MIGSOCK

A Migratable TCP Socket in Linux

Bryan Kuntz
Karthik Rajan

Masters Thesis Presentation
21st February 2002

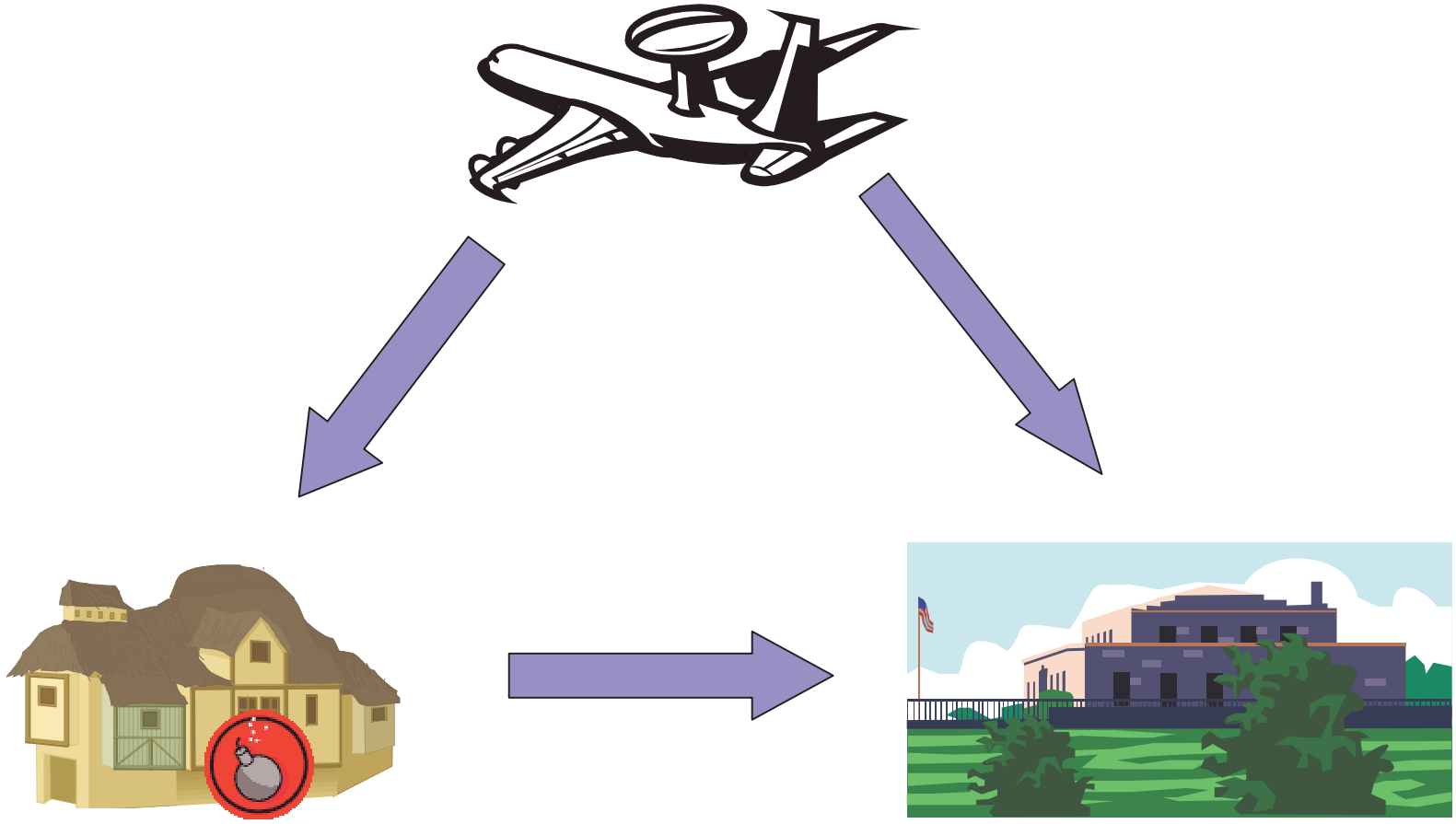
Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Introduction

- What is process migration?
- What is socket migration?
- What is MIGSOCK ?

Scenario 1



Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Process Migration

- What is it?
  - the act of transferring a process between two machines during its execution
- More specifically:
  - Capture the state of a running process at a given moment
  - Transfer this state information to the new host of the process
  - Resume the process execution from the point at which it was captured

Process Migration Importance

- Load balancing
- Fault tolerance
- Anytime system administration
- Data access locality

Process Migration Difficulties

- Places limitations on the architecture – similar computers, similar libraries, etc.
- Complexity of adding transparent migration to systems
- Overheads / performance

Types of Solutions

- Operating System level support
  - Use a standard OS and modify kernel for migration
    - MOSIX, CRAK, SPRITE
  - Use/code a custom OS better suited for migration
    - MACH microkernel architecture
- User level support
  - Application re-link and re-compile
    - CONDOR
  - Programming environment – Java libraries
    - NOMADS

Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Introduction

- What is a Socket ?
  - OS interface that abstracts TCP & UDP over IP
  - Behaves like a file to the user application
  - A two-way connection between two hosts
  - Sockets identified by

    `(hostIP, hostPort, destinationIP, destinationPort)`

OSI Model

Application	7
Presentation	6
Session	5
Transport	4
Network	3
Data Link	2
Physical	1

