread/process
    - **Multithreaded server**
  - Await new incoming request

The **endpoint** issue

- Clients send their requests to an endpoint
  - **Port** to which a server listens to
- But how do clients know about a port?
  - Globally assign endpoints for well-known ports
    - Internet Assigned Numbers Authority (IANA)
    - FTP {TCP, 21}, HTTP {TCP, 80}

Implementing each service with a separate server could waste resources

- Instead of having multiple servers awaiting client requests
  - Have a single **super-server**
- INETD daemon on Unix
  - Listens to **several ports** for Internet services
    - Pop3 {110}, FTP {21}, Telnet {23}
  - When request comes in:
    1. **Fork** process to handle it
    2. Process **exit**s once done

Designing Servers: Support interruption

- Terminate client session
  - Server will eventually detect connection loss (TCP)
- Send **out-of-band** data
  - Data to be processed before any other client data
- But how can we send this out-of-band data?
  1. Send to a different port
  2. Reuse same connection
     - TCP **urgent data** e.g., `socket.sendUrgentData(int data)`

Some folks like to get away
Take a holiday from the neighborhood
Hop a flight to Miami Beach or to Hollywood
But I'm taking a Greyhound
On the Hudson River line
I'm in a New York state of mind

New York State of Mind; Turnstiles; Bill Joel

STATE

Tracking State in Servers

- Stateless servers
- Stateful servers

Stateless servers

- No state information about clients
  - E.g., Web Servers
- Usually, some state is maintained
  - Log of documents accessed by client
  - But if this is lost, there should be **no disruption** of service
- **Soft state**: track state for a limited time
  - When timer elapses, revert to default behavior

Stateful servers

- Maintain **persistent** information on clients
- Use this to improve **performance**
  - Real and perceived
- Special measures needed to recover from failures

Stateful servers: A file server example

- Allows client to maintain **local copy** of file
  - Even for updates to the file
  - Maintain {client,file} tuples to track file state
- Identify who has most recent version of file
- If server crashes it must recover the {client,file} entries

A hybrid approach: Have the client send its state to the server

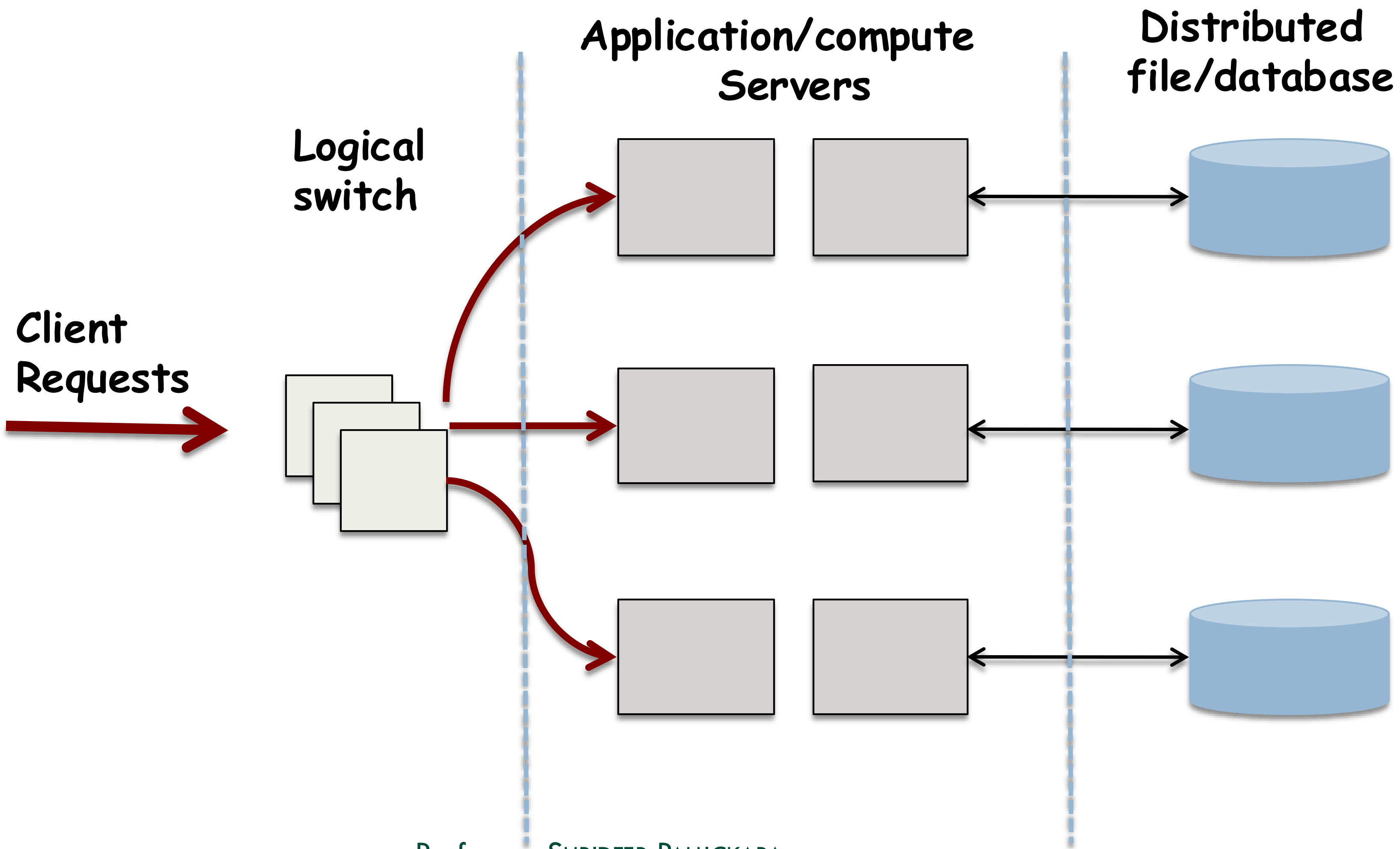
- **Cookies** serve this purpose for Web pages
- Tells a site about the pages accessed by a user
  - Use this to decide how to manage client
  - Sent back to browser every time state info changes
- Cookies don't stay where they are baked!

