# Migratory TCP (MTCP)
## Transport Layer Support for Highly Available Network Services

Division of Computer and Information Sciences

Rutgers University

# The People

- Deepa Iyer
- Fang Song
- Kiran Srinivasan
- Florin Sultan
- Liviu Iftode

# Services Rather than Servers

- Client oblivious to a given server's location
- Client interested only in content provided and QoS
- Internet services
  - Most services are based on TCP
  - Clients demand both high-availability and performance
- Cooperative service model
  - Service -> set of logically equivalent servers
  - servers cooperate in sustaining service to a client

# TCP-based Internet Services

- Adverse conditions
  - Core network congestion,  overloaded servers, failed servers or under DoS attack
- TCP response
  - Network delays => packet loss => retransmission
- TCP limitations
  - Creates an implicit binding of service to *a* server !
  - Clients cannot change to another server for sustained service

# Migratory TCP

- Transport layer protocol that supports dynamic connection migration
- Extension to TCP
- A client connection can transparently migrate to different servers during its lifetime
  - Modified server applications cooperate to support the handoff
  - Client applications do not change
- Servers can be geographically distributed
- Does not depend on application-specific info

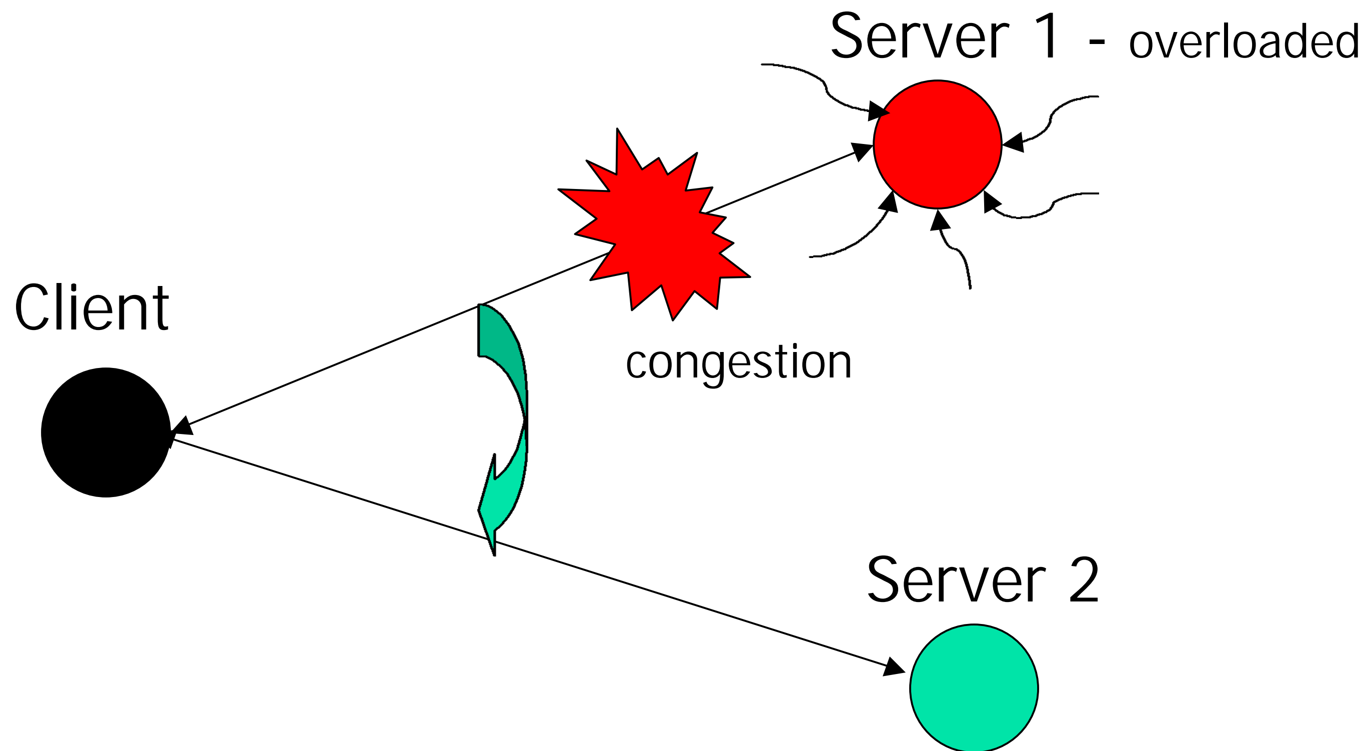
# Status

- Designed a transport layer protocol that enables dynamic connection migration
- Implemented a prototype on FreeBSD
- Demonstrated the utilization of our protocol in building highly-available services
- Conducted preliminary performance evaluation
- F. Sultan, K. Srinivasan, L. Iftode - "*Transport Layer Support for Highly-Available Network Services*", HotOS 2001