Migration vs. Handoff

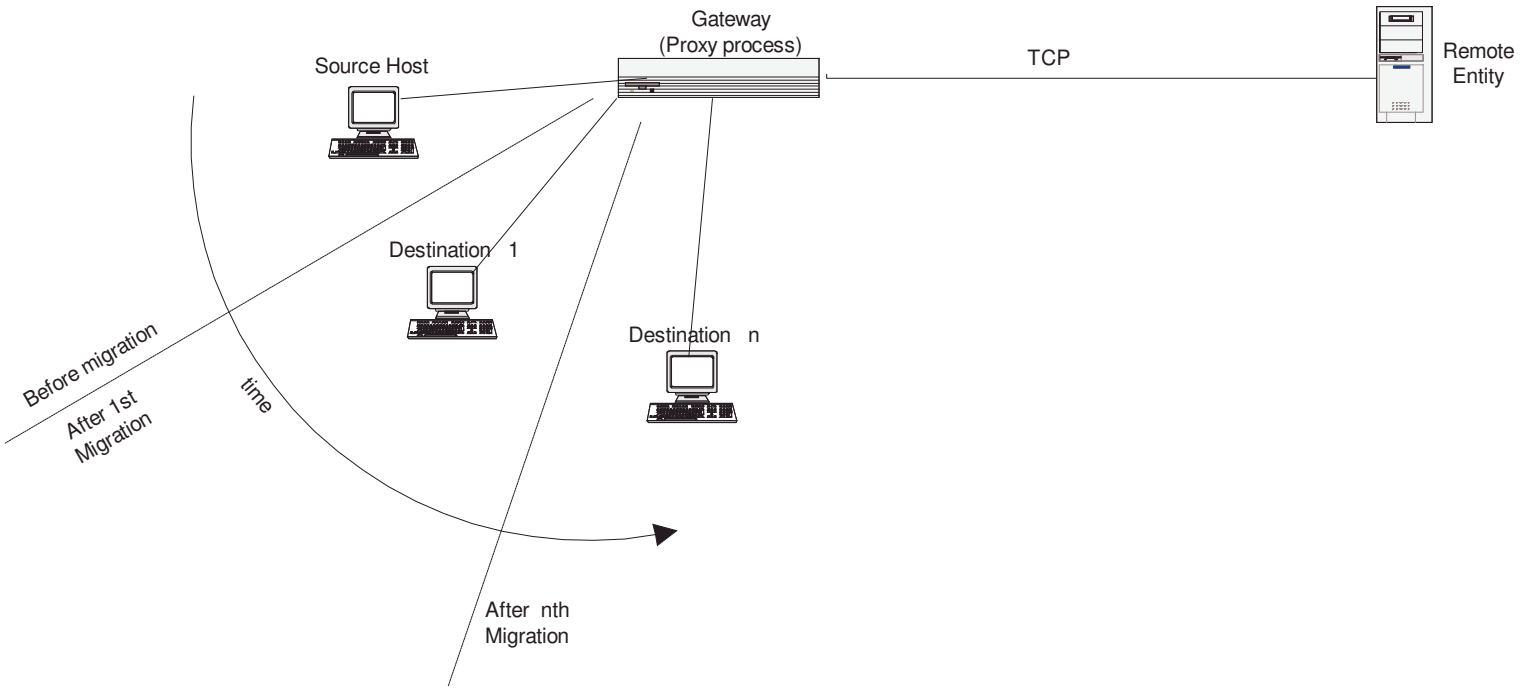
- Socket Migration
  - Involves transferring the end-point of the connection that is migrating with the process
  - Preserve the connection state
  - Important use: Allows processes with open sockets to migrate
- Socket Handoff
  - Transfer only the socket state, not the process
  - Requires some cooperation from the new socket host
  - Important use: Enables processes to maintain execution during a network reconfiguration.

    Ad-hoc private networks that suddenly go online, for example.

Migration Solutions – Proxy

- Migration within the proxy network using NAT
- Proxy Knows Current Location (IP & port) of process
- Advantages
  - Easy to understand and implement
  - Stateless
  - Avoids changes at the OS level
- Disadvantages
  - Limited Migration
  - Proxy becomes single point of failure
  - Hard to maintain TCP end-to-end semantics
- Examples
  - MSOCKS, “TCP Handoffs for Clustering”

Proxy Method



Migration Solutions – User Level

- Layer 7 of the OSI model – Application layer
- Wrap socket library with migratable API
- Use migratable API in user applications
- Advantages
  - Simple to implement
  - No kernel changes
- Disadvantages
  - Applications need to be re-written – not transparent
  - Slow as low level sockets are established again
- Examples
  - MOCKETS

Types of Solutions – OS Level

- Layers 4 and below of the OSI model – TCP and IP
- Modify TCP directly
- Support at the kernel level
- Advantages
  - Fast
  - Potentially transparent – depending on where control lies
- Disadvantages
  - Both hosts should implement TCP stack
  - Kernel changes require special permissions
- Examples
  - Our Design ☺

Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Our Solution: Design Goals

- Transparency
- Kernel level support
- Compliance with TCP semantics
- Modular implementation
- Performance
  - No serious overheads
  - Minimize message exchange
- Ease of integration into existing process migration environments

Assumptions

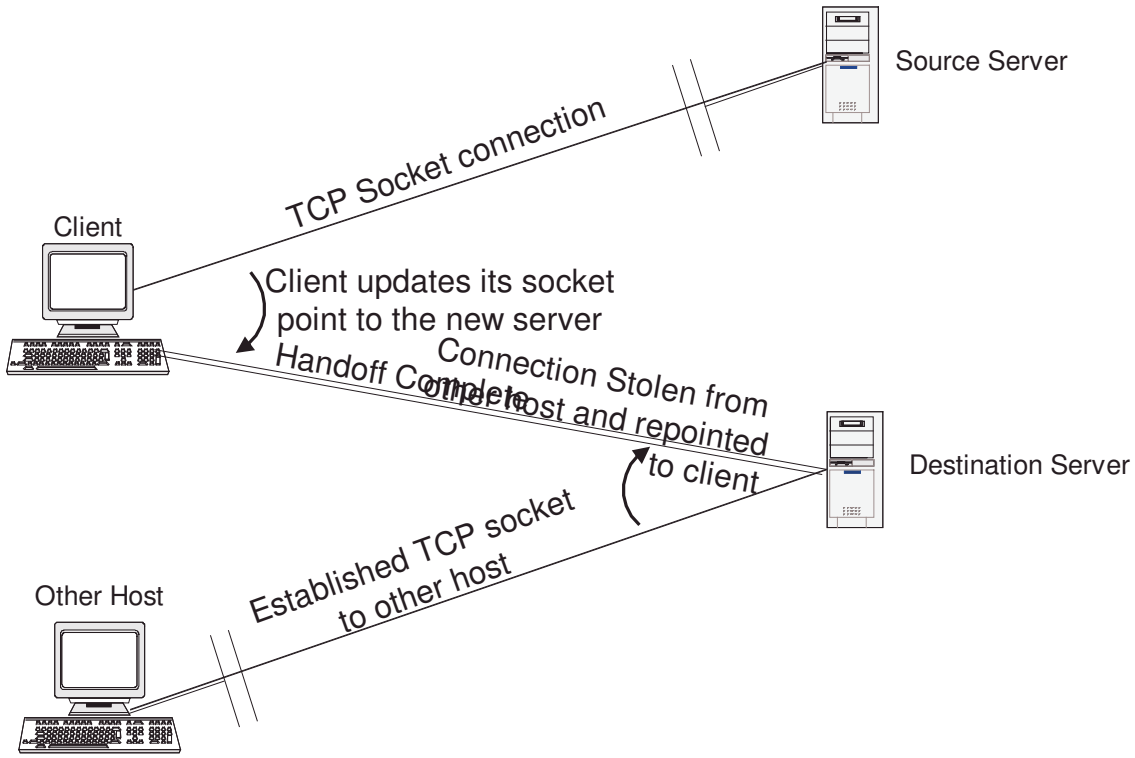
- Kernel support at all participating hosts
- Minimal security measures / not our primary concern
- A process migration system is already in place, otherwise steal a socket
- Upper layer state consistency
  - Socket handoff requires a connected state
  - Post-migration attempts by the remote host to reconnect to the original server IP and port will not succeed.
- Unavoidable latency during migration
- Shared file system (NFS) or some other way to transfer state

