Transparent TCP-to-SCTP Translation Shim Layer

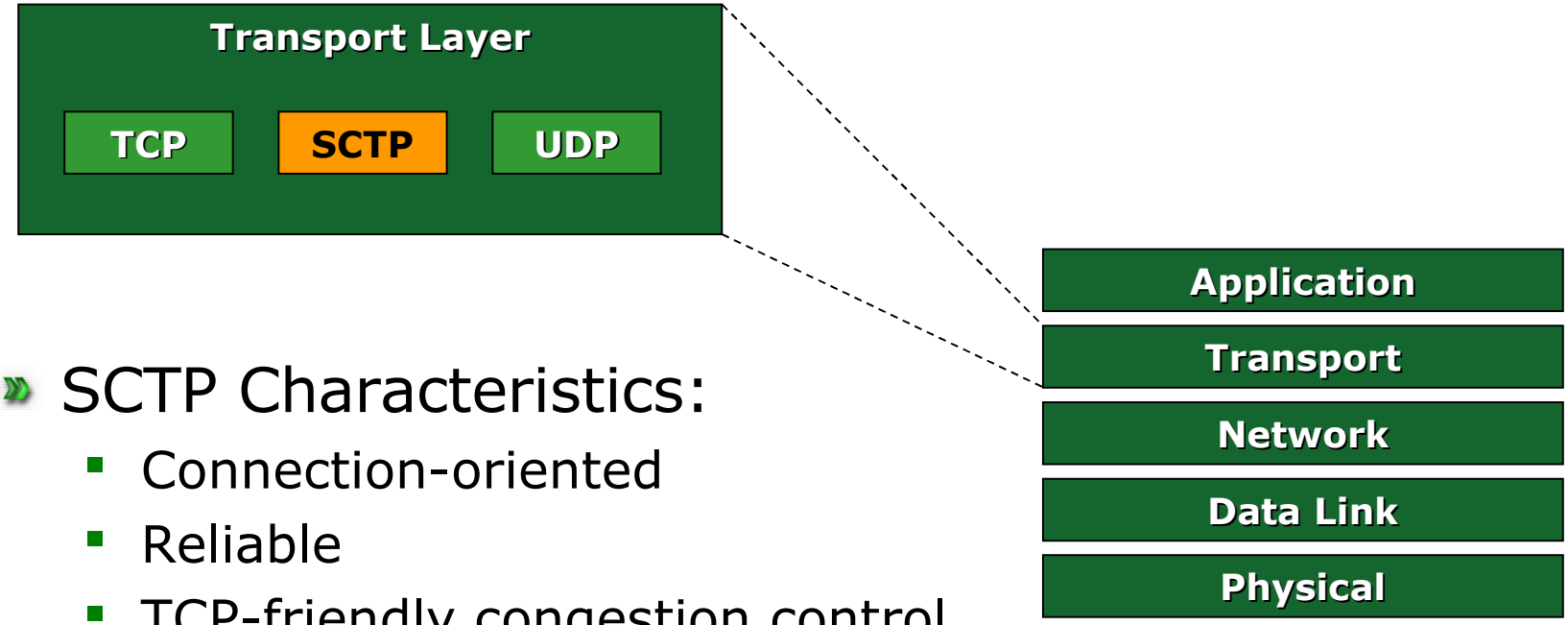
EuroBSDCon 2007 / Copenhagen, Denmark

Ryan Bickhart

ryan.bickhart@gmail.com

Protocol · Engineering · Laboratory
University of Delaware

SCTP: Stream Control Transmission Protocol



- SCTP Characteristics:
  - Connection-oriented
  - Reliable
  - TCP-friendly congestion control
  - Message-based
  - Partial reliability extension
  - Multistreaming ability
  - Multihoming support

For SCTP protocol specifics, see RFC 2960

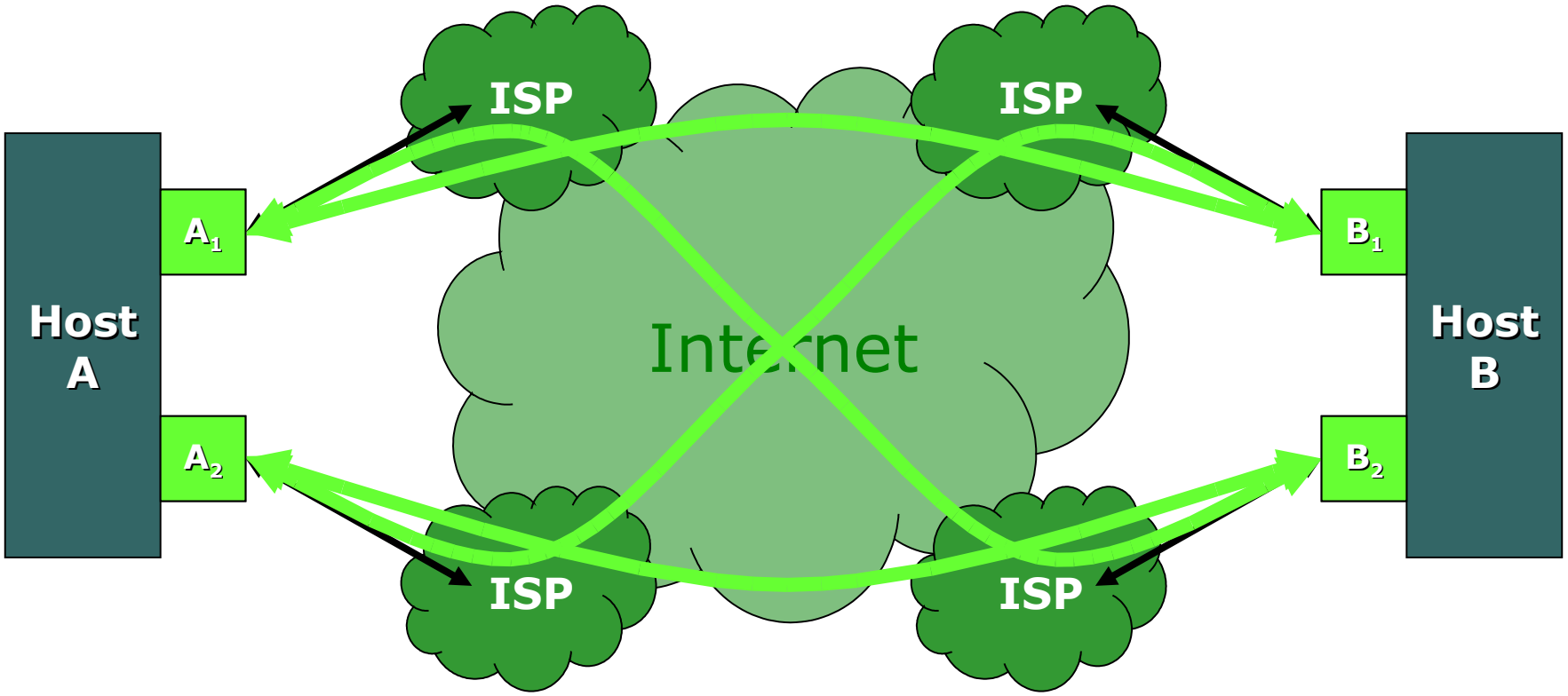
Shim Concept Explained

- TCP-to-SCTP translation: kernel will map calls to TCP to equivalent calls to SCTP
- Transparent: applications will not be aware the TCP-to-SCTP translation is even happening – kernel will trick them
- Shim layer: decision logic to control SCTP use will be inserted into existing kernel

Outline

- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

Multiple Addresses in TCP

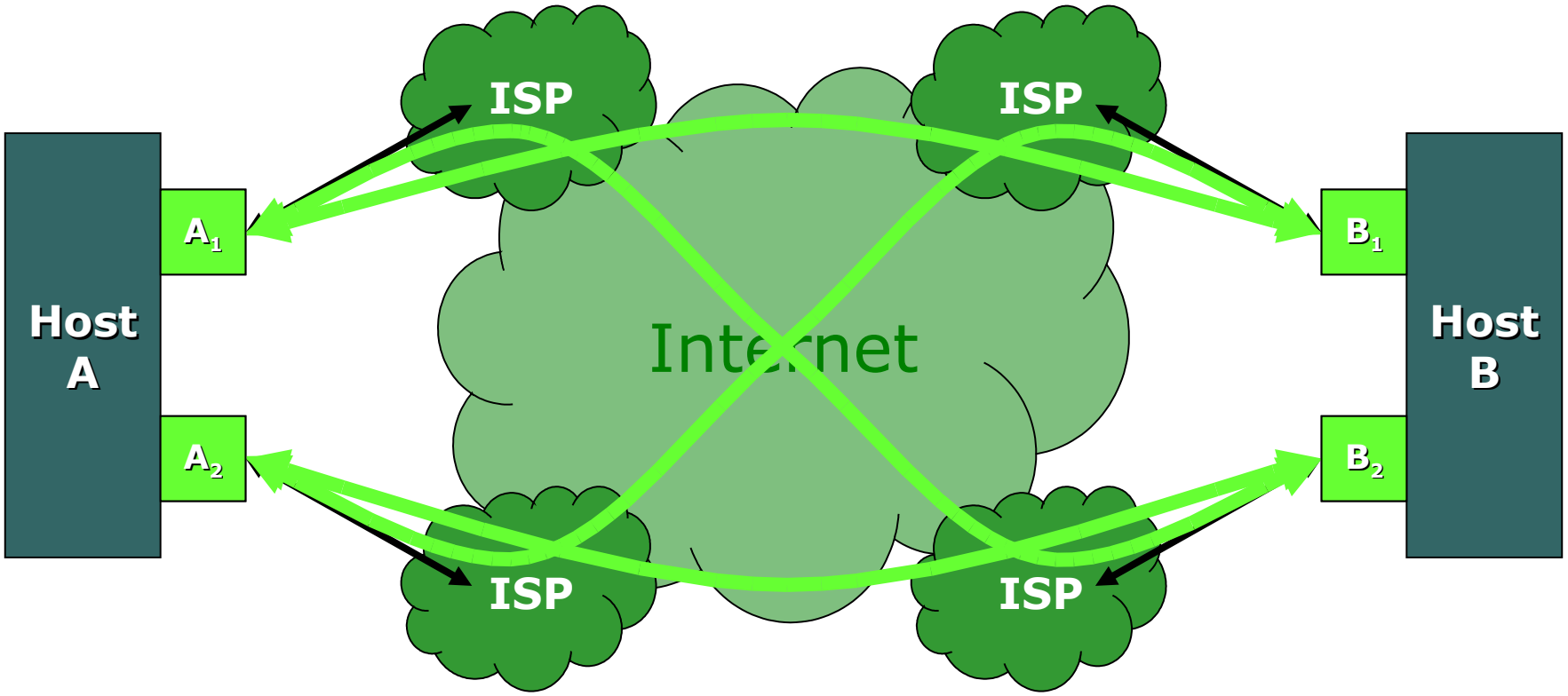


- TCP: Hosts choose 1 of 4 possible connections

(A_{11}, B_{11}) or (A_{11}, B_{22}) or (A_{22}, B_{22}) or (A_{22}, B_{11})

P.E.L.