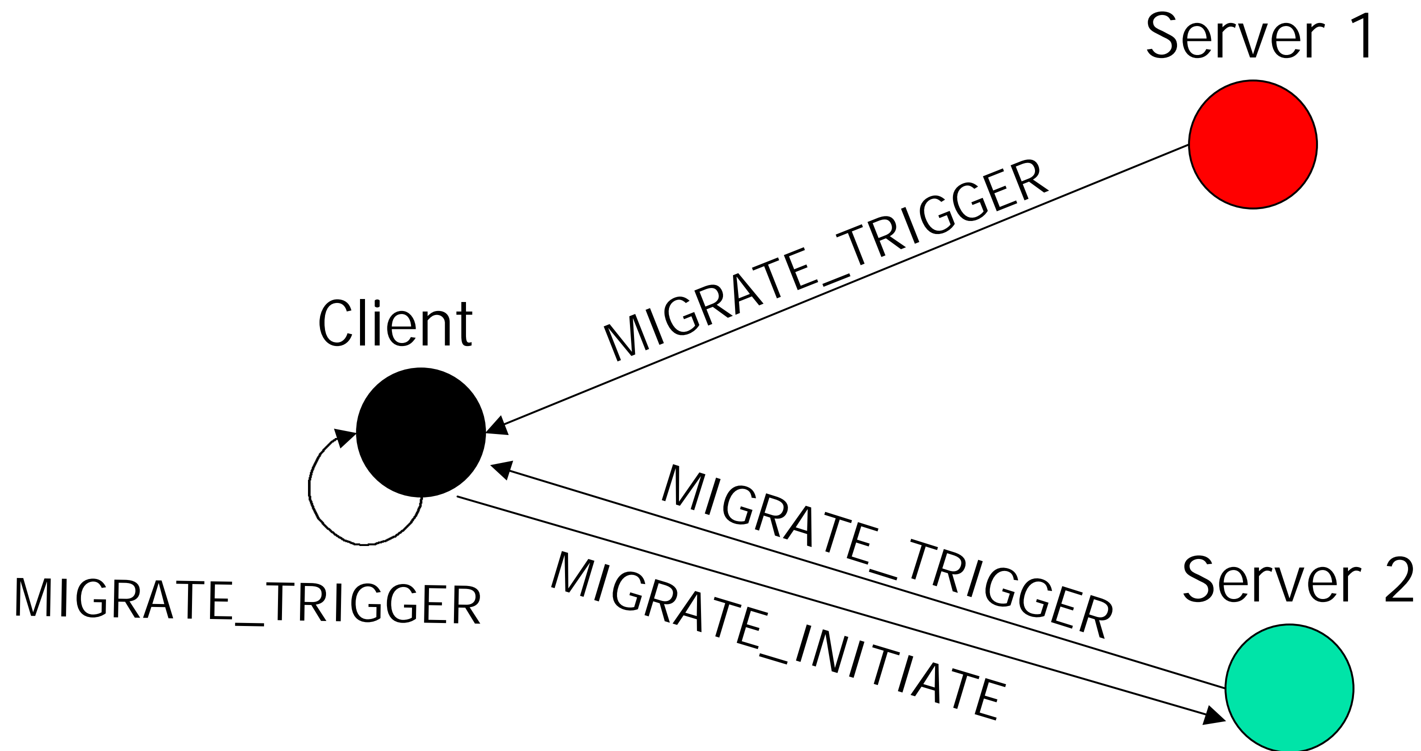
# Outline

- Motivation
- Protocol Design
- Prototype Implementation
- Preliminary Performance Evaluation
- Protocol Utilization
- Related Work
- Conclusions and Future Work

# The Migration Model



# Triggers and Initiators



# Design Issues

- How to resume the service at the new server?
  - Without the involvement of the client application
  - Without full migration of the server application
  - With low overhead
- What is the state that needs to be transferred?
  - Protocol state
  - Application state

# Contract Between Server Application and MTCP

- Application
  - Define fine-grained per-connection application state
  - Export to the protocol a per-connection application state snapshot
  - Import per-connection state after migration and resume service
- MTCP
  - Transfer the per-connection state
  - Synchronize the application state with the protocol state

# Server Migration API

- export_state(conn, state_buff)
  - origin server, many times (e.g., periodically)
- import_state(conn, state_buff)
  - destination server, once (after accepting connection)

- Enables light-weight migration *and* service resumption from a well-defined state
- Decouples execution of origin server from migration (no upcalls!)

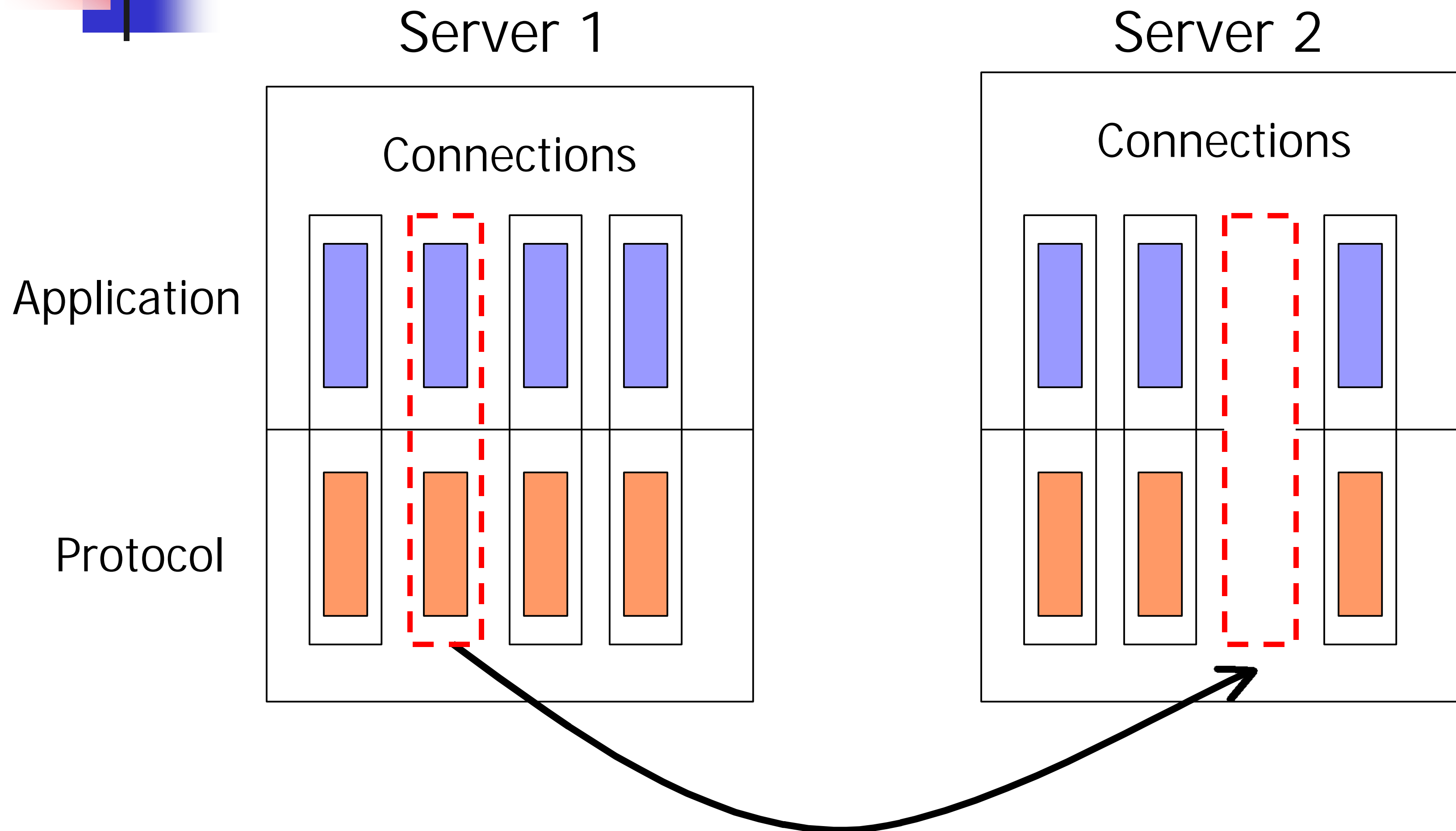
# Server Application Example

```
s = accept(ssock)

if (import_state(s, &state))
    num = state
else
    num = 0

for (i = num; i < NUM_MSG; i++) {
    recv(s, &msg)
    state = ++num
    export_state(s, &state)
}
```