Socket Handoff

Migrate only the socket connection; not the process.

1. Suspend an established connection and capture the socket state.
2. Transfer this state to a host with a running process.
3. This process must be running the same application layer, using a socket descriptor that is connected.
   - In most applications, this will require *STEALING* a socket that is already in the connected state.
4. Replace this socket's state with the transferred state.
5. Resume the connection with the remote host.

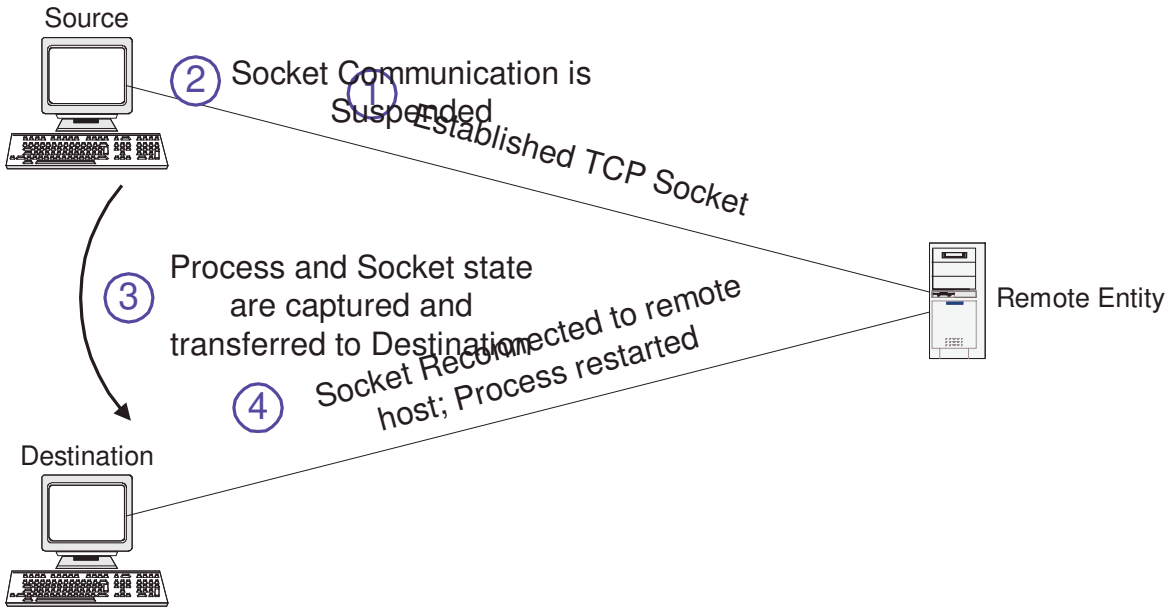
Socket Handoff Scenario



Socket Migration

1. Socket migration is initiated at source host
2. TCP message is sent to remote host suspending his side of the connection
3. Socket state is gathered and written to a file
4. State is transferred to the destination host
5. Socket is recreated with new host and port information of the destination host, and connected to remote host
6. TCP message is sent to remote host updating his half of the socket and resuming his side of the connection

Socket Migration Scenario



MIGSOCK API

- Send_Migration_Request(pid, sockfd)
- Serialize_Socket_Data (pid, sockfd, buf)
- Deserialize_Socket_Data (pid, sockfd, buf)
  - For socket handoff
- Deserialize_And_Restart(pid, sockfd, buf)
  - For socket migration
- Send_Remote_Restart (pid, sockfd, msg_contents)
- Get_Local_Host (pid, sockfd)

Usage Example

- Source side algorithm

```
// Obtain the pid and socket fd for the migrating socket.
Send_Migration_Request(pid, fd)
num_bytes = Serialize_Socket_Data (pid, sockfd, buf)
write(somefile, dest_buf, num_bytes);
// Transfer somefile to destination host
```

- Destination side algorithm

```
// Create a socket using the socket() call for the process
// Obtain the pid and socket fd for the process and newly created socket.
read(somefile, buf);
newport = Deserialize_And_Restart(pid, sockfd, buf)
newhost = Get_Local_Host(pid, sockfd)
// Add the newport and newhost to msg_contents
// Get oldhost and oldport from buf and add this to msg_contents
Send_Remote_Restart(pid, sockfd, msg_contents)
```

Migration Initiation Request (Source Host)

1. Suspend local process
2. Flag set in *struct sock* tells packet-send routine that migration is under way.
3. Special TCP message generated over the socket
4. Remote host sets a flag in *struct sock*
5. Remote host unimpeded unless he tries to write (send()) to the socket, in which case he blocks
6. Capture and record socket state, return it in a buffer to the user