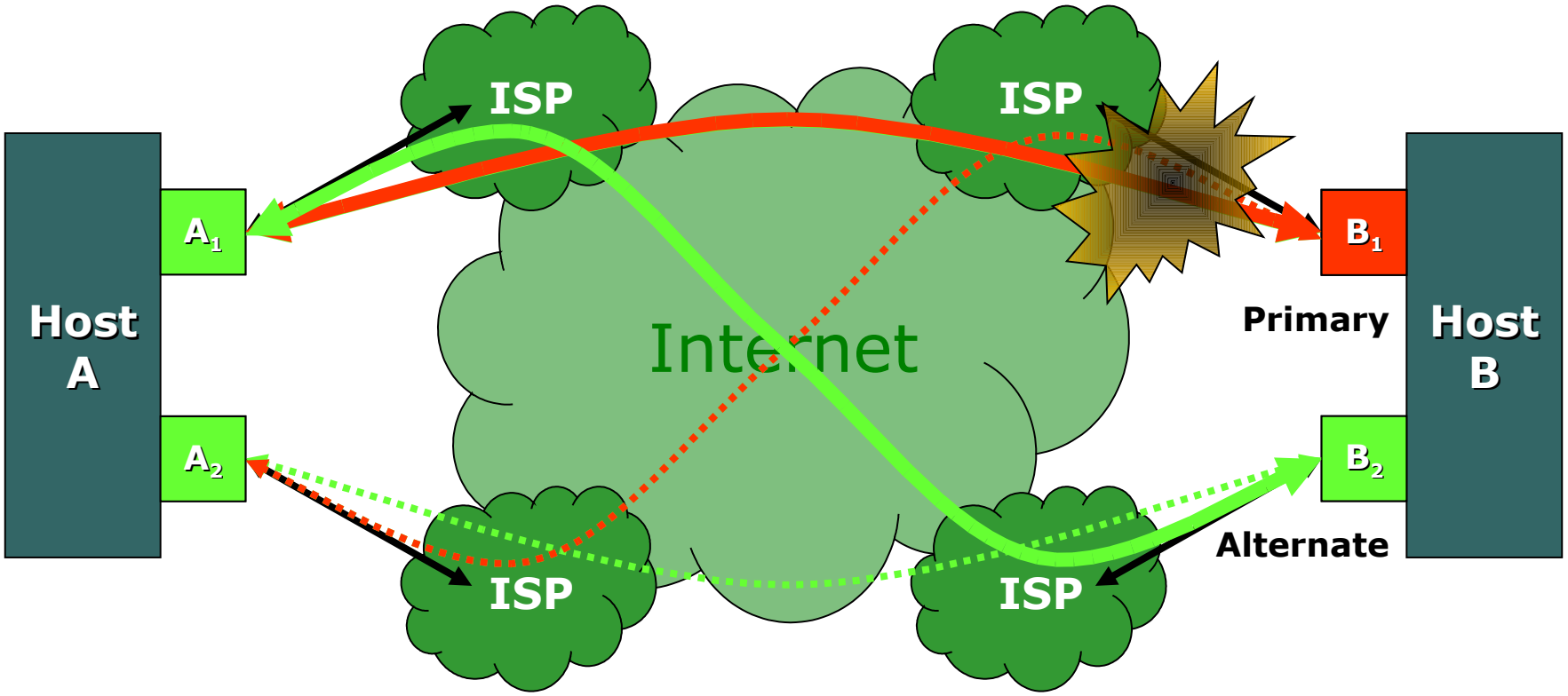
SCTP Multihoming



- SCTP: 1 association incorporating all addresses

  ({A_1, A_2}, {B_1, B_2})

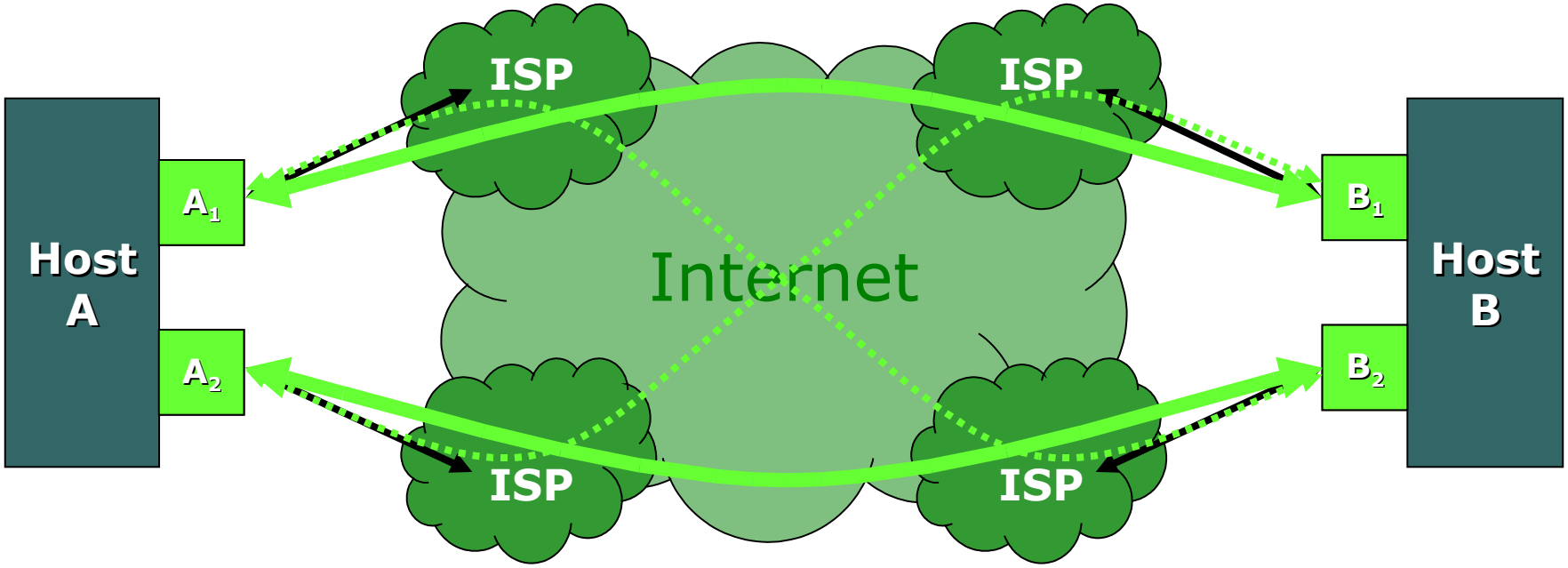
SCTP Multihoming Provides Fault Tolerance



- Primary and alternate destinations
- SCTP fails over to alternate if primary becomes unreachable

SCTP Concurrent Multipath Transfer (CMT)

- Idea: Actively send data to all available destinations to increase throughput



- Solves problem of usable destinations sitting idle
- Current research by Janardhan Iyengar (see links at end)

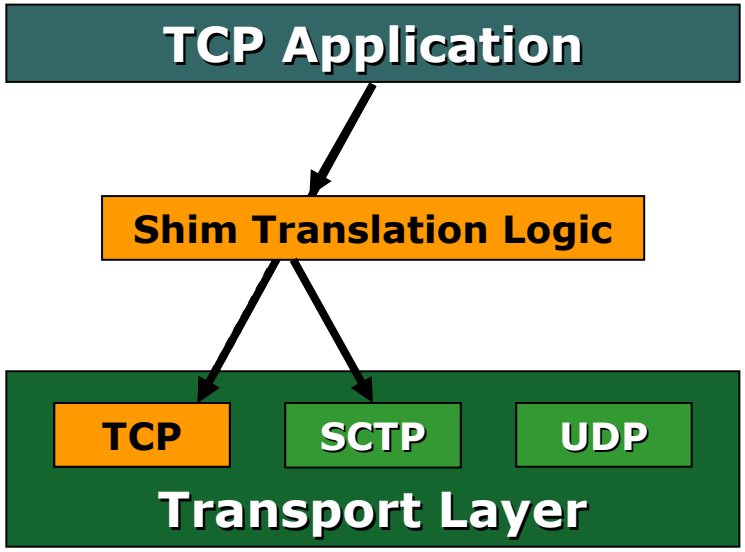
Motivations to Migrate from TCP to SCTP

- Increase application fault tolerance and reliability:
  - SCTP Multihoming
- Increase application throughput:
  - SCTP Concurrent Multipath Transfer
- How to take advantage of SCTP benefits?
  - Rewrite all existing TCP applications – lots of work
  - Incremental deployment ("chicken and egg") problem
- Idea: translate system calls to TCP into equivalent calls to SCTP, using SCTP for end-to-end transport

Outline

- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

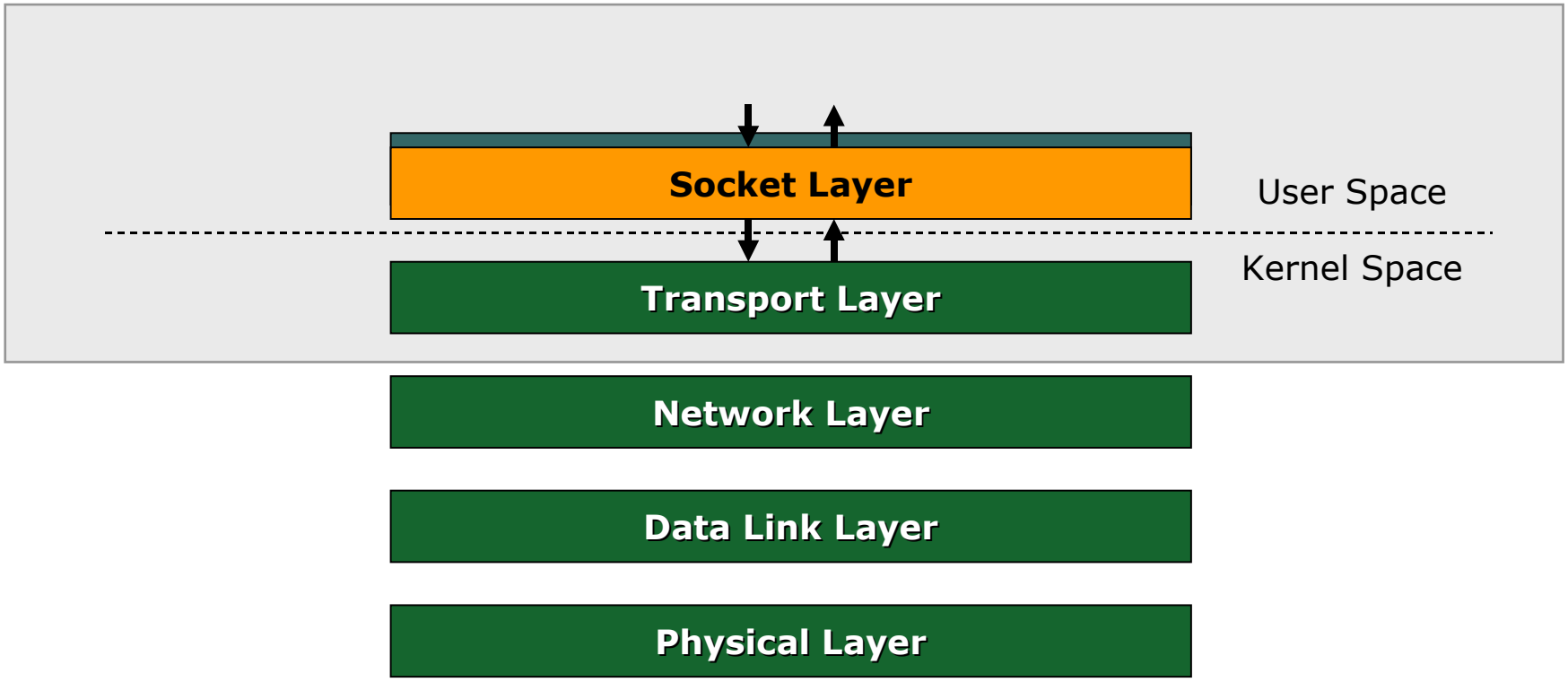
Transparent TCP-to-SCTP Translation



- Translation from TCP to SCTP by shim layer is transparent to application – no modifications to applications whatsoever
- Initial implementation is in FreeBSD 4.10 kernel; currently porting to FreeBSD 7.0
- Why kernel versus user library – pros and cons?