# Contract Between Server Application and MTCP

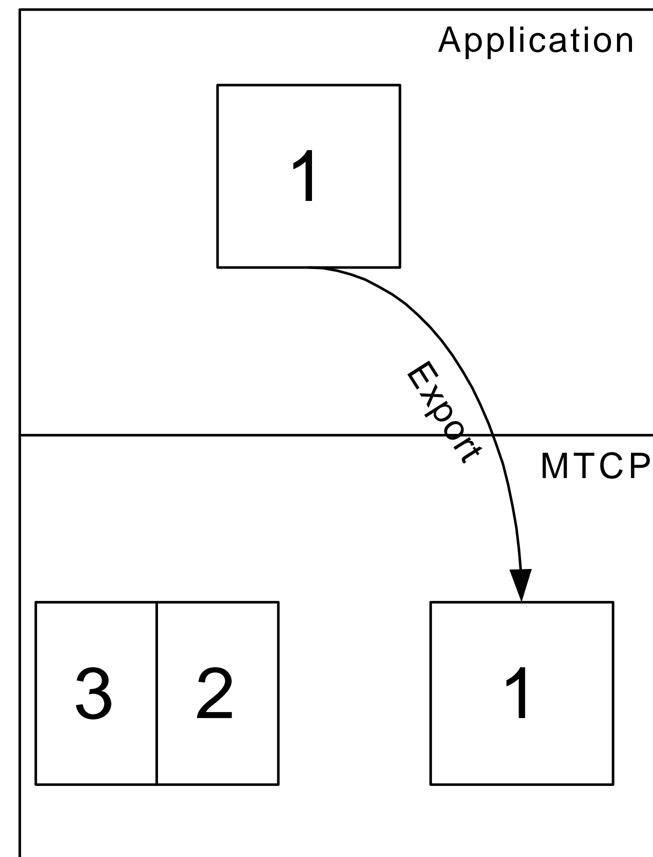
- Application
  - Define fine-grained per-connection application state
  - Export per-connection application state snapshot periodically
  - After migration, import per-connection state to resume service
- **MTCP**
  - **Transfer the per-connection state**
  - **Synchronize the application state with the protocol state**

# Per-connection State Transfer



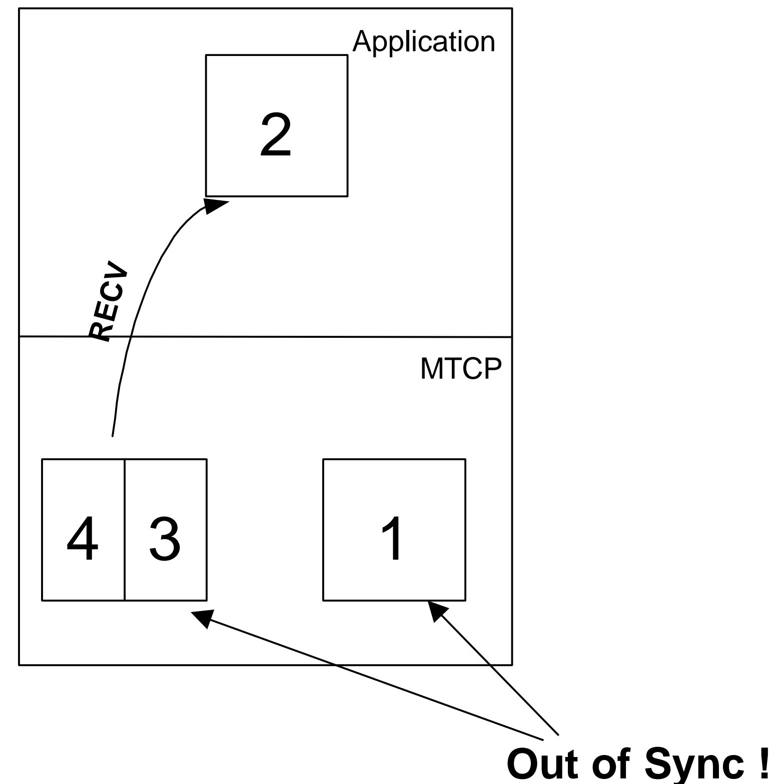
# State Synchronization Example (1)

```
s = accept(ssock)

if (import_state(s, &state))
        num = state
else
        num = 0
recv(s, &msg)
state = ++ num
export_state(s, &state)
recv(s, &msg)
state  = ++ num
export_state(s, &state)
   . . .
```