Distributed Servers

Mean time for failures and the premise for **distributed servers**

- Group several machines together
- Don't rely on the availability of any single machine
- Together, achieve better stability than each component individually
  - The sum is greater than the parts

Server Clusters



Server Clusters

- Switch is also responsible for **load balancing** requests
  - Simplest way to do this is using round-robin

- If there are different services offered within the cluster?
  - Switch needs to dispatch requests appropriately

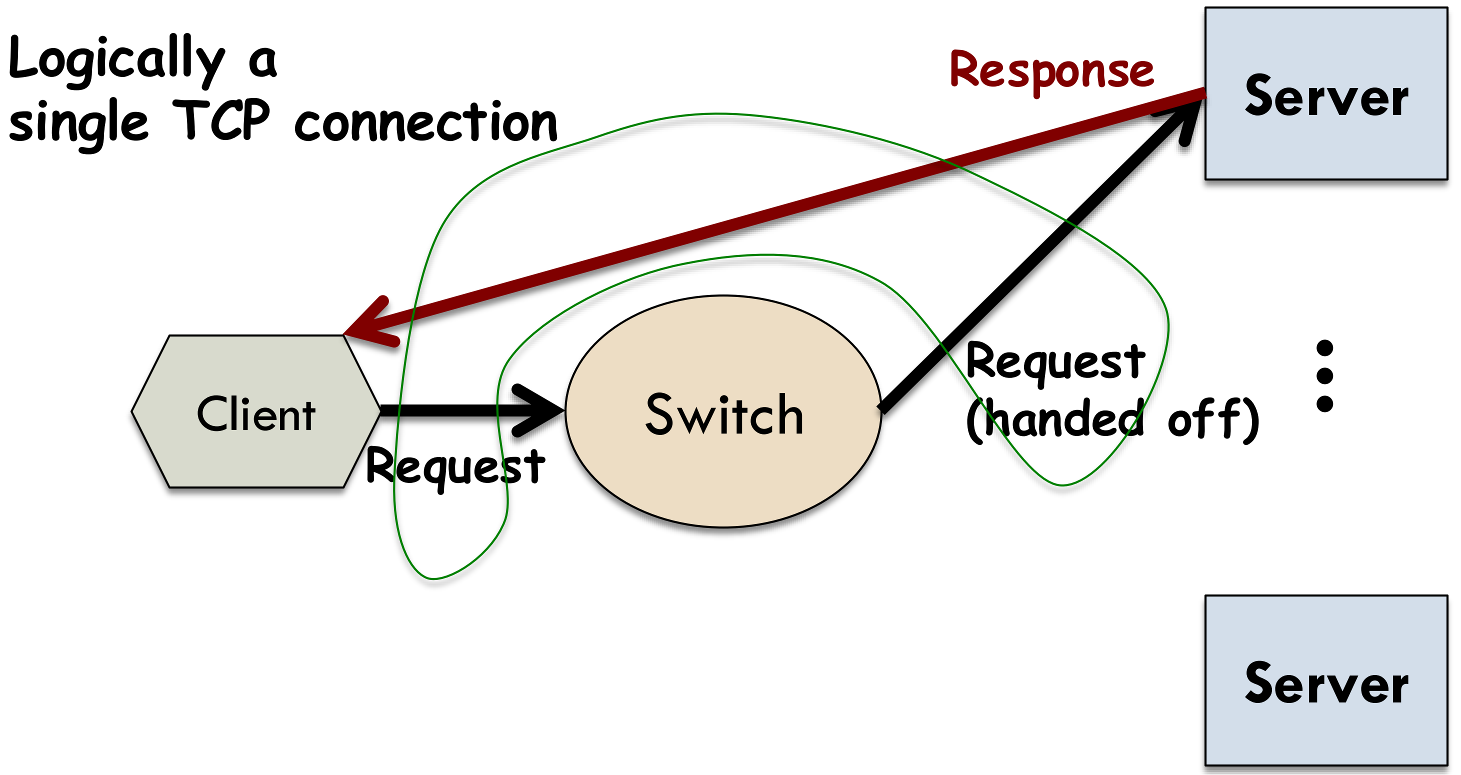
But what about transparency?