Socket Layer / API

- Maps protocol-independent requests from application to protocol-specific implementation in kernel

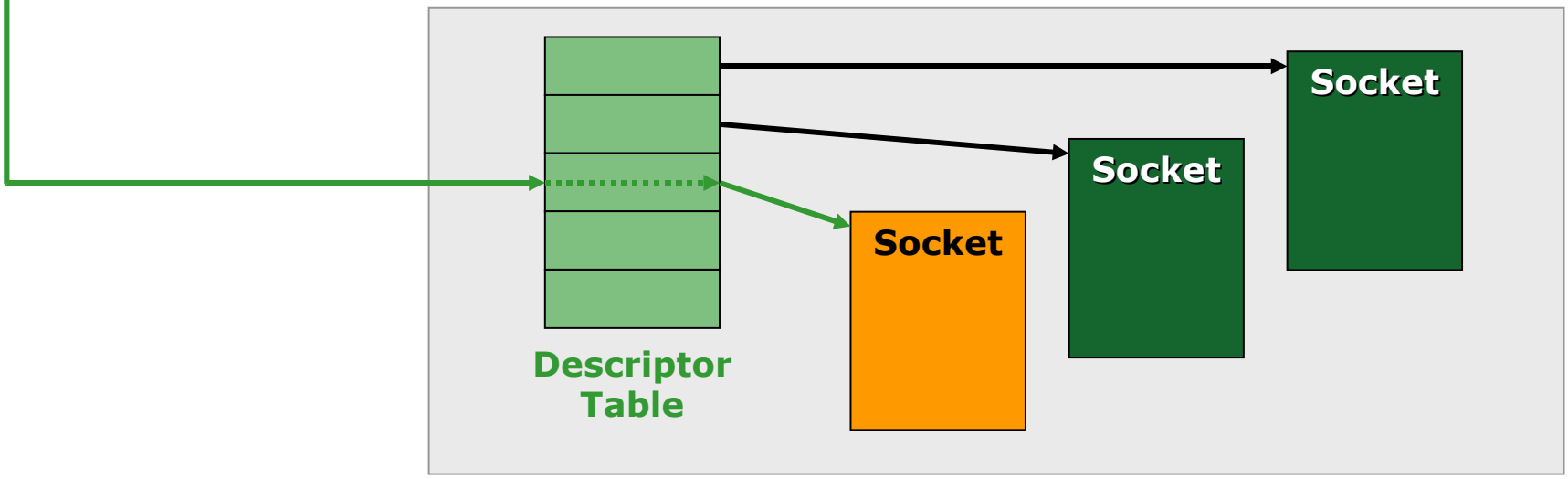


Sockets

- Represent endpoint of network communication

```
socket_desc = socket(domain, type, protocol);
…
connect(socket_desc, …);
…
send(socket_desc, …);
receive(socket_desc, …);
…
close(socket_desc);
```

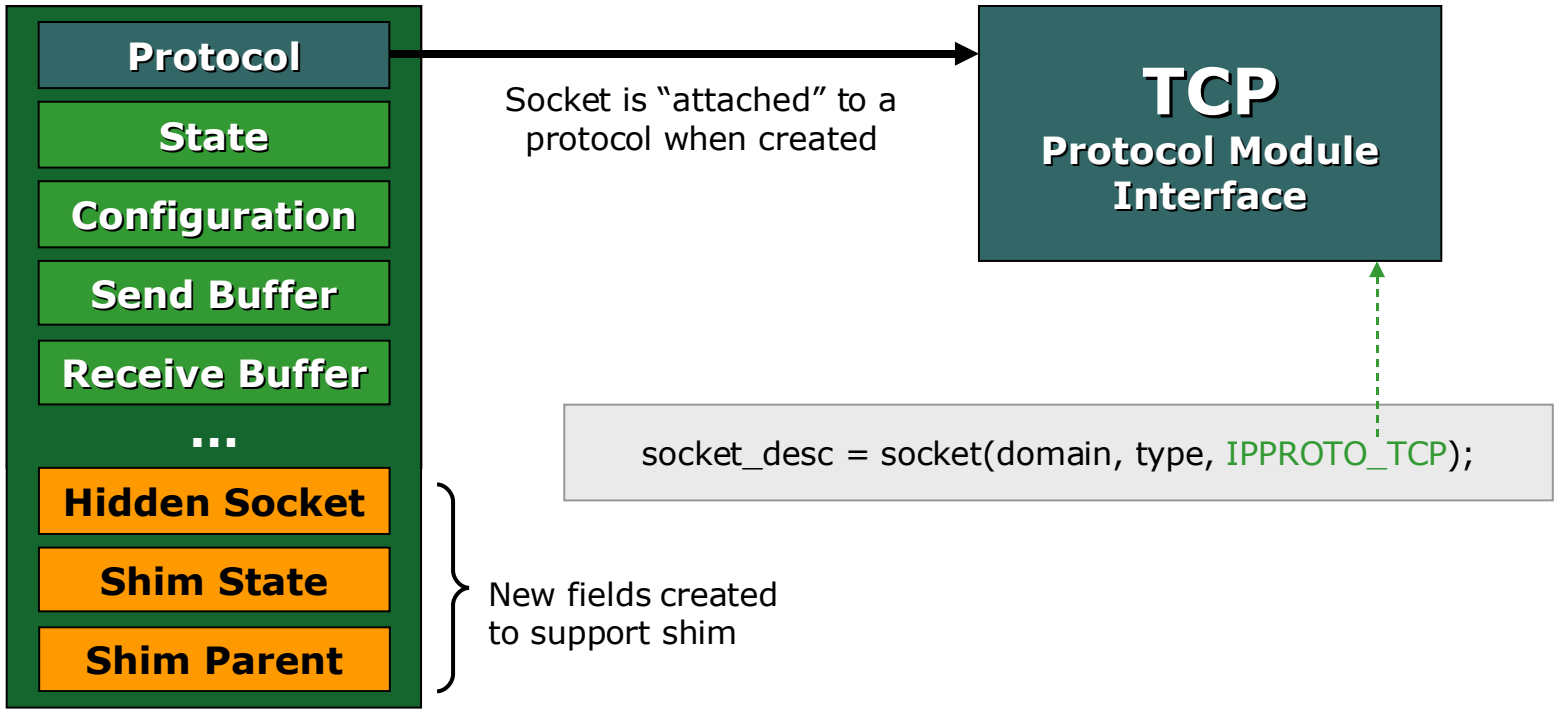
Application



P.E.L.

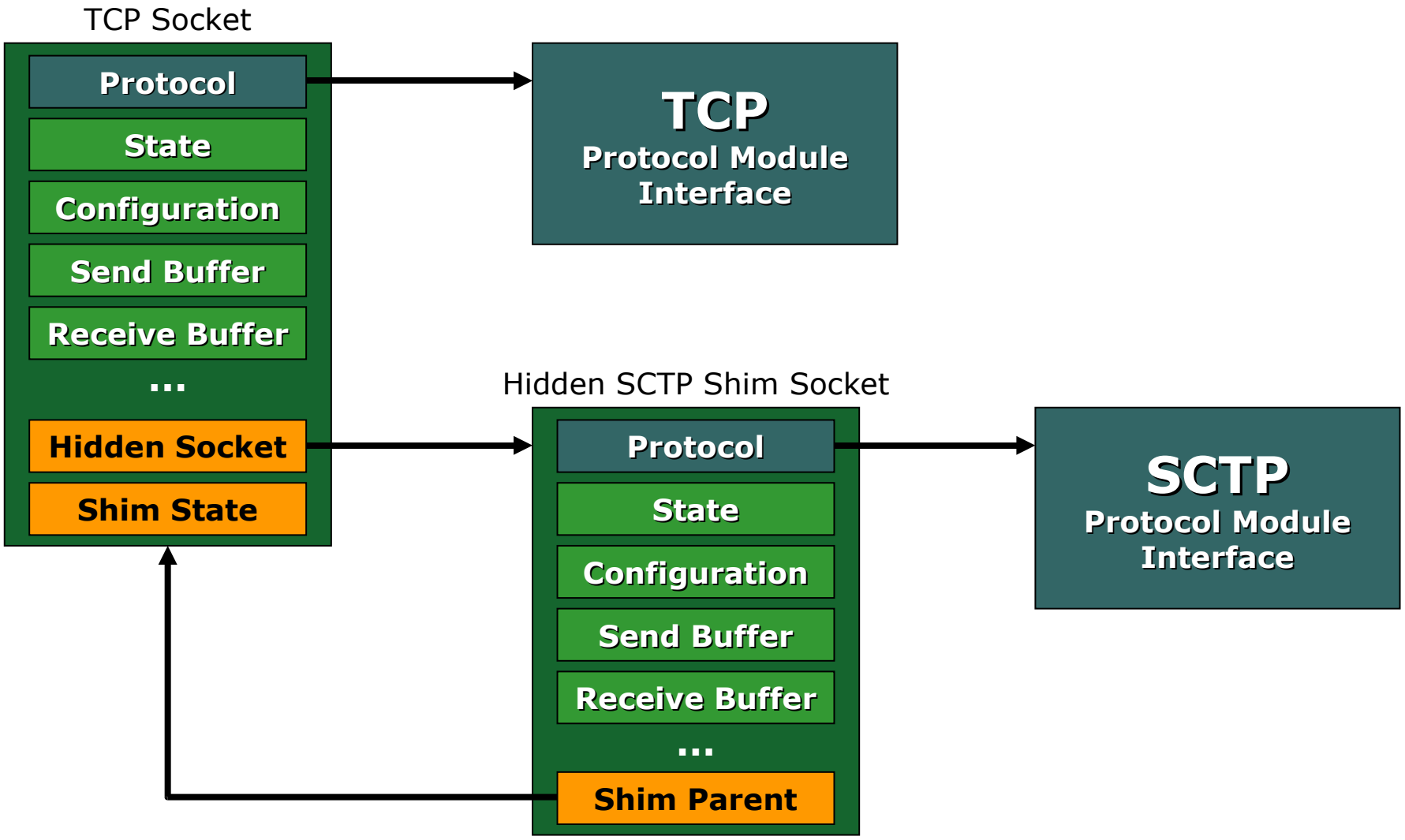
Socket-Protocol Binding

- Socket fields:
  - Protocol (TCP, UDP, SCTP, etc)
  - Configuration information (socket options, etc)
  - State (connected, disconnecting, etc)
  - I/O buffers (sending, receiving)



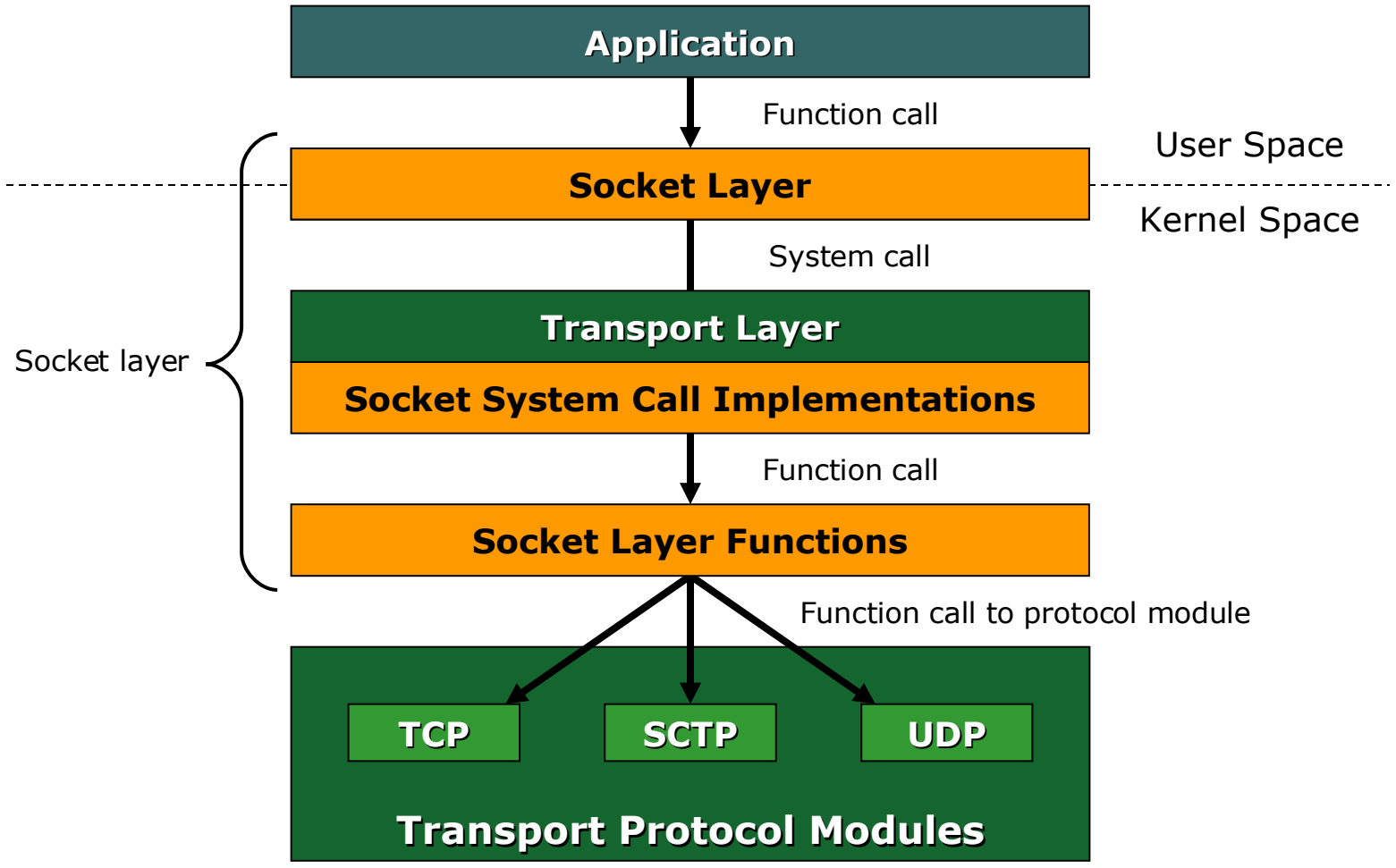
P.E.L.