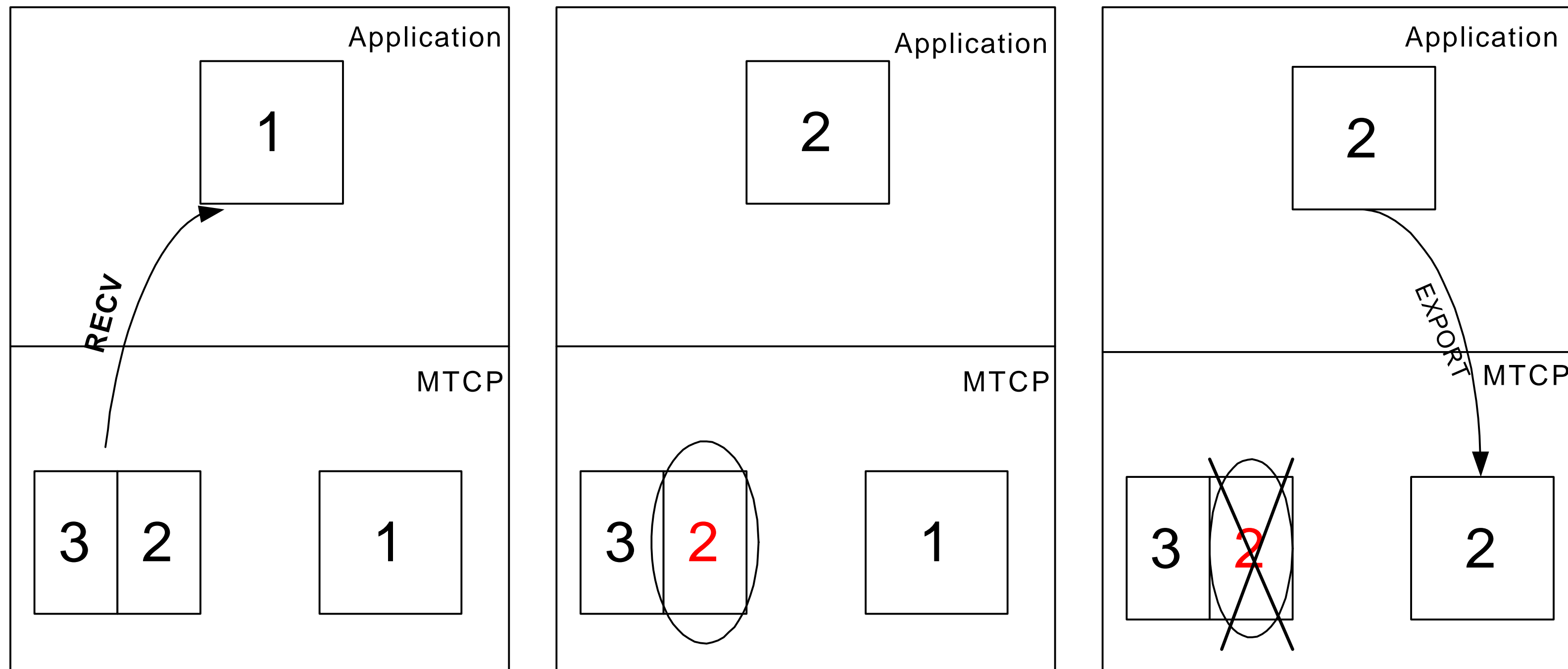


# State Synchronization Example (2)

```
s=accept(ssock)

if (import_state(s, &state))
      num = state
else
      num = 0
recv(s, &msg)
state = ++num
export_state(s, &state)
recv(s, &msg)
state  = ++num
export_state(s, &state)
  . . .
```



# State Synchronization Example (3)



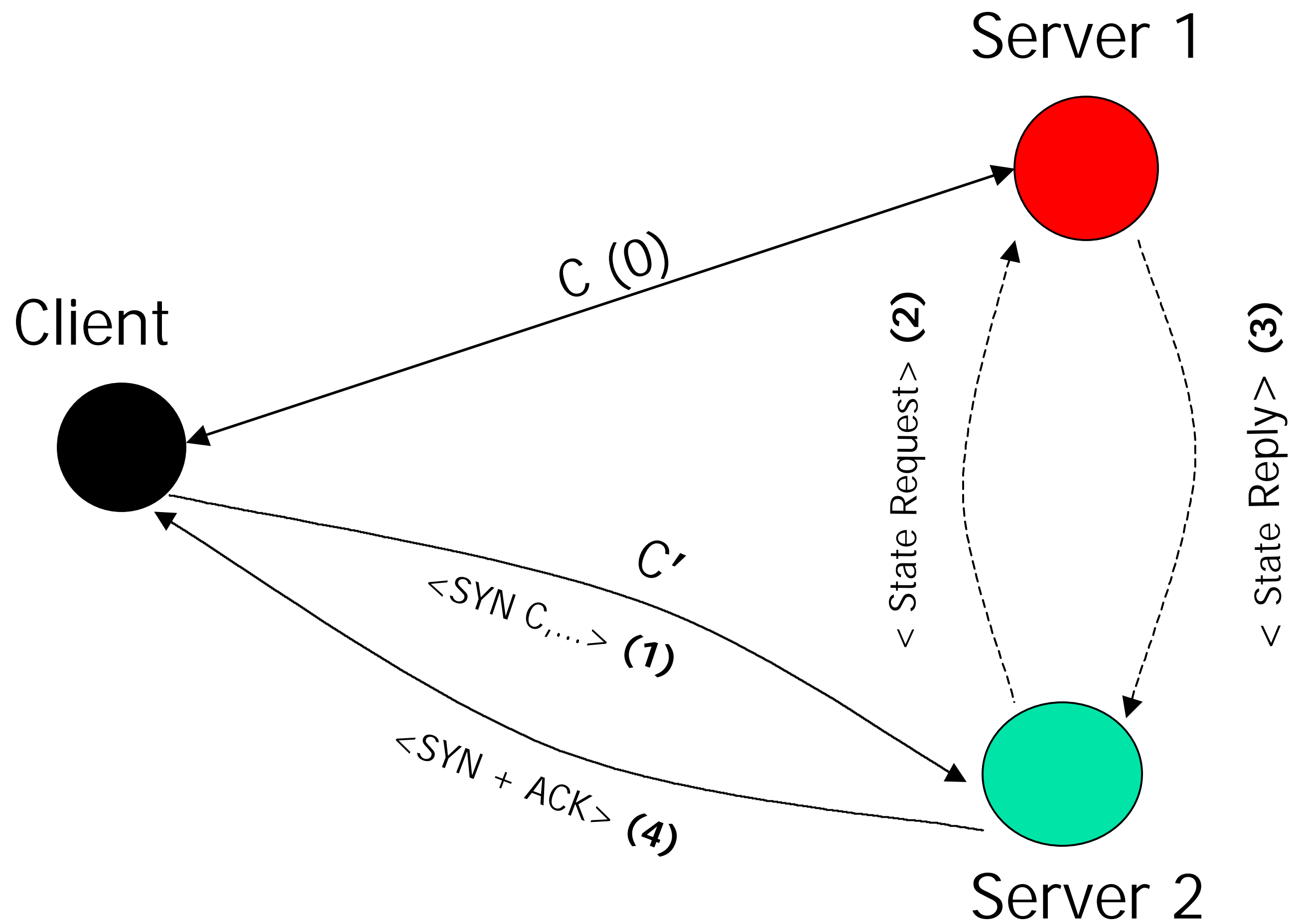
# Log-Based State Synchronization

- Logs are maintained in the protocol
  - discarded at export time (state is sync'ed)
- Logs are part of the state to be transferred during migration
- Service resumes from the last state snapshot and uses logs for execution replay