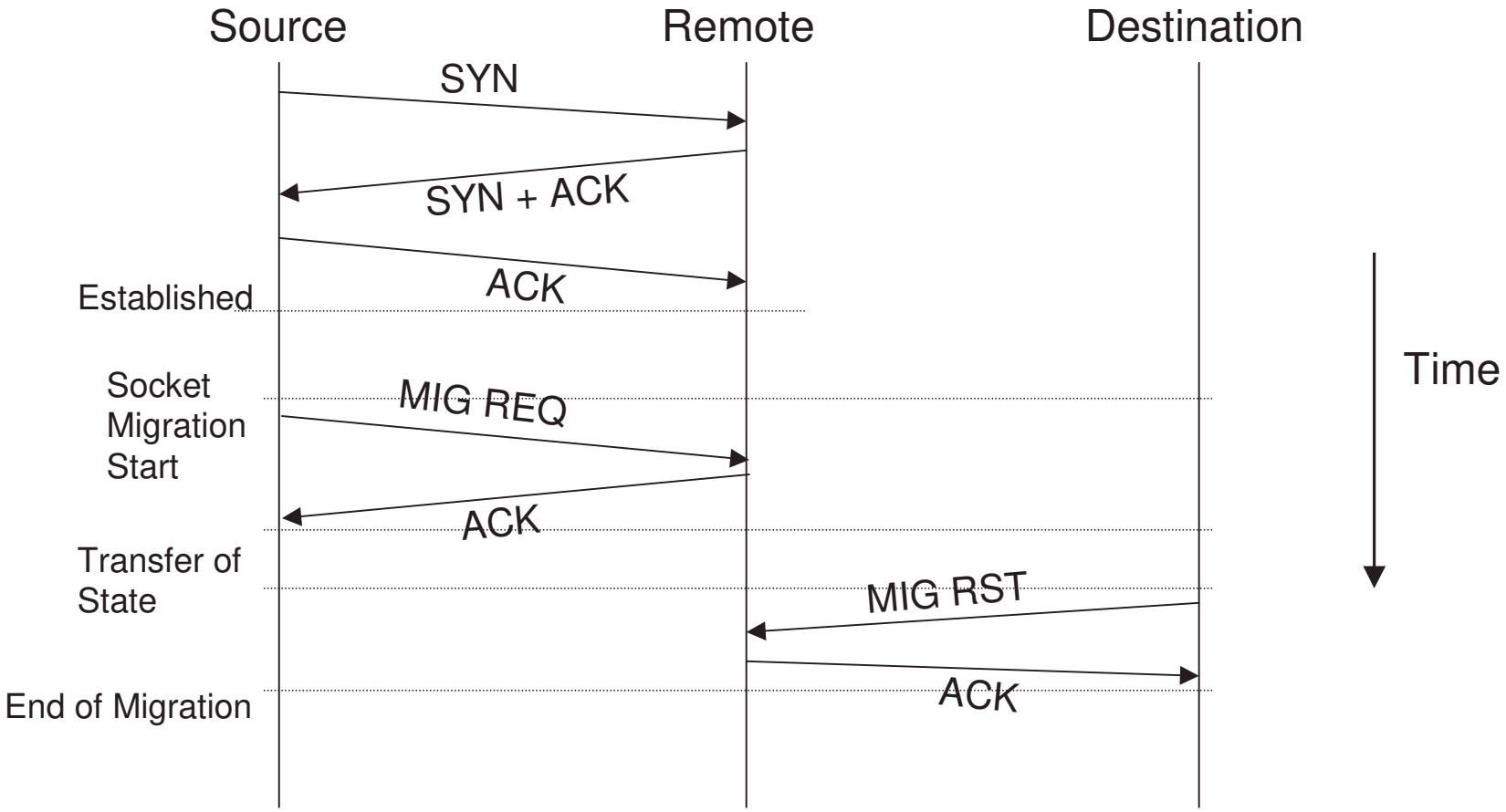
State transfer

1. The user program should write the buffer to a file or a network message
2. Send the file or message to the destination host
3. The data in this buffer should not be modified in any way or it could become unreadable by the API

Migration Completion (Destination Host)

1. Obtain socket descriptor that is used by the application layer in an appropriate state
2. Call a deserialize function to construct a connected socket – it returns a local port number
3. User constructs the restart message by adding the new and old ports and IP addresses
4. Special TCP message generated to remote host
5. Remote host resets the migration flag, repoints the socket to the new destination, and signals any blocking send() calls.