Hidden SCTP Socket

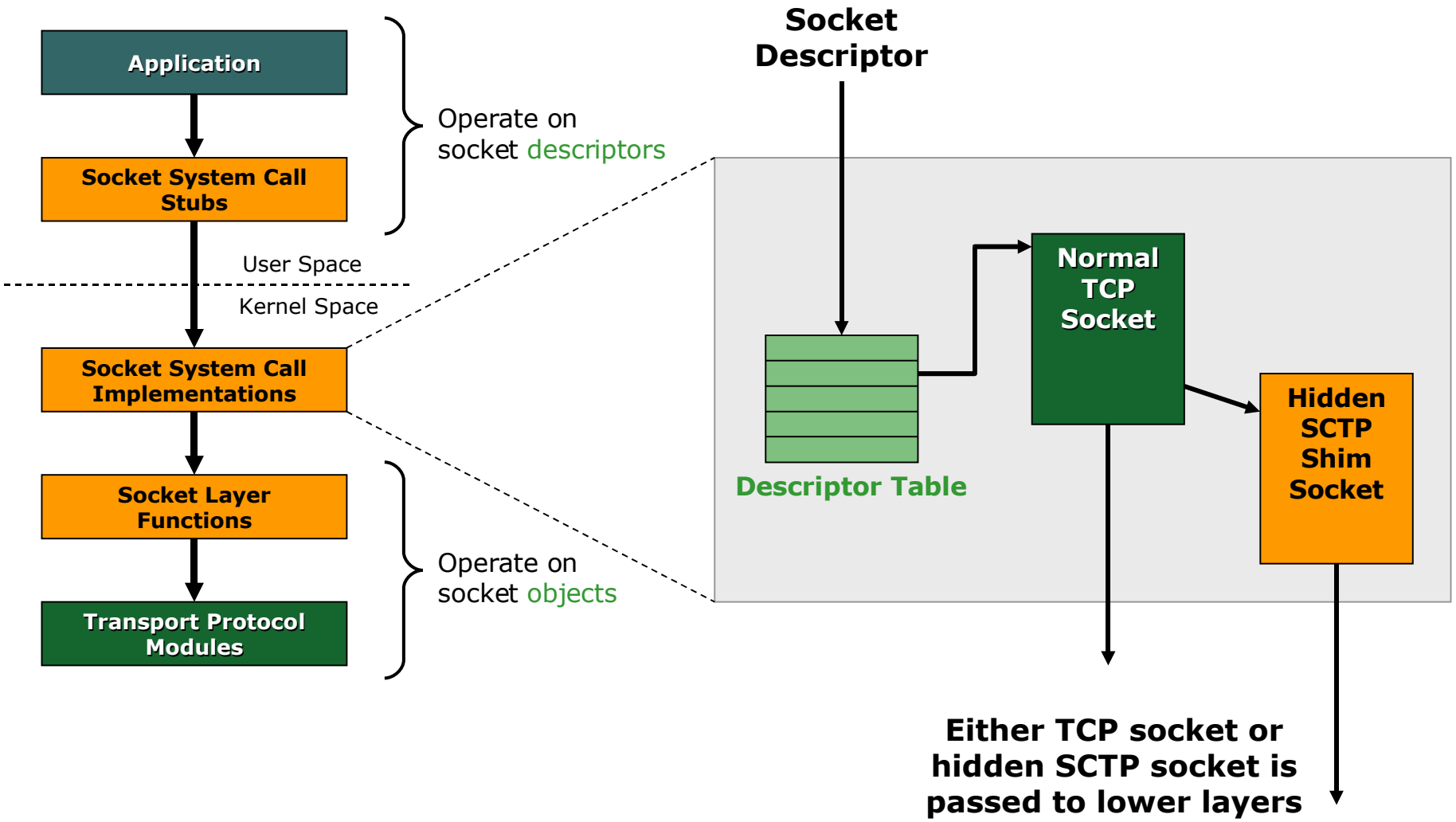


P.E.L.

Socket Layer in Detail

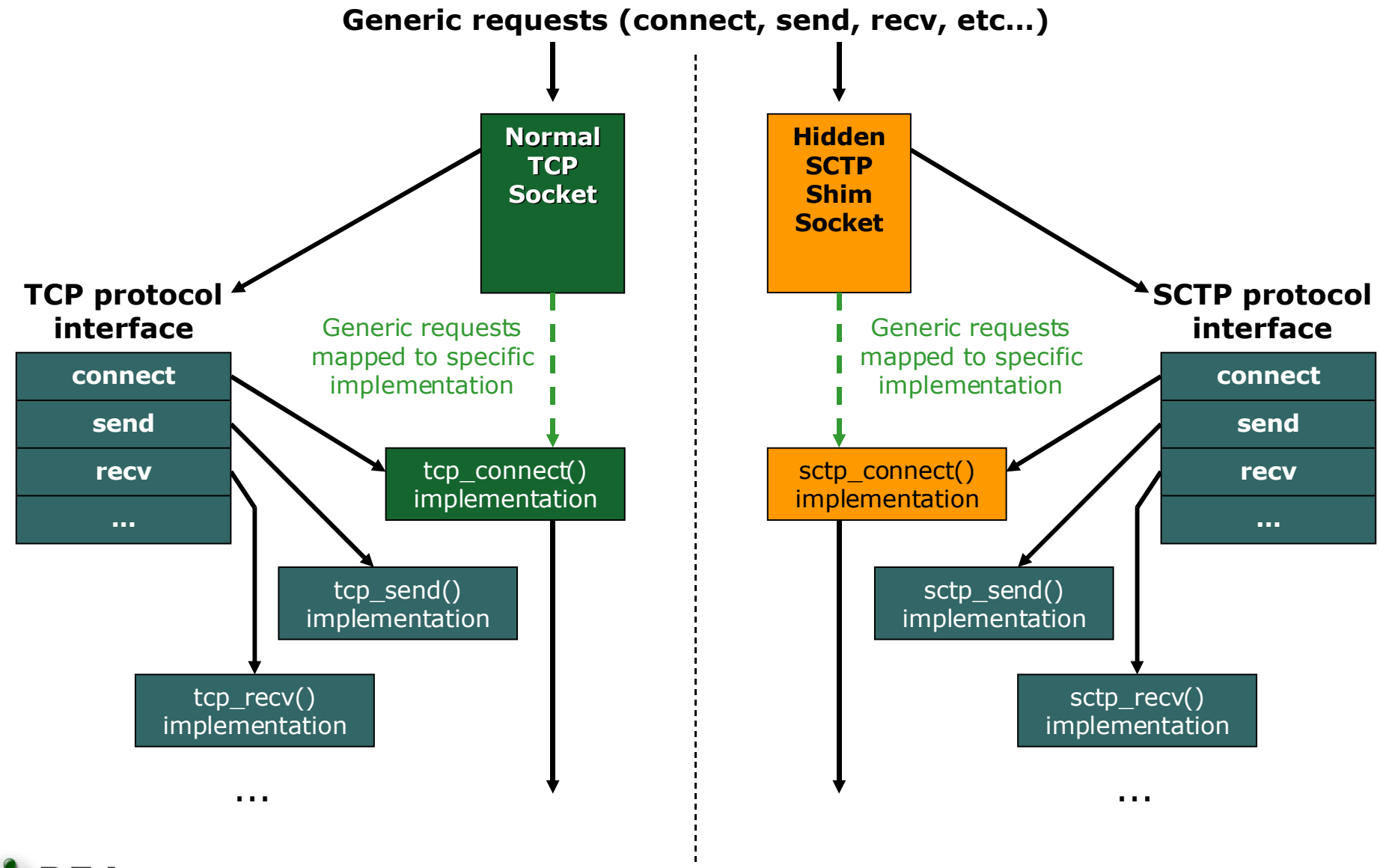


Hidden Socket Substitution



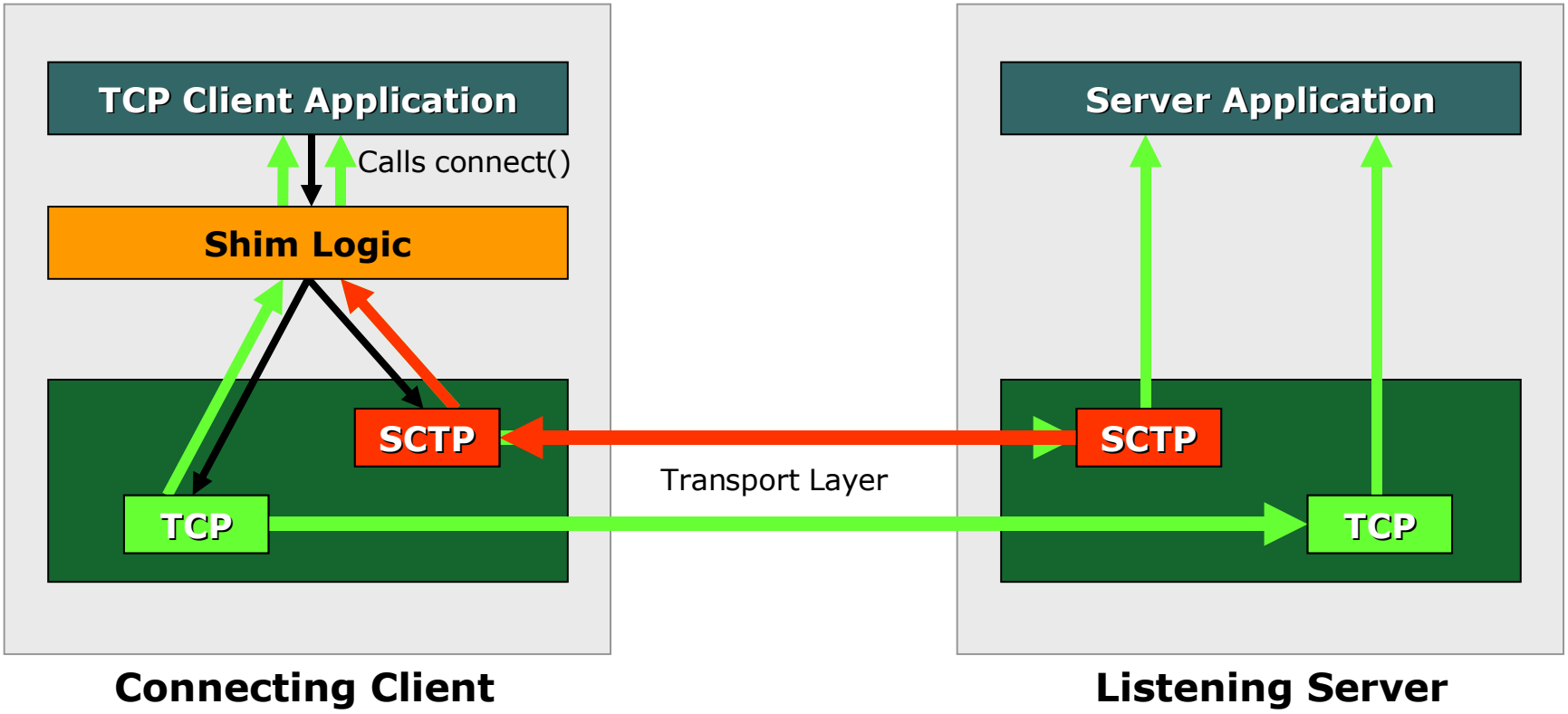
P.E.L.