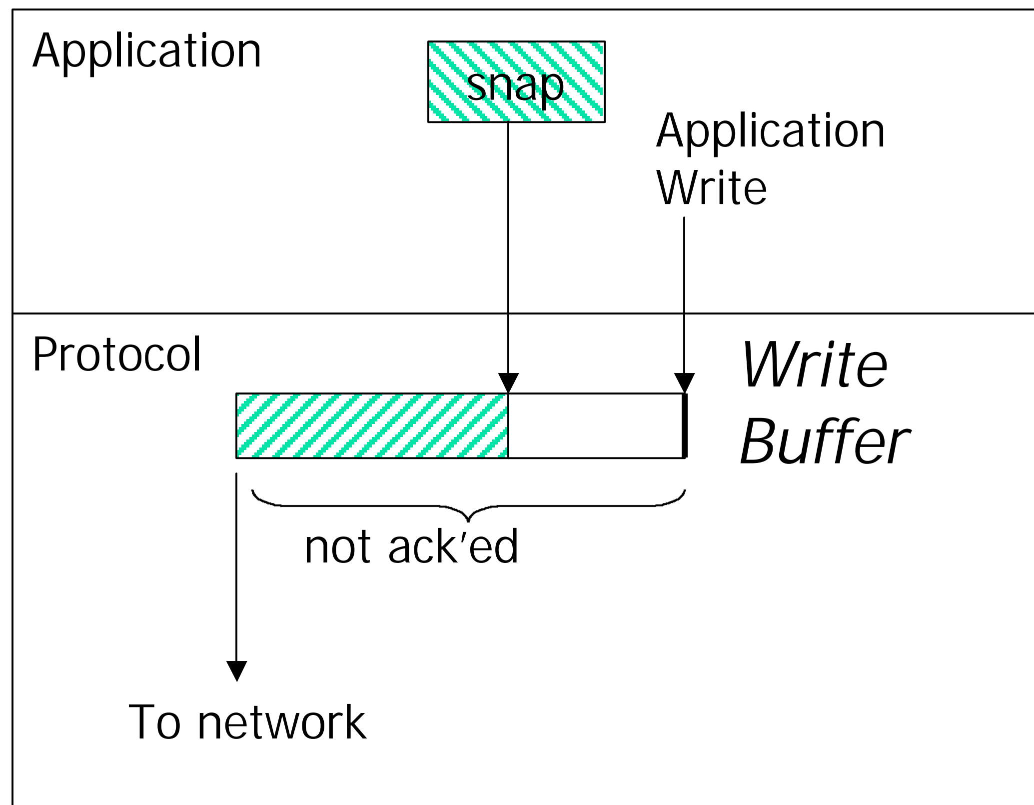
# Implementation

- Modified the TCP/IP stack in FreeBSD kernel
- Lazy connection migration mechanism
- Evaluation on synthetic benchmarks
- Target applications
  - PostgreSQL front-end
  - Streaming server

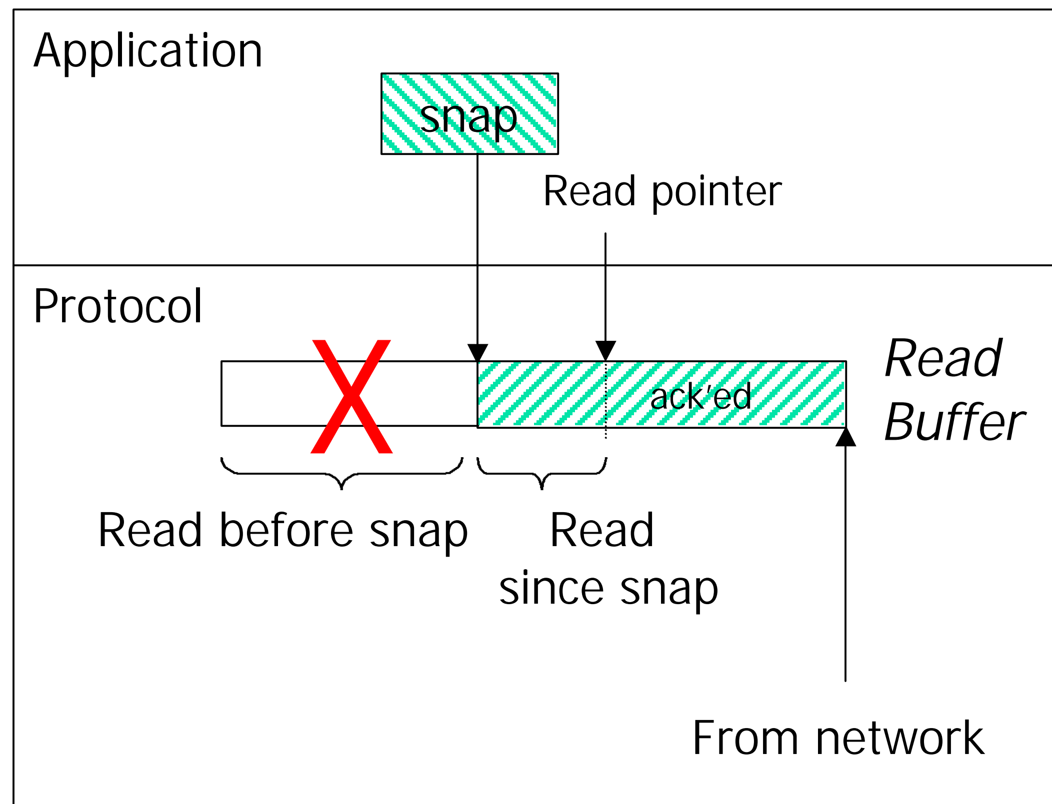
# Lazy Connection Migration



# Implementation: Per-Connection State



# Implementation: Per-Connection State



# Preliminary Performance Evaluation

- Experimental setup
  - Two servers, one client: P II 400MHz, 128 MB RAM
  - Servers connected by dedicated network link
- Benchmarks
  - Microbenchmark
  - Simple streaming server application