- An important consideration is that the server cluster is **transparent**
- Clients typically set up network connections over which requests are sent

But TCP expects an answer from the switch not some arbitrary node

- When server responds to client
  - Inserts **switch's IP address** in source field of the IP packet
- Requires **OS-level modifications**
- Also used in content-aware request distribution

The principle of TCP handoffs



When a cluster offers a single point …

- When there is a failure at that access point?
  - The entire cluster becomes unavailable

- Several access points are typically provided
  - DNS can return **several addresses** all mapped to the same host name
  - Client makes several attempts if there are failures
  - Still requires static access points

Pulls and trade-offs

- Stability
  - Long lived access point
- Flexibility
  - Ability to configure a server cluster including the switch

What would be really nice

- Distributed server with a **dynamically changing** set of machines
- And also varying access points

Mobile IPv6

Mobile IPv6 at a glace

- IPv6 provides the foundation
  - Provides addressing and packet routing
  - Defines the structure of communication on the Internet
- The problem of mobility
  - Devices move across Wi-Fi, cellular, or other networks
  - Without support, each move would **break connections** or **force a new IP address**
- Mobile IPv6 as the solution
  - Extends IPv6 with mobility support
  - Lets a device keep a **stable home address** while roaming
- Manages handoffs so applications stay connected, uninterrupted
- Crux: Separate ***where*** you are (current point of attachment) from ***who*** you are (your permanent IP identity)

Mobility support in IP version 6 (MIPv6)

- A **mobile node** has a **home-network**
- This node has a **home-address**
- The node has a **home agent**
  - Takes care of traffic to the mobile node while it is away

Mobility support in IP version 6 (MIPv6)

- When a mobile node attaches to a **foreign network**
  - Gets a temporary **care-of address**
- Care-of address reported to the home-agent
  - **Forward** all traffic to the mobile node

Apps communicating with mobile node only see the home address and not the care-of-address

- Offers a stable address for a distributed server
  - A single, unique contact address is initially assigned
- Contact address is server's **lifetime** address