Normal TCP / Hidden SCTP Socket Use



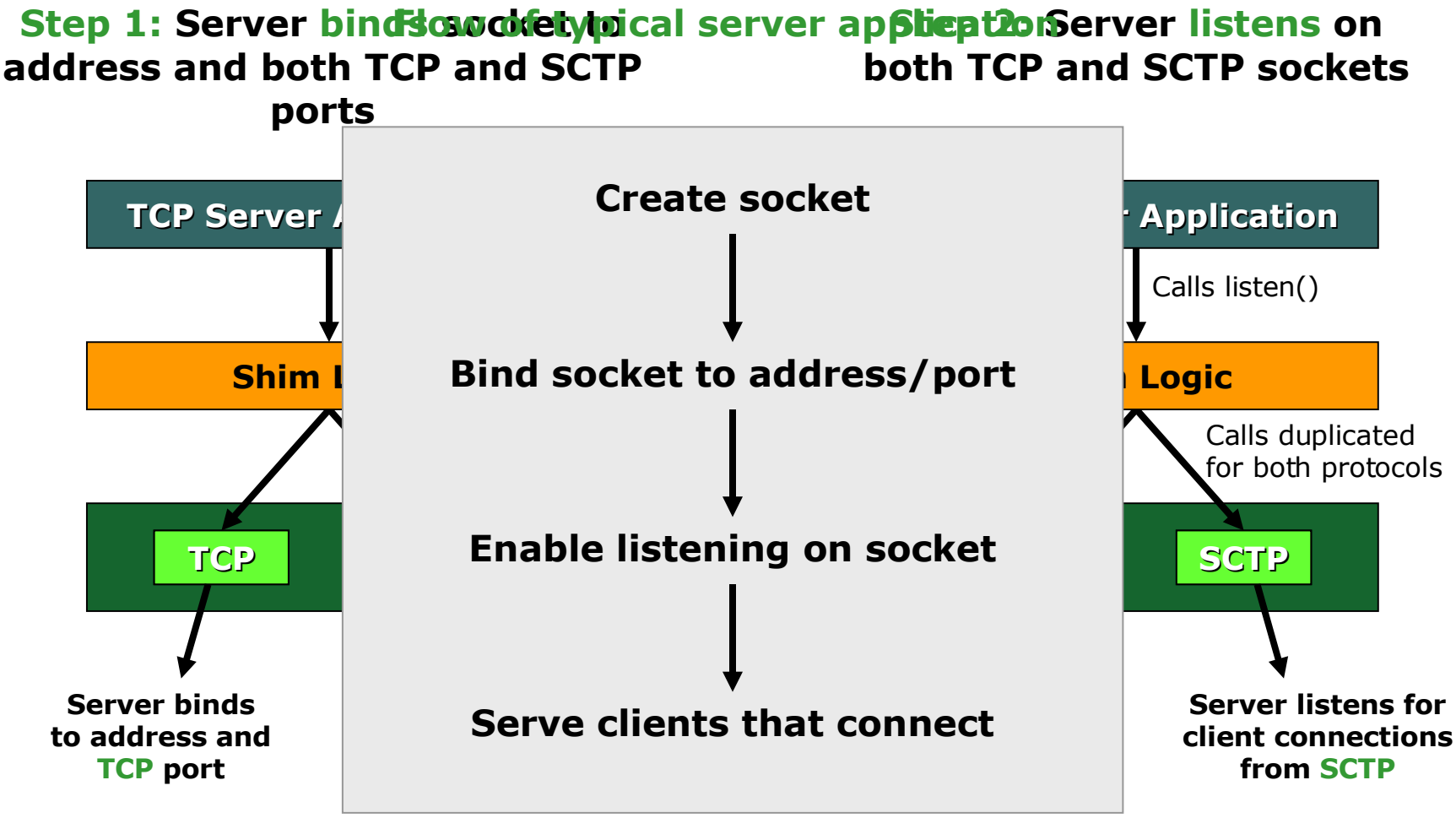
P.E.L.

Client Connecting with Shim Enabled

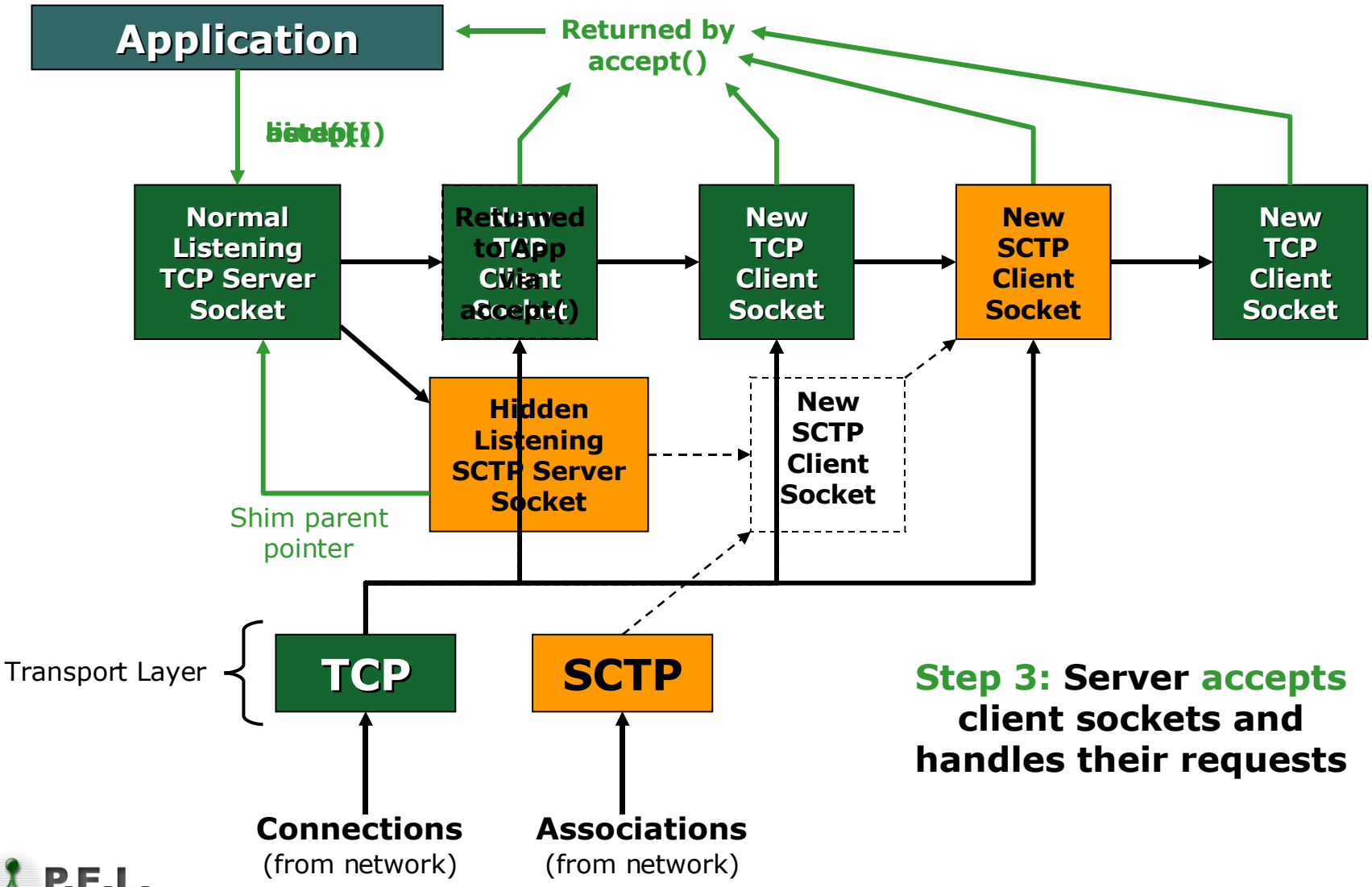


- Client tries connecting with SCTP first and falls back to TCP if SCTP does not work

Server Bind/Listen with Shim Enabled



TCP and SCTP Listening Sockets



Step 3: Server accepts client sockets and handles their requests

P.E.L.

Outline

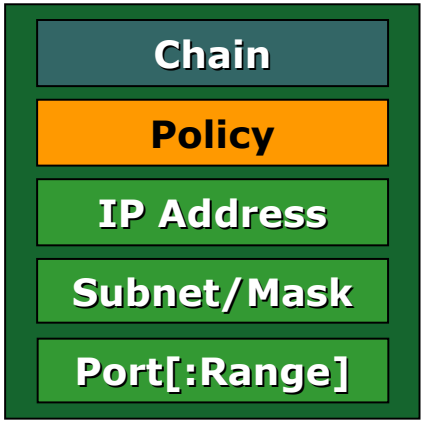
- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

Controlling Shim Operation

- Global on/off switch for shim lacks precision – every application has same setting
- Administrators need finer control
- Solution: selectively enable/disable shim on per-application basis using rules
- Rules match application network use based on:
  - Addresses
  - Subnets
  - Port numbers or ranges

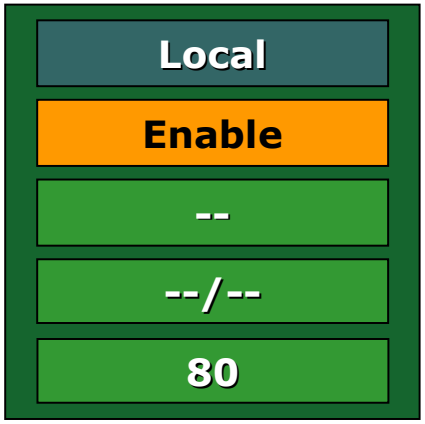
Rule Format

- Chain
  - Local: Rule for local listening (server) sockets
  - Remote: Rule for connecting (client) sockets to remote endpoints
- Policy
  - Enable: Shim enabled if rules match
  - Disable: Shim disabled if rules match
- If address, subnet, or port matches rule, use rule's policy, else use global default policy

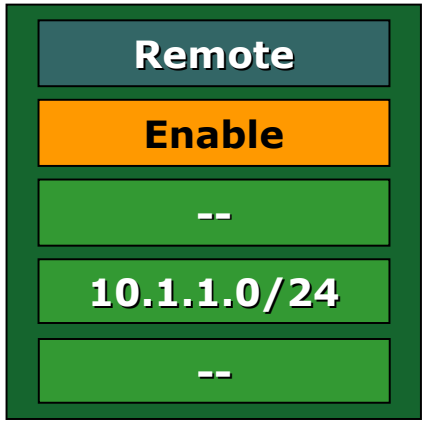


Shim Rule

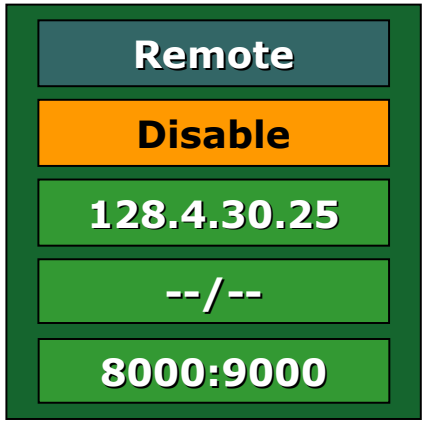
Rules Semantics



Matches applications using listening (server) socket that is bound to any address and port 80