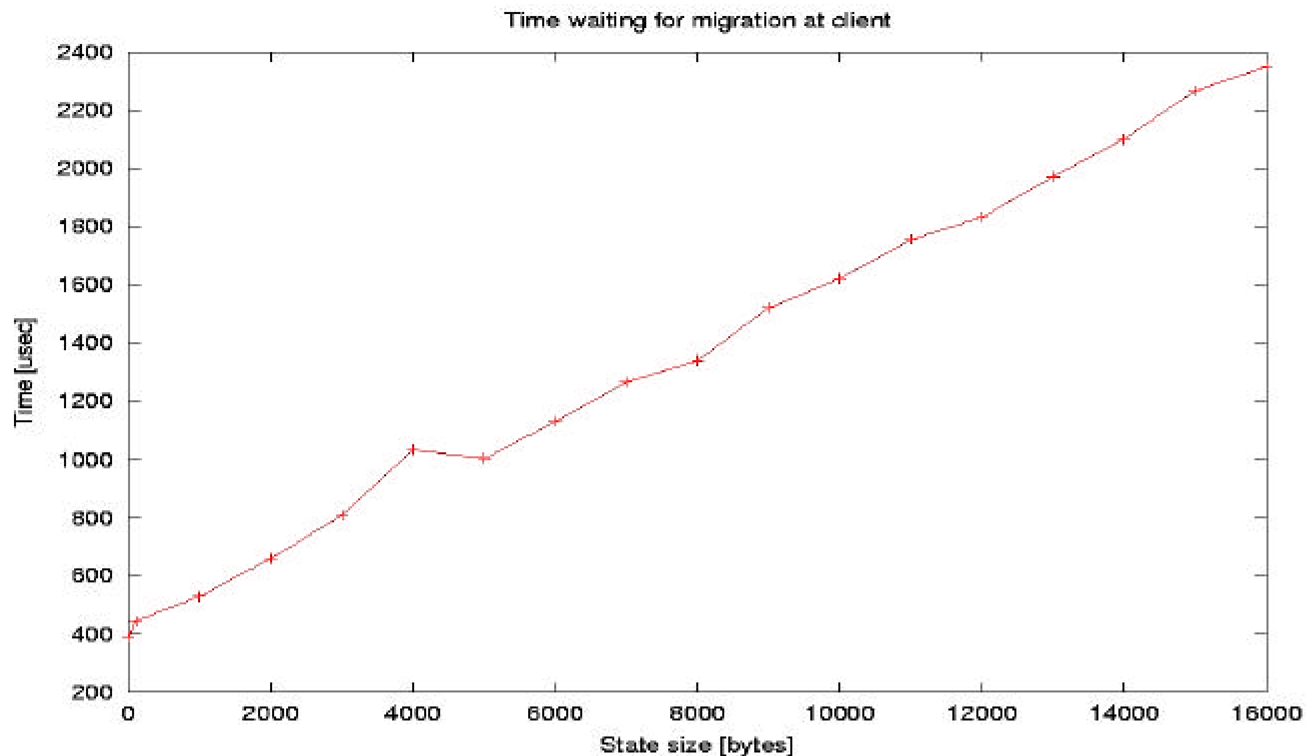
# Microbenchmark

- Breakdown of migration times for one-way migration averaged over 200 runs

| State size [kB] | Client migration phases [us] | | | Server 2 state handling [us] | | | Server 1 state handling [us] |
|---|---|---|---|---|---|---|---|
| | Initiate | Wait | Complete | Prepare request | Wait reply | Reinstate state | Prepare reply |
| 0 | 73 | 387 | 10 | 40 | 191 | 20 | 50 |
| 5 | 87 | 1002 | 7 | 42 | 784 | 25 | 89 |
| 10 | 90 | 1621 | 7 | 42 | 1382 | 27 | 120 |