Message Timeline



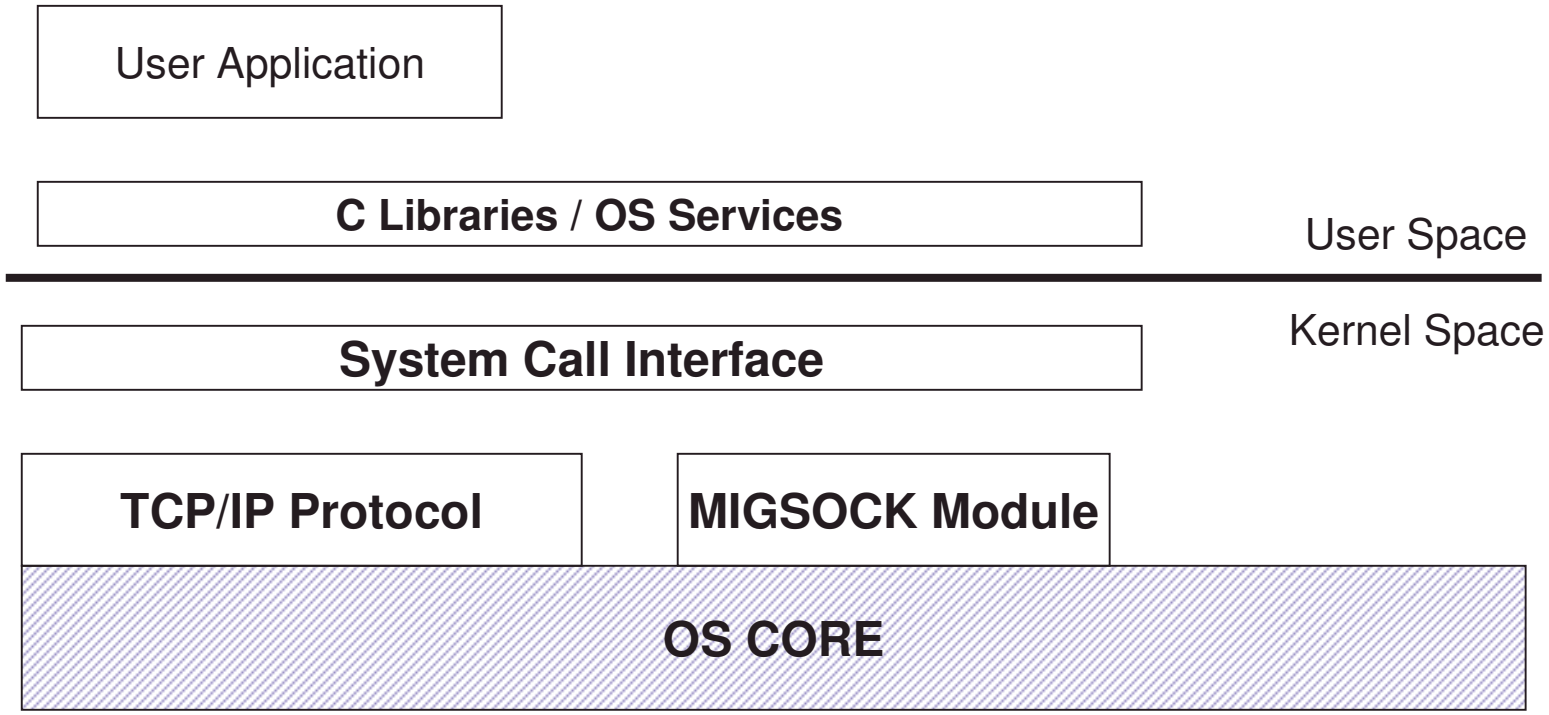
Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Implementation

- Modules
- Kernel function changes
- Kernel data structures
- Message format

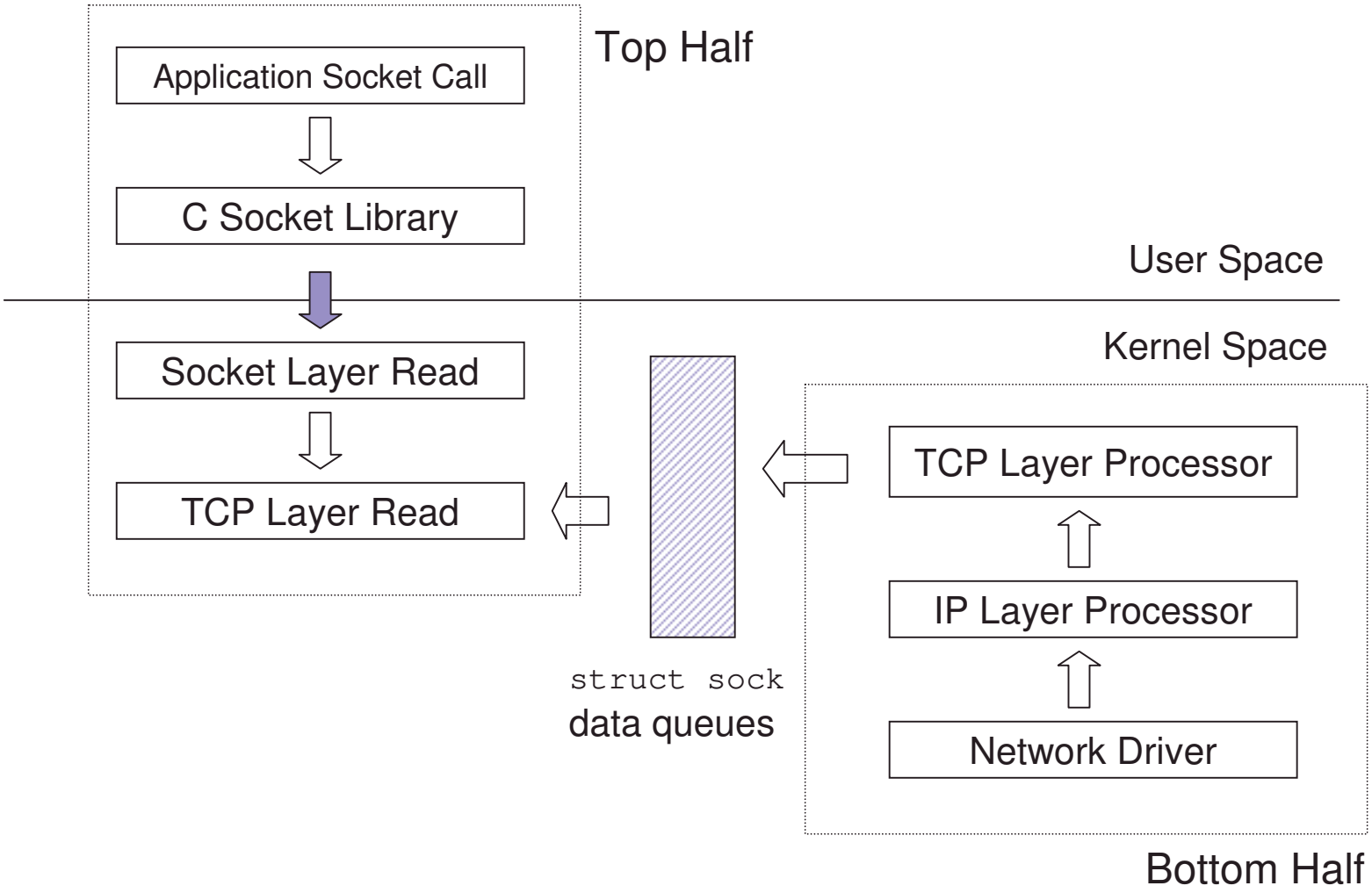
Migsock Architecture



Modules

- Syscalls implemented using the device ioctl() interface
- Requires the creation of a device /dev/migsock
- Most functionality is contained within the modules except for the ***bottom-half*** side of the kernel TCP implementation
- Migratable socket option can be selected / deselected in the kernel configuration menu

Kernel Flow (Socket Read)



Kernel Function Changes (TCP)