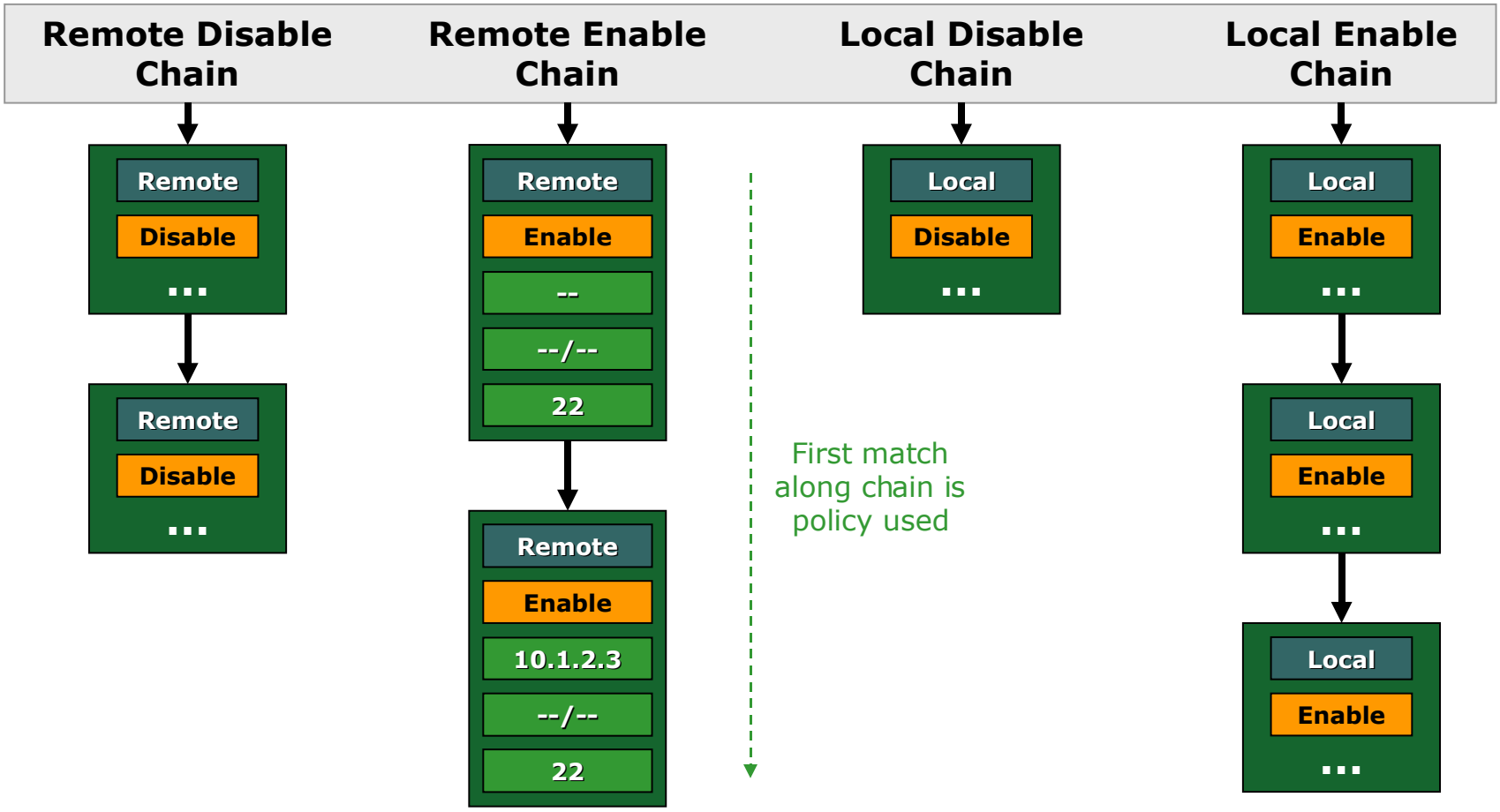


Matches applications connecting to remote host on 10.1.1.0/24 subnet on any port



Matches applications connecting to 128.4.30.25 on any port in the range 8000 to 9000

Shim Rules Table Design



Suppose application calls:

connect(10.1.2.3 port 22);

Global Remote Policy → Disable

Global Local Policy → Disable

P.E.L.

Shim Administrative Practices

- Shim rules system allows policies to fit needs of individual sites
- Default local/remote policies regulate how aggressively the system attempts to use shim
- Fine tune default settings with rules:
  - Enable or disable shim for specific applications
  - Ensure that most restrictive rules appear earlier in chains than less restrictive rules

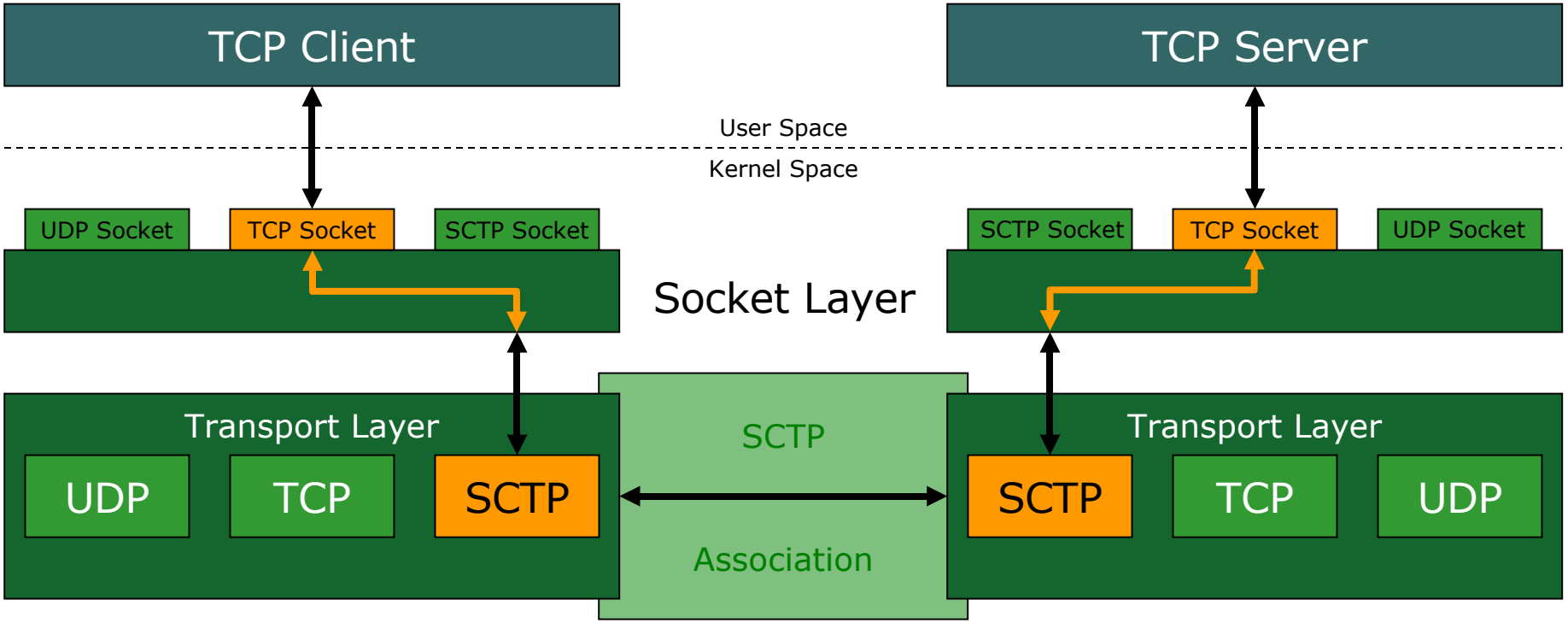
Outline

- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

Experimental Results

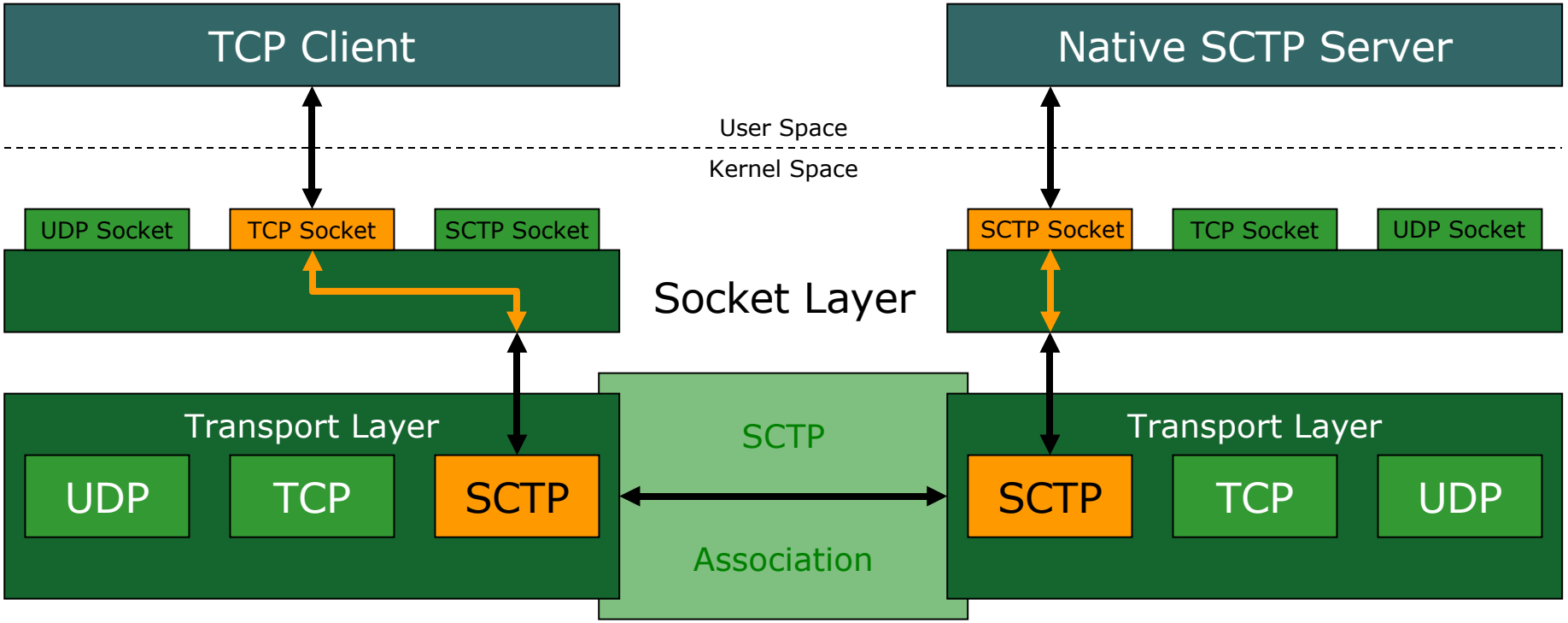
- So far, several applications verified to work as expected without modification running over shim:
  - Telnet
  - SSH
  - HTTP using Apache server and Firefox browser
  - Streaming audio using Icecast server and XMMS player
- End user cannot distinguish between normal TCP and shim using SCTP (except by wiresharking!)
- Two experiment configurations

TCP-SCTP-TCP Translation



- Telnet, SSH, HTTP (Apache + Firefox), streaming audio (Icecast + XMMS) work in this configuration!

TCP-SCTP Translation

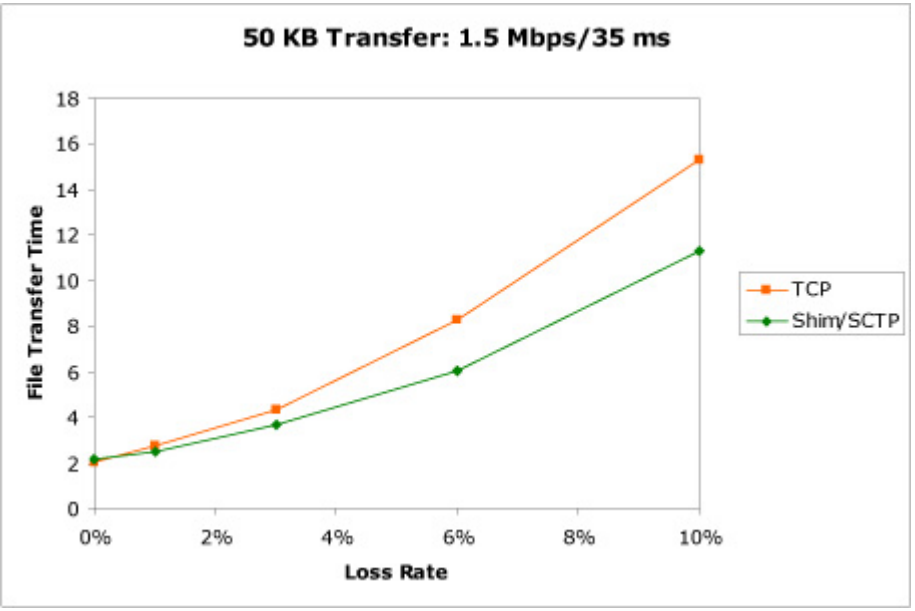


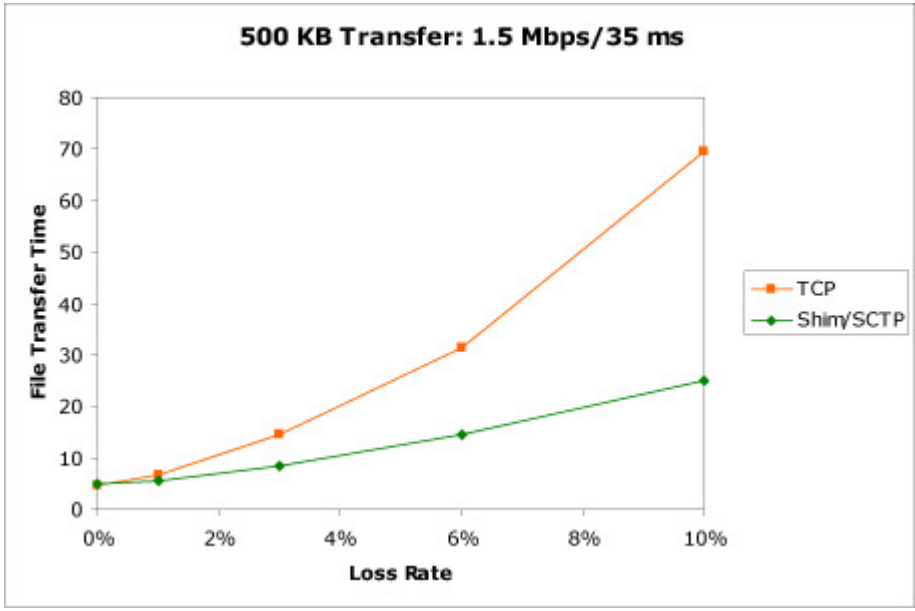
- HTTP (native SCTP-enabled Apache + regular Firefox) works in this configuration!