# Microbenchmark

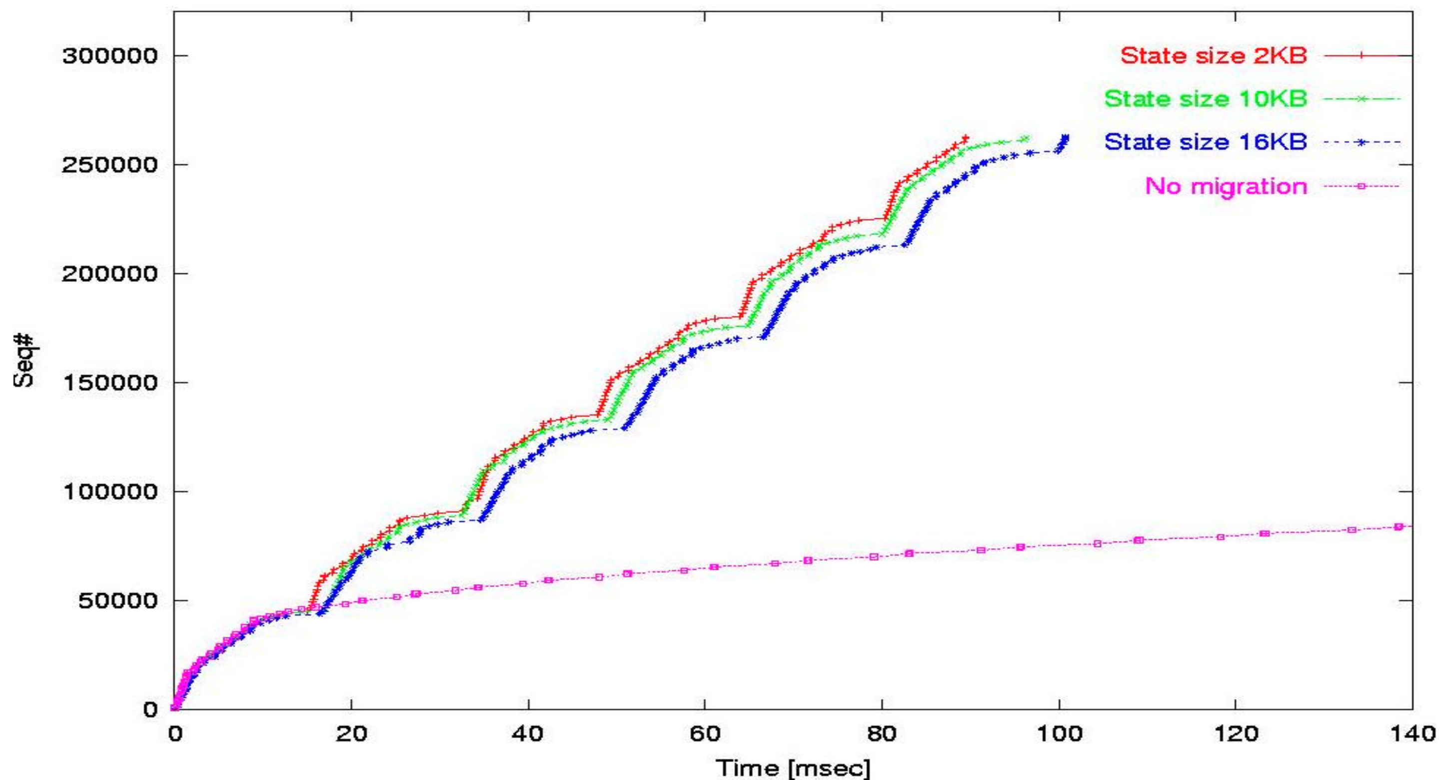
Endpoint switching time vs. state size



# Streaming Server Experiment

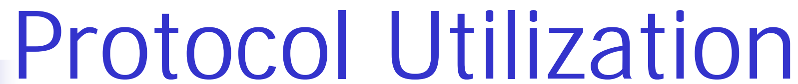
- Server streams data in 1 kB chunks
- Server performance degrades after sending 32 kB
  - emulated by pacing sends in the server
- Migration policy module inserted in the client kernel
  - Metric: inbound rate (smoothed estimator)
  - Migration trigger: rate drops under 75% of max. observed rate

# Stream Server Experiment




Effective throughput is close to the average rate seen before server performance degrades

# Protocol Utilization

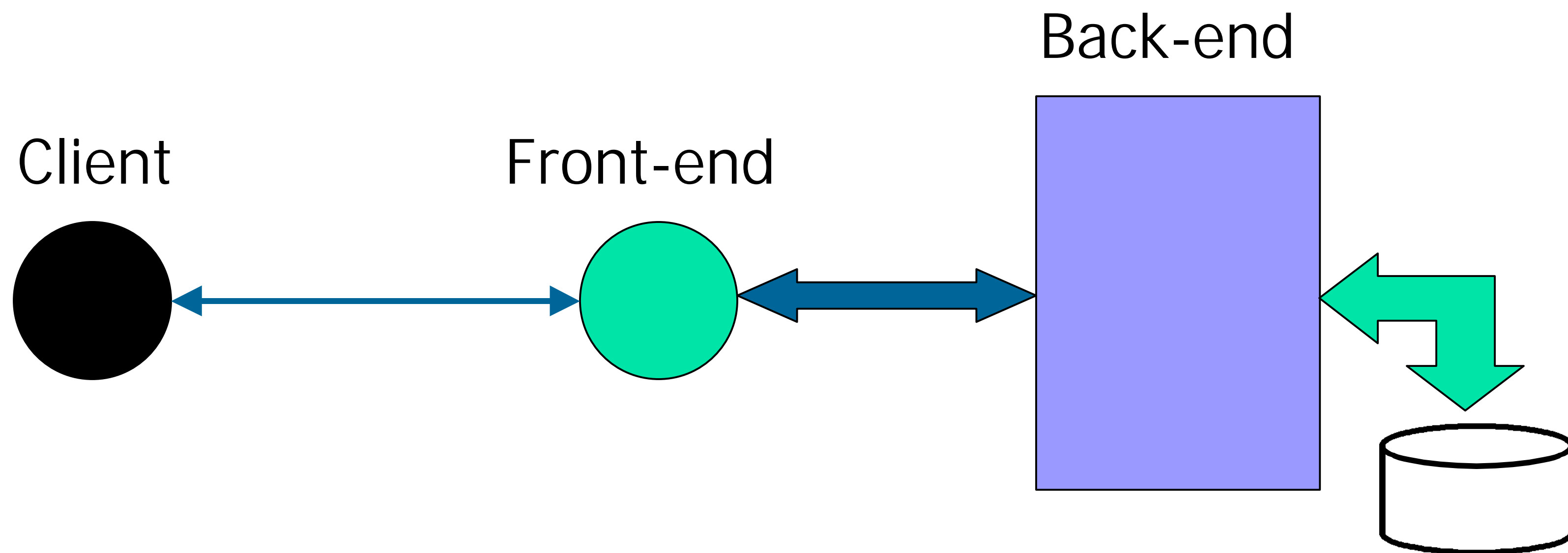


- For high availability
  - applications with long-lived connections
  - critical applications (banking, e-commerce etc.)
- For performance through load balancing
  - migration trigger: at server side, based on load balancing policy
- For fault tolerance
  - eager transfer of connection state