- The ***bottom-half*** portion of the TCP functionality needs to be changed to
  - Send specially flagged migration request messages – source and destination
  - Prevent the passing of migration messages to the upper layer (application) – remote
  - Update IP address and port information after migration – remote

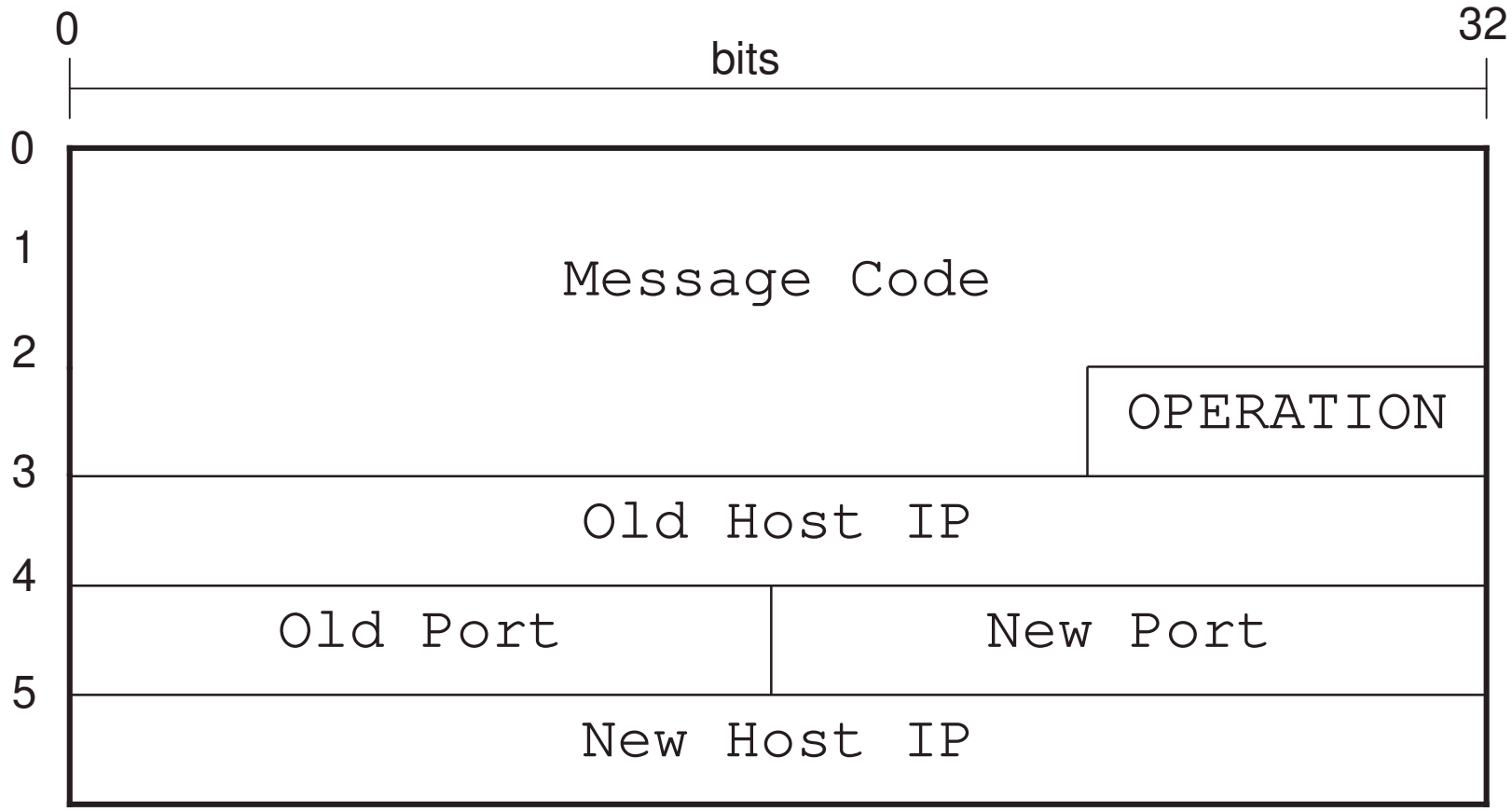
- Achieved by changing the following source files:

`tcp_input.c, tcp_output.c, tcp_ipv4.c`

Kernel Data Structures

- struct sock
- struct socket
- struct tcp_opt
- struct file
- struct task_struct

Message Format



Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Testing and Demo

- Handoff
  - Handoff one end of a live socket from a process on a host to another host's process
- Migration
  - Migrate a program along with its TCP socket from one host to another

Integration with CRAK

- MIGSOCK aware CRAK
  - Source side – prevent CRAK from aborting
  - Destination side – recreate socket entity
- Serialize socket state before checkpointing process
- Process restarts only after sockets are restored
- Change scheduling to account for delay in socket restoration

Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Impact

- Flexible and easy to integrate into any system level process migration schema – demonstrated with CRAK
- Implementation is kernel level and modular
- Low messaging overheads
- Transparent to running processes
  - Any binary program compiled for Linux is eligible for socket migration
- Control can be integrated or separate from migrating application
- Maintained TCP semantics
- Modular implementation

Future Work

- Build on a security infrastructure
- Reconstruct a listening socket
- Improve compatibility with non-migration hosts
- Port it to other Kernels
- Merge with the official Linux kernel source tree
  - Submit patch to LKML and Linus
- Integrate with other process migration systems
- Add a state-transportation mechanism – remove responsibility from user or shared file system
- Initiate an RFC ☺☺