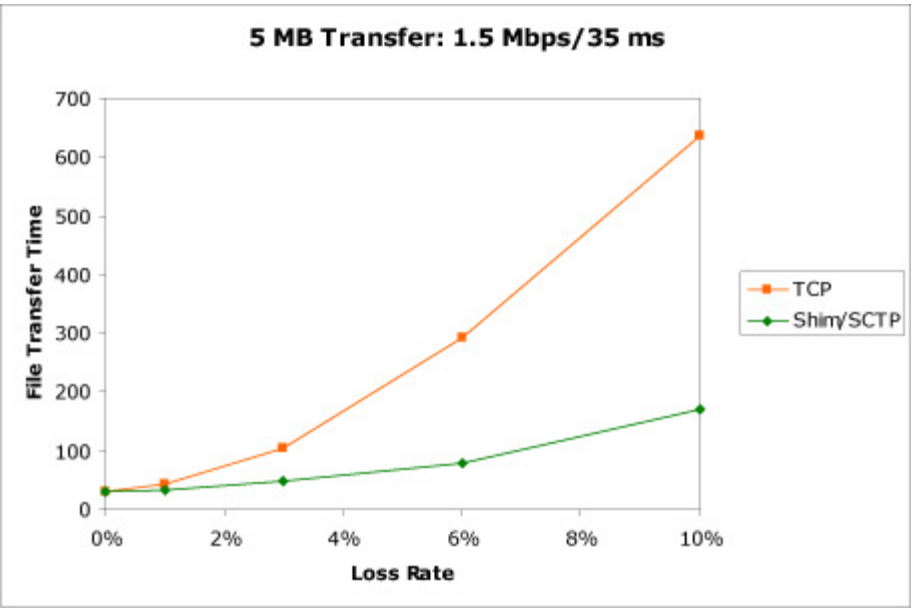
Performance Measurements

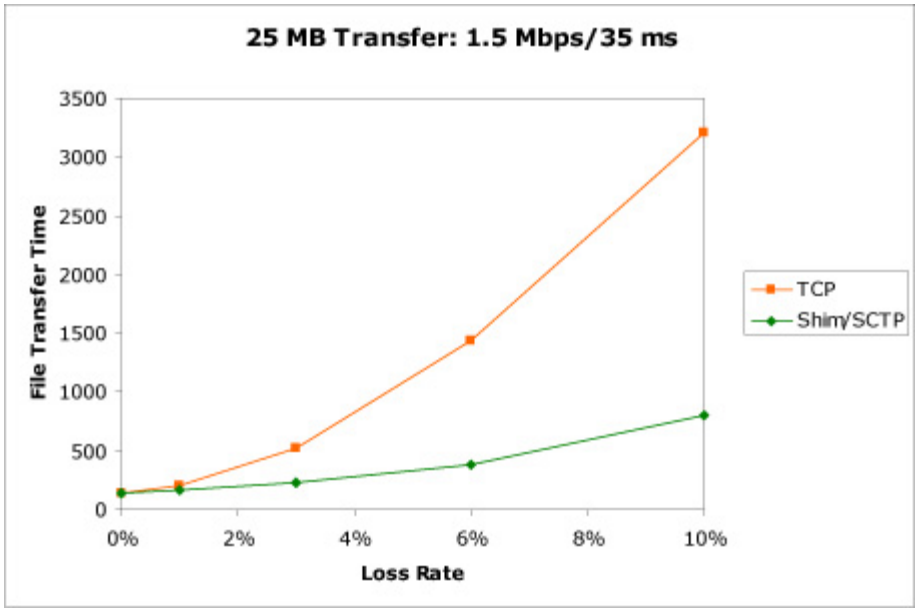
- Measuring file transfer time with `scp` over shim
- 1.5 Mbps / 35 ms latency path created using Dummynet running on FreeBSD 4.10; 50-packet tail drop queue
- Uniform random loss rates of {1%, 3%, 6%, 10%}
- Files sizes of {50 KB, 500 KB, 5 MB, 25 MB}
- Average transfer times of 30 runs for all but 50 KB; 90 runs for 50 KB transfers due to higher variance

Performance Results









Interpretation of Results

- For low loss rates (less than 3%) and short transfers (50 KB), TCP and SCTP perform similarly
- At high loss rates for longer file transfers, SCTP clearly outperforms TCP
  - Both protocols have AIMD congestion control
  - SCTP uses SACK by default
  - SCTP has Appropriate Byte Counting
- Using the shim and SCTP provides performance no worse than TCP, and significantly better in some situations

Outline

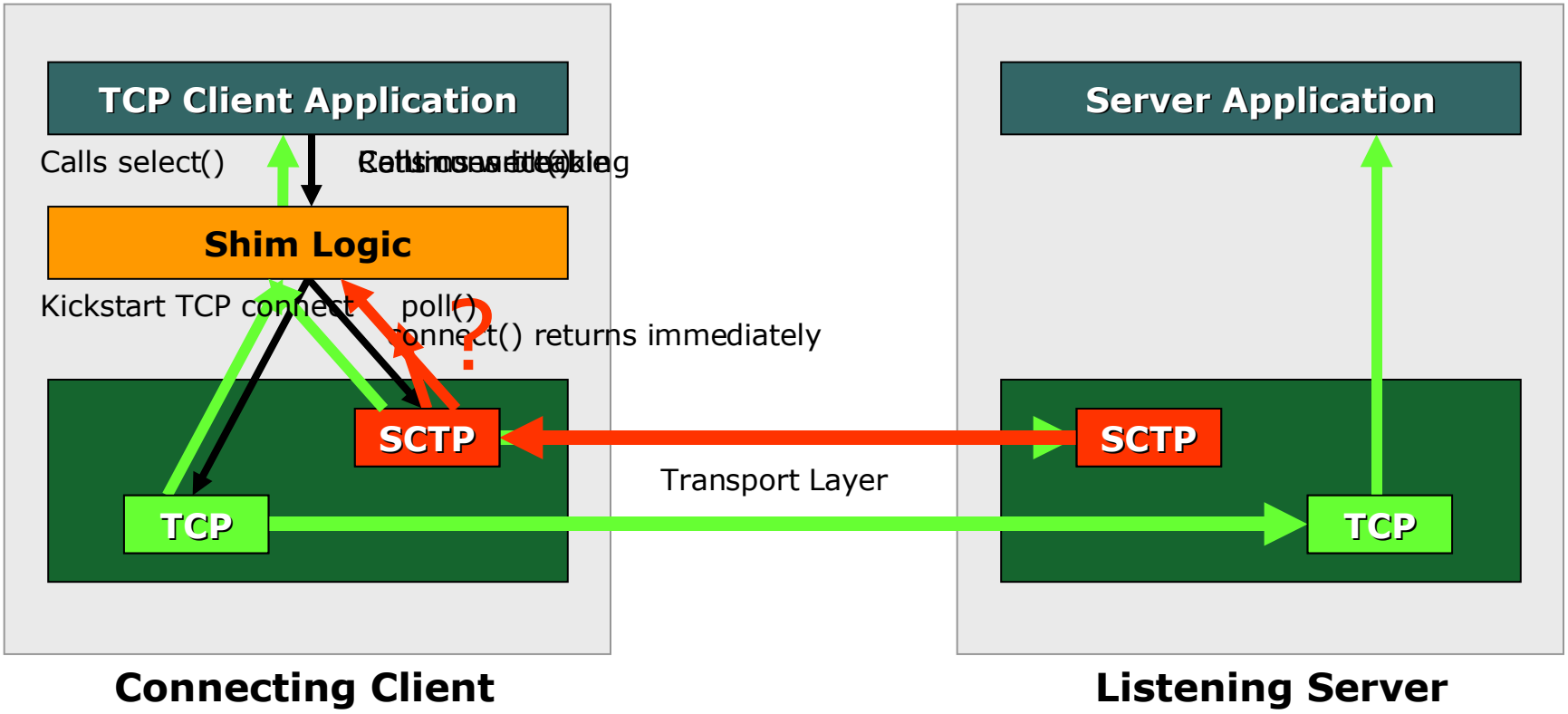
- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

Handling Nonblocking Connects

- Shim is application-driven: when application requests action on a socket, hidden SCTP socket is used instead

- What happens when action or response is asynchronous, like nonblocking connect?

Nonblocking Connect Events

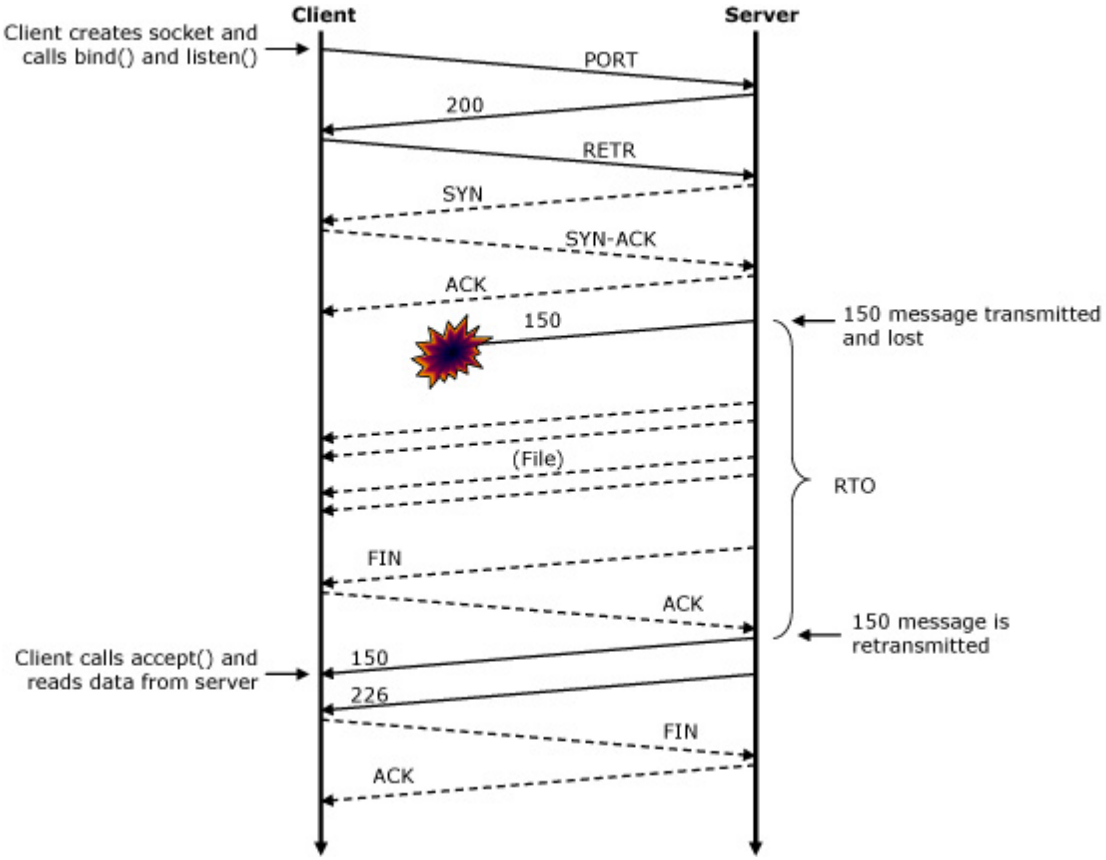


TCP's Half-Closed State

- TCP uses a 4-way handshake for closing the connection, which allows the connection to be in a half-closed state
- SCTP uses a 3-way handshake for closing the association
- Both TCP applications must call close before connection is torn down; only 1 SCTP application calling close will tear down the association
- When is this a problem?