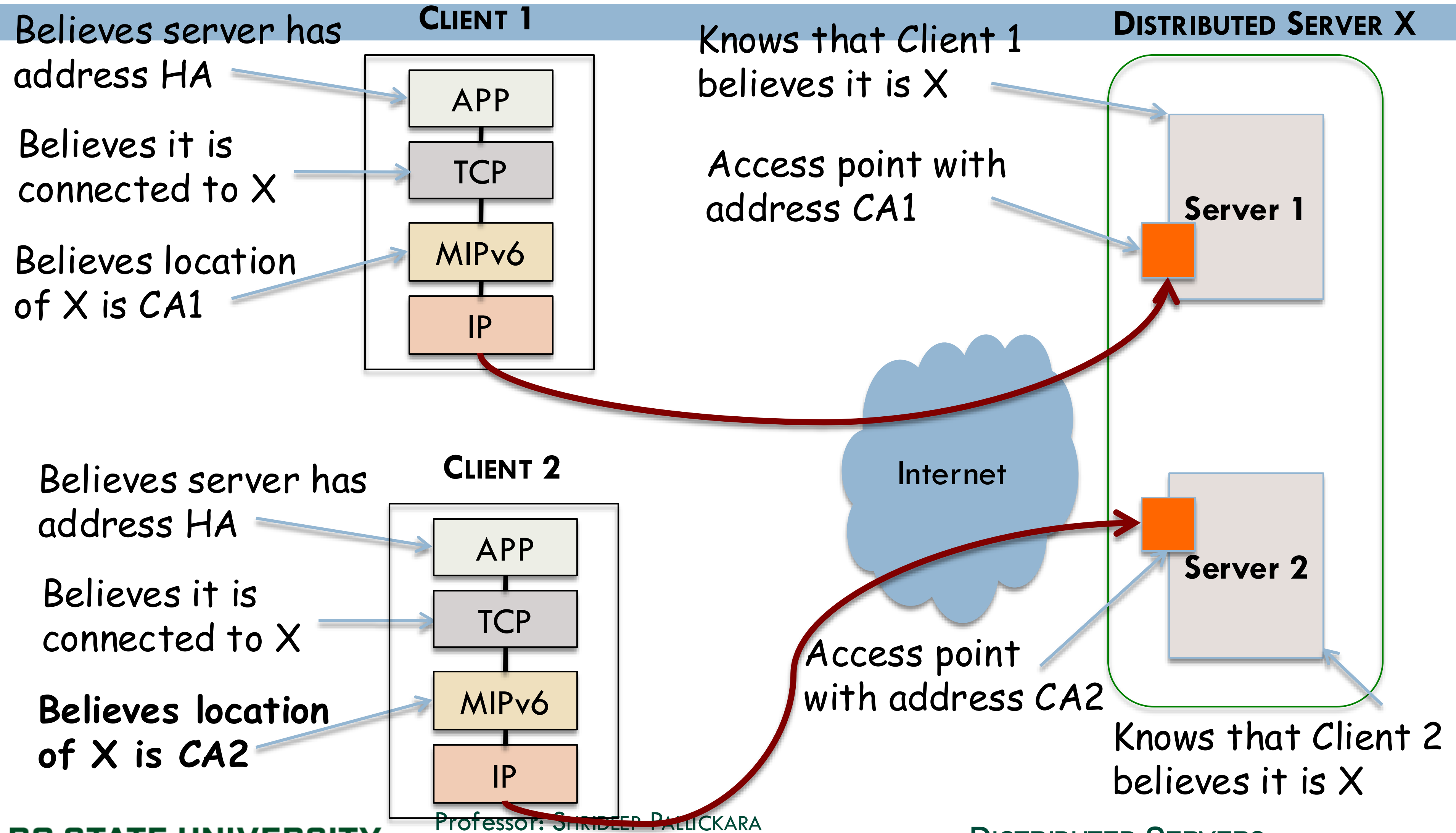
Any node can act operate as the access point

- Record own address as the care-of address
- All traffic will be directed to the access point
- If there's a failure at the access point?
  - Another node takes over
- Potential bottlenecks?
  - Home agent and access point
  - All traffic **must flow** through them

The route optimization feature in MIPv6

- When a mobile node reports its care-of address (CA) to the home-agent (HA)
  - The HA reports the CA to a client
- Client keeps **{HA, CA}**
- Communications will be with the CA
  - Applications can still use the HA
  - MIPv6 protocol stack will translate HA to CA

Depicting Route Optimizations



The contents of this slide-set are based on the following references

- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van der Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 6, 2]
- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 7, 14]