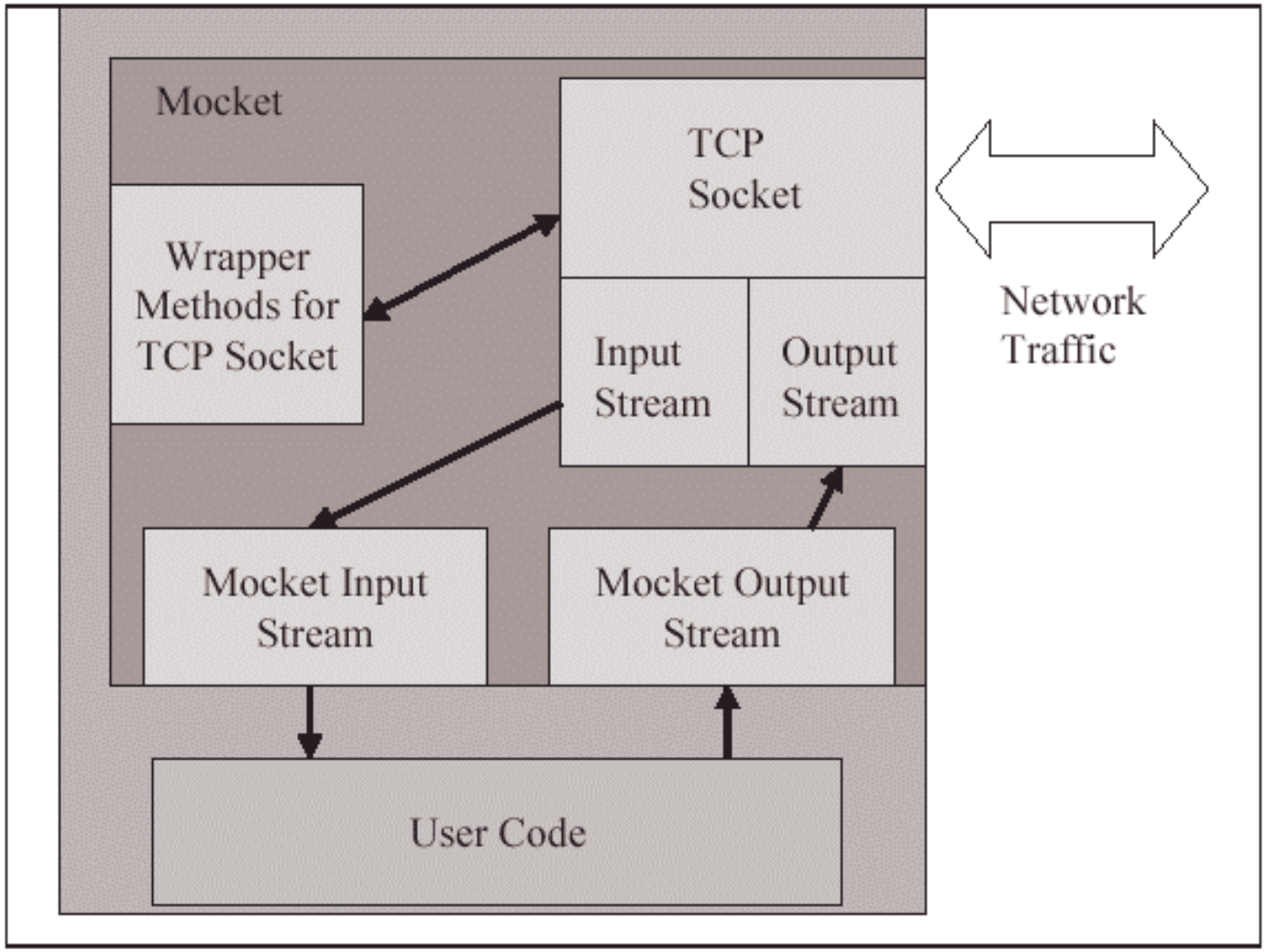
Agenda

- Introduction
- Process Migration
- Socket Migration
- Design
- Implementation
- Testing and Demo
- Impact and Future Work
- Related Work (time permitting)
- Demo (at NSH 1604)

Mockets

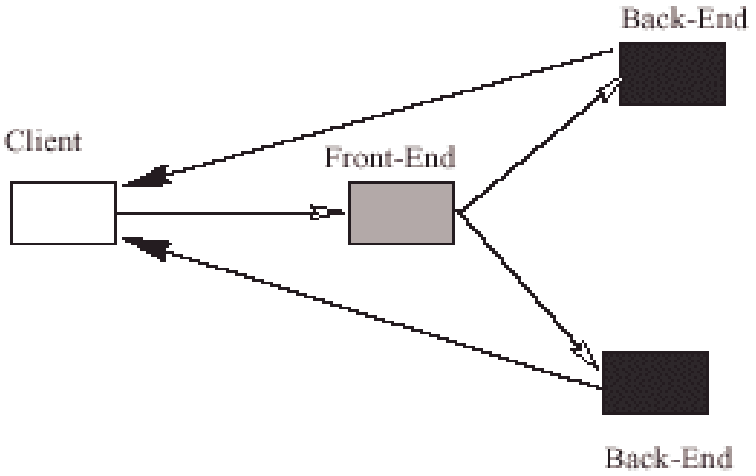
- Part of the NOMADS project from the University of West Florida
- User level socket migration
- Provides API on top of Java sockets
- Abstracts the re-connection process
- Each Mocket uses a Java socket
- Specially coded applications
- Large overheads