FTP Transfer over TCP

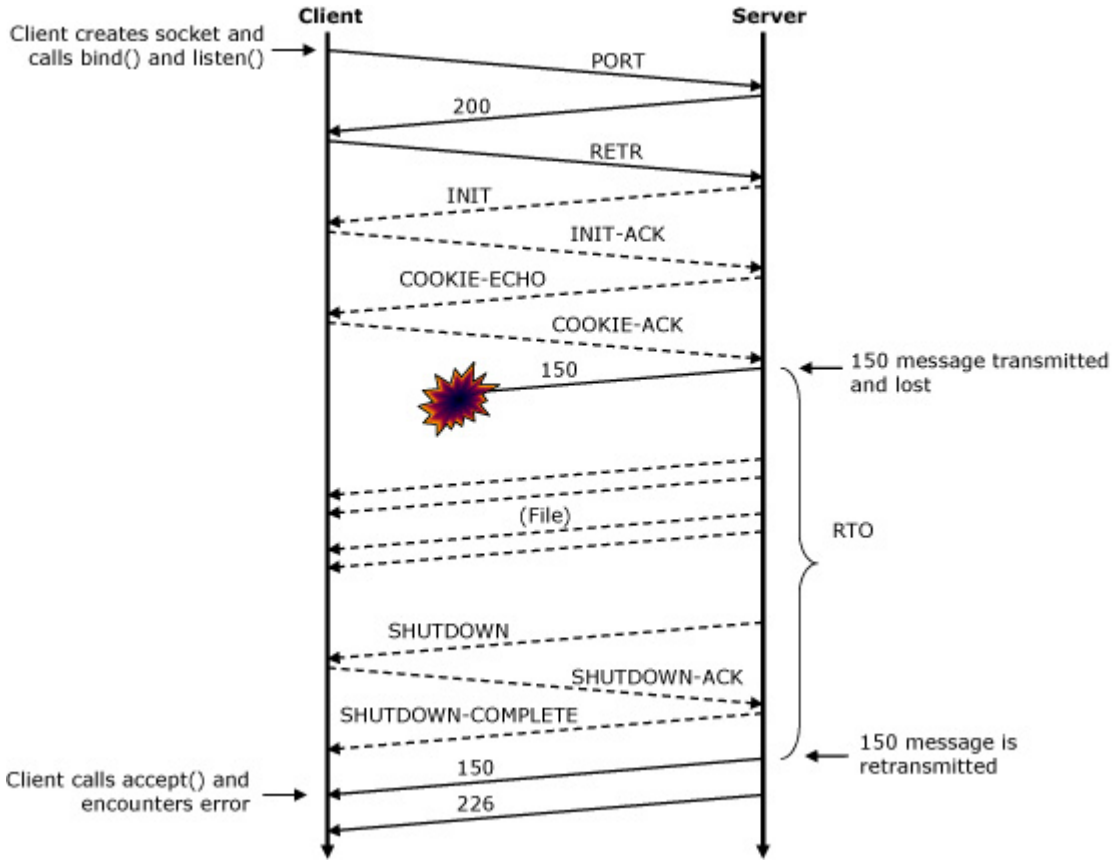
Solid lines: Control connection
Dashed lines: Data connection



P.E.L.

FTP Transfer over SCTP

Solid lines: Control connection
Dashed lines: Data connection



P.E.L.

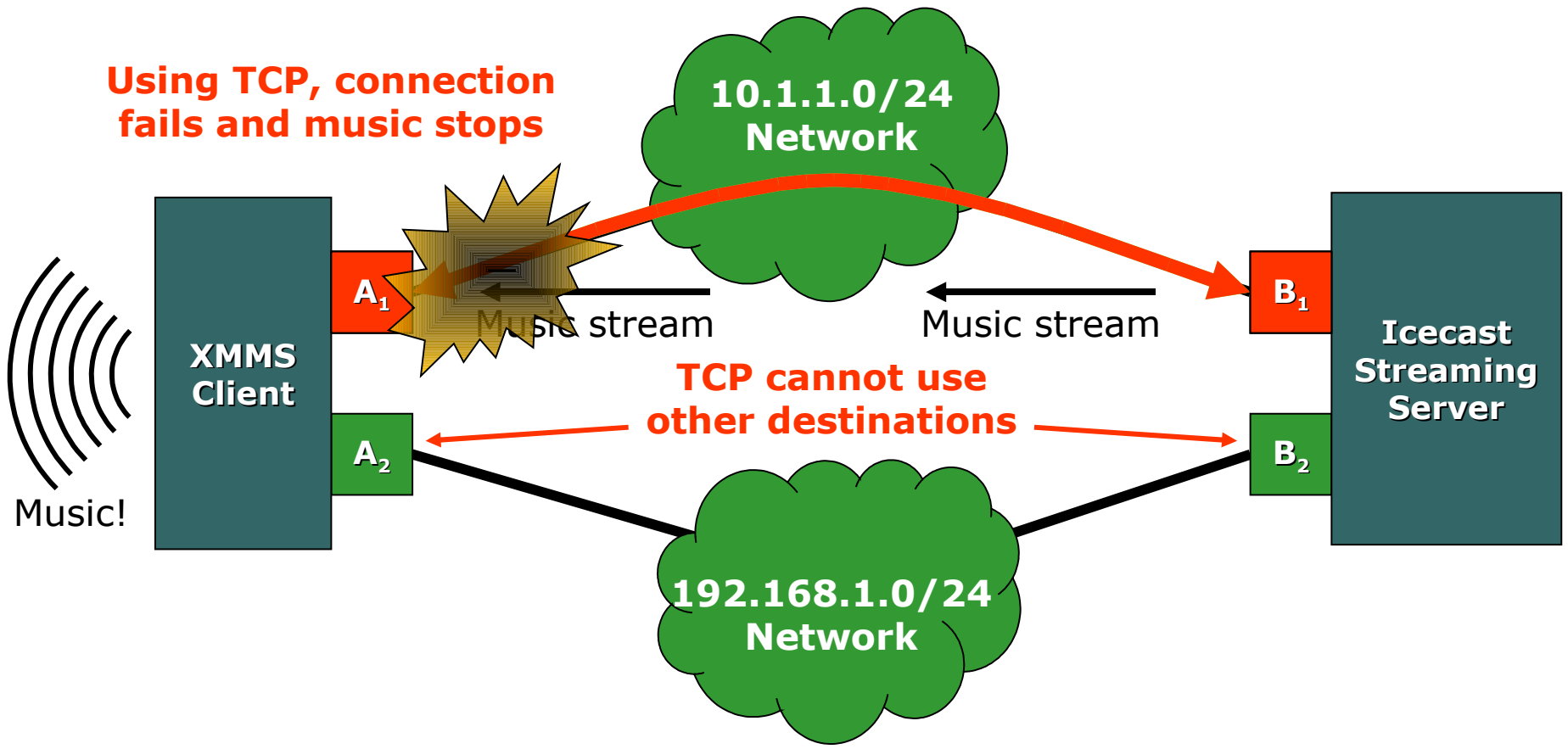
Handling Half-Close with SCTP Shim

- Some applications depend on specific knowledge of how TCP handles half-close to function correctly
- Goal of transparent translation requires application behavior not be changed
- Possible solution: emulate the TCP half-close semantics by passing state between the two endpoints using an unused SCTP data stream

Outline

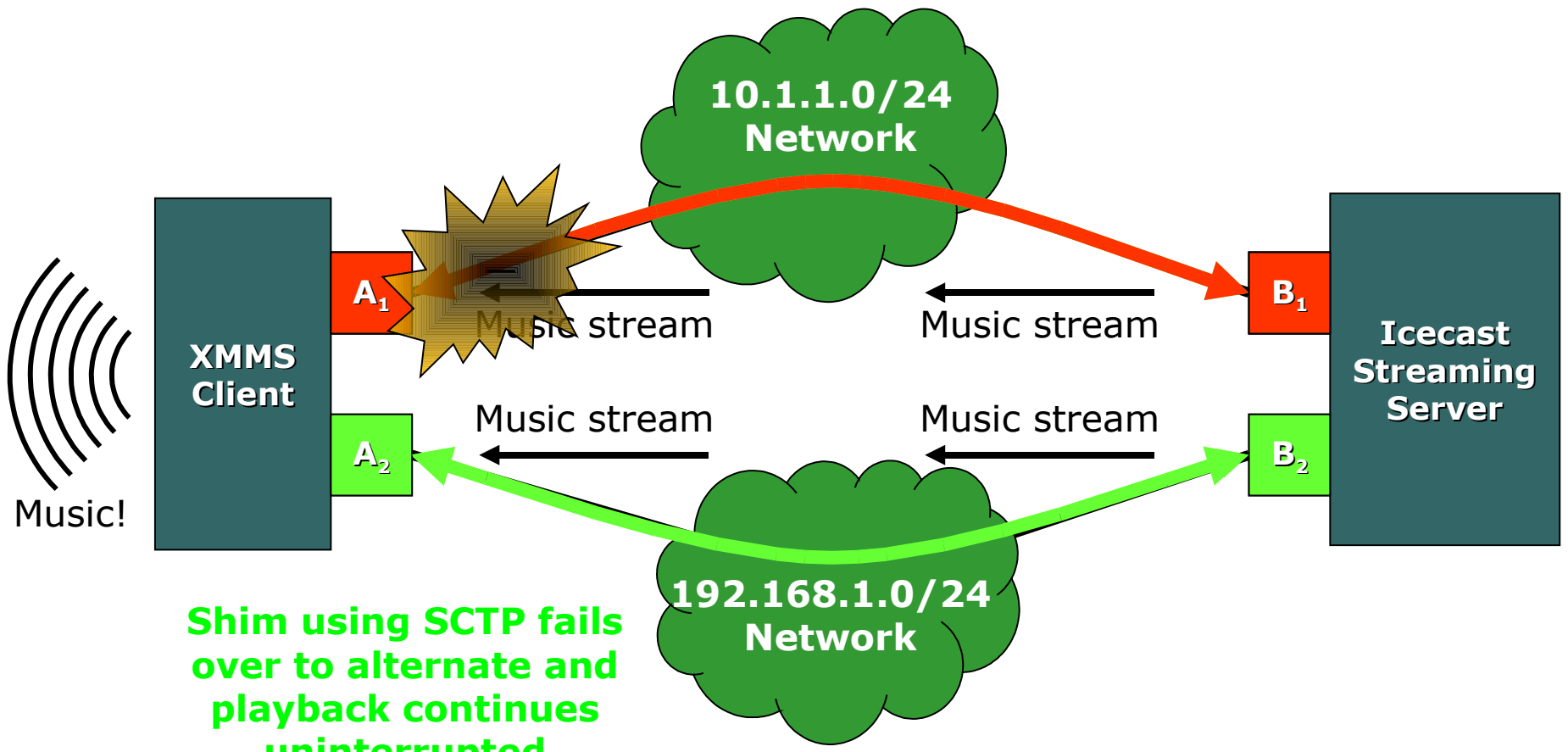
- Motivations
- Implementation overview
- Controlling the shim
- Experimental results
- Challenges and future work
- Demo

Demo: Path Failure with TCP



P.E.L.

Demo: Path Failure with Shim using SCTP



P.E.L.

Further Reading

- General SCTP information:
  - RFC 2960 (Stream Control Transmission Protocol)
  - RFC 3758 (Partial Reliability)

- HTTP and FTP over SCTP:
  - SCTP: An innovative transport layer protocol for the web
    - http://www.cis.udel.edu/~amer/PEL/poc/pdf/WWW2006-SCTPf
  - Improving multiple file transfers using SCTP multistreaming
    - http://www.cis.udel.edu/~amer/PEL/poc/pdf/IPCCC2004CORRE

Questions or Comments?

- For more information about SCTP research:
  - http://pel.cis.udel.edu
- Concurrent Multipath Transfer (CMT)
  - http://www.cis.udel.edu/~iyengar/publications/
- Questions about transparent TCP-to-SCTP translation shim layer:
  - ryan.bickhart@gmail.com