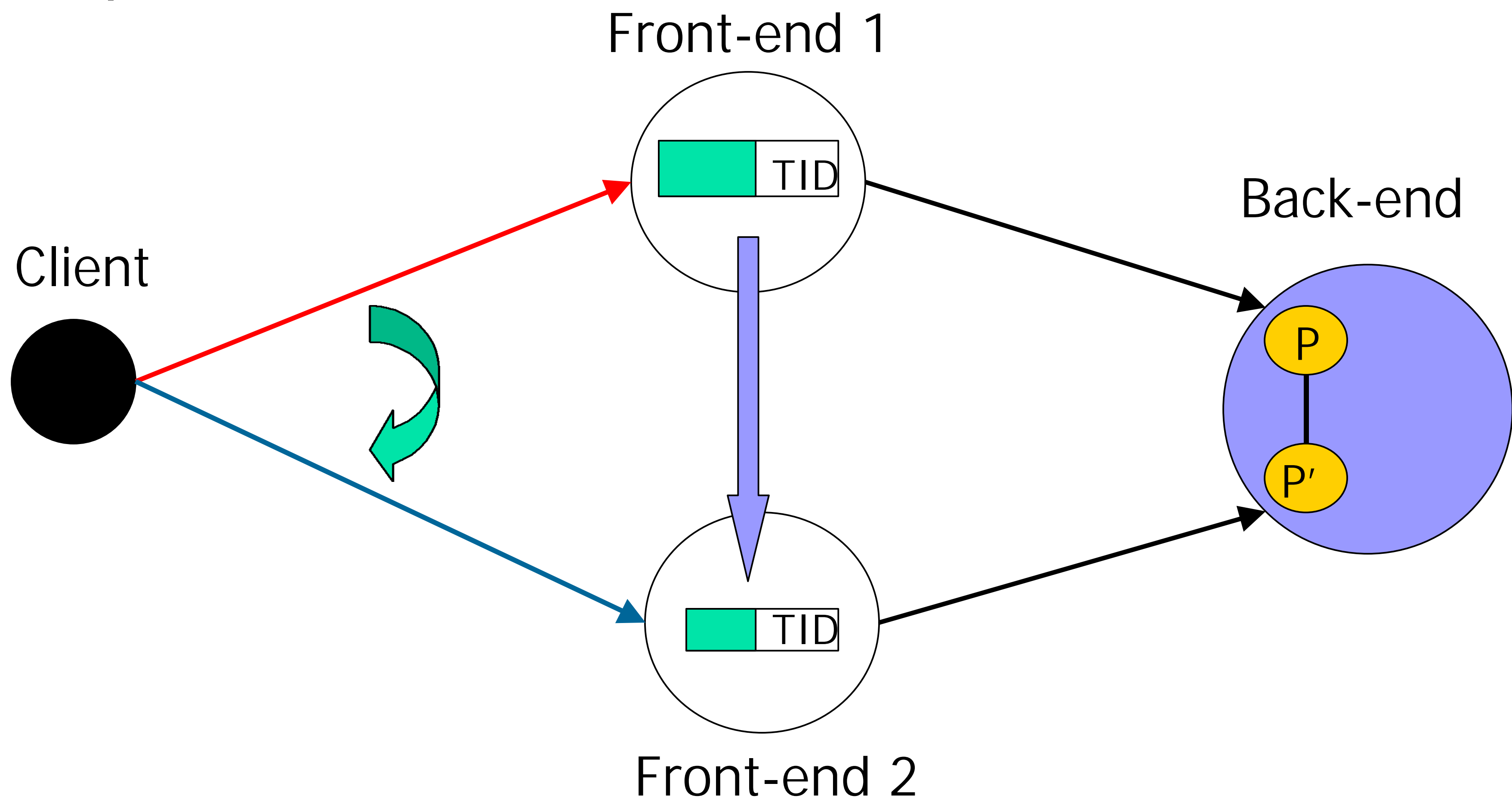
# Highly Available Database Service

- Access a transactional database over Internet
- System architecture
  - front-end host
    - interfaces with clients, e.g. via HTTP
    - translates a service request into queries to native database
    - retrieves/processes results and replies back to client
  - back-end host
    - manages data repository
- Problem: front-end availability

# System Architecture



# Migration-Enabled Front-Ends



# Issues for FE Migration

- Transactional DB
  - must preserve ACID semantics across migration
- Determinism
  - front-end execution has persistent side-effects (DB)
  - do not re-execute at FE2 transactions committed at FE1
- May need migration support in the back-end
- Migration granularity
  - FE discipline: when to take state snapshots?

# Granularity of Migration

- Extent of application execution that could be replayed at FE2 after migration
- Upper bound: one transaction
  - FE discipline: export state *at least* every transaction
- Lower bound: application dependent
  - smallest unit: one query
  - tradeoff: work lost/redone vs. overhead

# Fine-grained (Query-level) Migration

- Problem: when resuming, results of past queries may be needed
- Solution 1: log results at back-end
  - potentially huge! E.g. SELECT * FROM table
  - log management is a problem
- Solution 2: re-do past queries
  - requires partial rollbacks inside a transaction: DB must support *savepoints*
  - a front-end takes a snapshot before a savepoint

# Our Implementation: PostgreSQL

- Migration granularity: transaction-level
  - no support for savepoints in PostgreSQL
- FE discipline
  - FE1 takes snapshots before final query in transaction (COMMIT/ABORT) to make it atomic w.r.t. migration
  - snapshots include transaction ID tid
  - FE2 retrieves the outcome (committed/aborted) of tid from back-end and decides how to proceed
- Changes to back-end to support FE2 reconnect

# Example: Migratory PostgreSQL FE

```
if (import_state(conn, &tid, &last)) { // a migrated connection
  reconnectDB(tid, &laststatus)
  send(conn, laststatus)                // report outcome of last txn at FE1
  goto last + 1
}

N:                                      // start of txn N
  tid = txnBEGIN()
  last = N
   ... do work for txn N ...
  export_state(conn, tid, last)
  status = txnCOMMIT()
  send(conn, status)                    // report outcome of txn N to client
```

# Prototype: Migratory PostgreSQL

- Extended the DB API to enable migration
- Integrated a migration-aware PostgreSQL front-end with Apache web server
- Client sends service requests via HTTP
  - request: execution of a sequence of transactions
- Client connection can transparently migrate to another front-end
  - receive results and/or continue execution
  - ACID preserved; no unwanted side-effects

# Related Work

- HTTP server fail-over by connection migration [Snoeren '00]
  - soft TCP and HTTP state maintained at back-up servers
- Fault-tolerant TCP [Alvisi '00]
  - failure masking using TCP wrappers
  - persistent connections across server crashes
- Stream Control Transmission Protocol (SCTP) - RFC 2960
  - multi-homing (many IP addresses / endpoint) ensures connectivity in the face of network failure

# Related Work (cont'd)

- MSOCKS [Bhagwat `98]
  - proxy based "TCP splicing" for mobile clients with multiple interfaces
- Indirect TCP [Bakre '95]
  - connection handoffs between MSRs allow mobile hosts to maintain open connections with a fixed host

# Conclusions

- Transport layer protocol that enables building highly available services
  - dynamic, light-weight, transparent connection migration
- Migratory API for server applications
- Migration architecture: decouples mechanism/policies
- Working prototype in FreeBSD
- Application: transactional database access over WAN
- http://discolab.rutgers.edu/mtcp

# Future (Current) Work

- More performance evaluation
- Investigate migration policies
- Alternate state transfer implementations
- Recursive connection migration mechanism
- Other applications