Mockets Structure



TCP Handoff

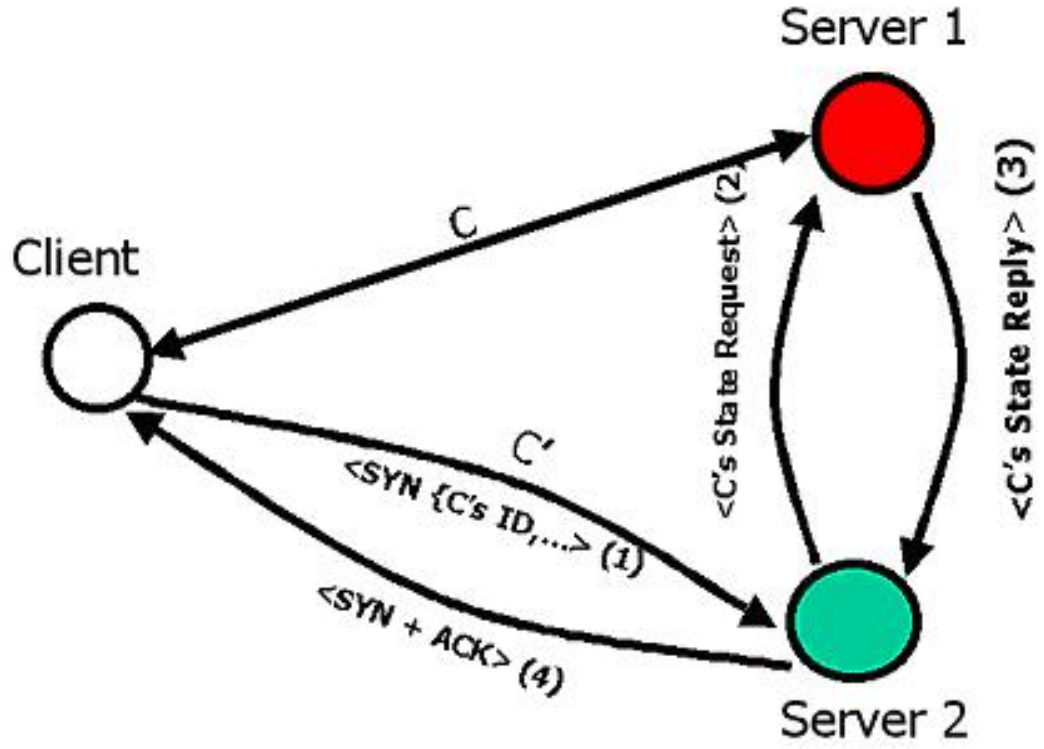
- From HP Labs and Michigan State Univ.
- Hand-off one end of a live socket connection
- Typical use in HTTP web clusters
- Client is unaware of the handoff
- How is it different from ours ?
  - One-way handoff only (ours is two way)
  - Client protocol stack can remain unchanged



M-TCP

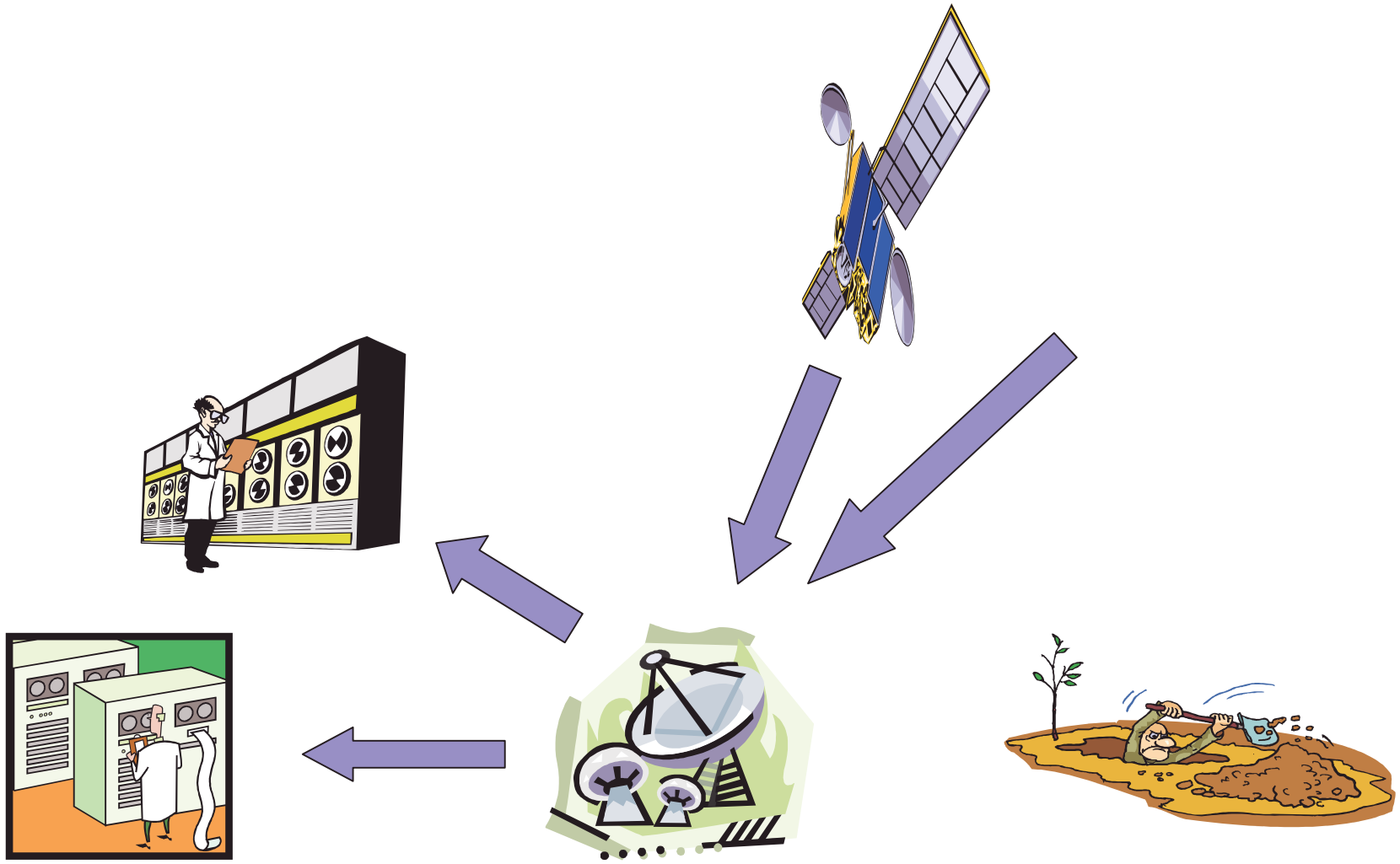
- Rutgers University
- Meant for Clustered Service Providers
- Decouples service provider from service
- Servers keep connection-specific state
- Lazy transfer of client connection state
- How is it different from ours ?
  - Client Initiated migration
  - No client side migration
  - Transfer takes place AFTER SYN is sent, leading to a possible performance weakness

M-TCP State Transitions



CRAK

- Columbia University
- No support of Socket Migration – Aug 01
- Method described – Jan 02
- No binaries or source code released to prove it!
- Proposed solution:
  - User level (application layer) calls
  - Remote host can still communicate during migration
    - Data is retransmitted / lost during this process
  - Extra overhead due to user level calls
  - Restore slower than our implementation:
    - CRAK = SSH
    - MIGSOCK = Existing socket

Questions?

Scenario 2