An Evaluation of Fault-Tolerant TCP-Splice Based Web Server Architectures

Manish Marwah Jacob Delgado Shivakant Mishra

Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309

Christof Fetzer

Department of Computer Science
Dresden University of Technology
Dresden, Germany D-01062

IEEE International Conference on Dependable Systems and Networks (DSN) 2006, Philadelphia, June 25 – 28, 2006

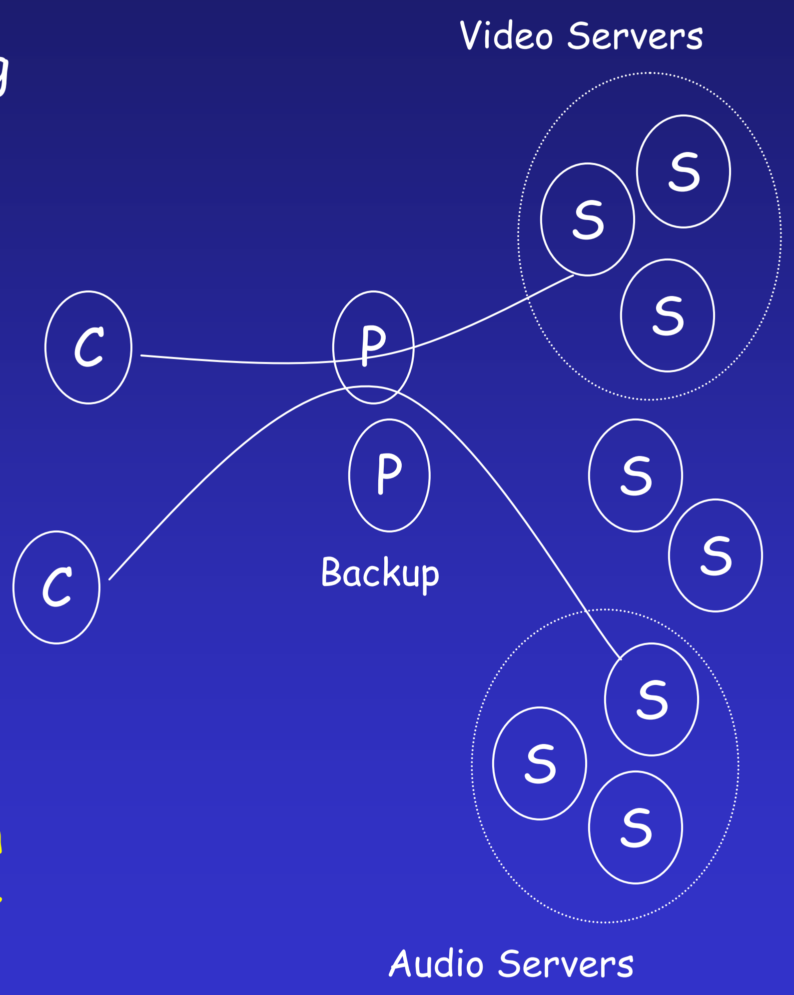
Outline

- Introduction
- Enhancements to TCP Splice
- Our Web Server Architecture
- Simulation Results
- Conclusions

Web Proxies

- Proxies used for
  - Content aware (layer 7) routing
  - Security policies
  - Caching
  - Network management
  - Usage accounting

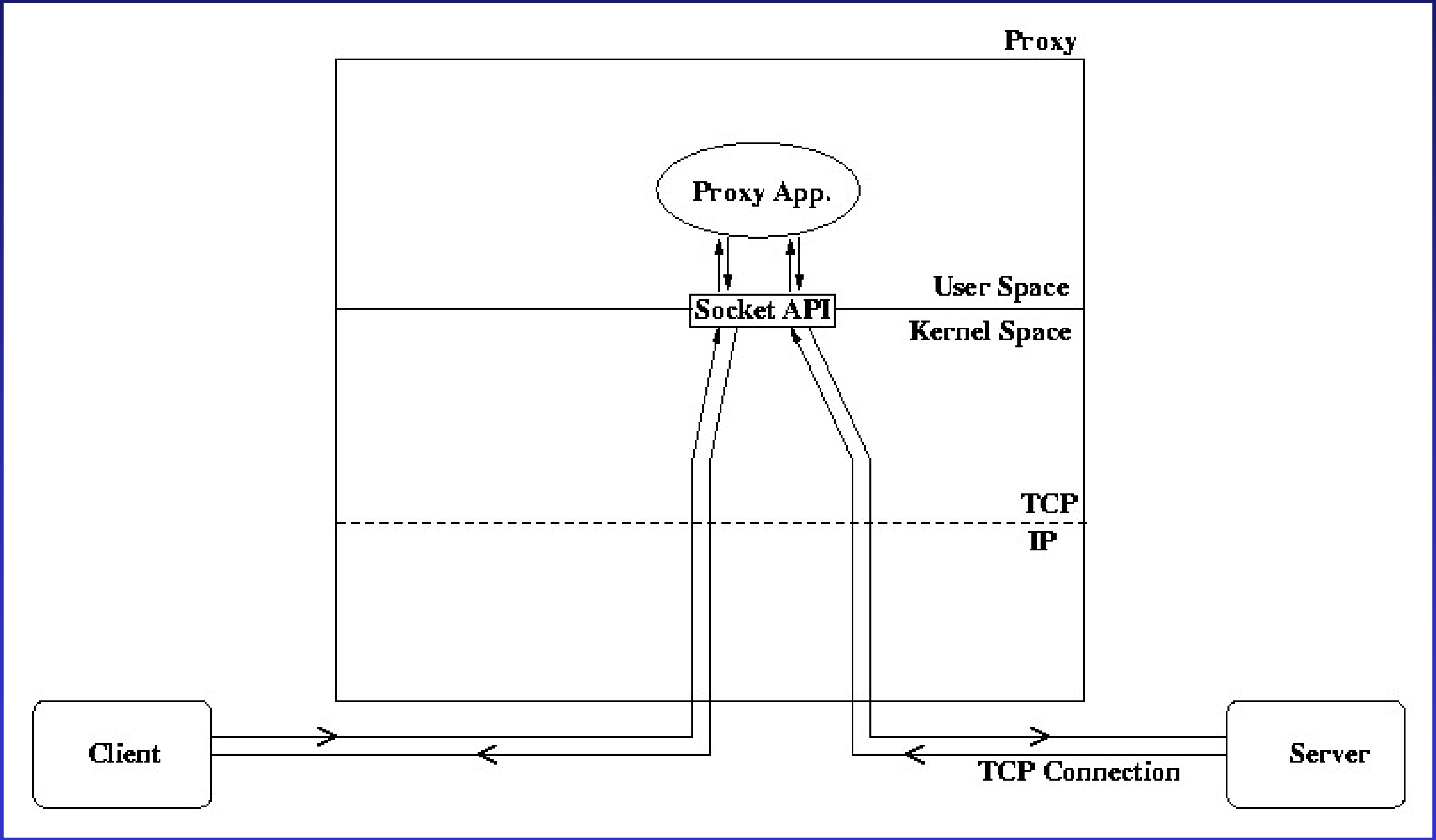
- Drawbacks
  - performance issues
    - kernel <-> user data copying
    - Context switches
  - Single point of failure
    - Even with a backup, connection is broken and would need to be re-established



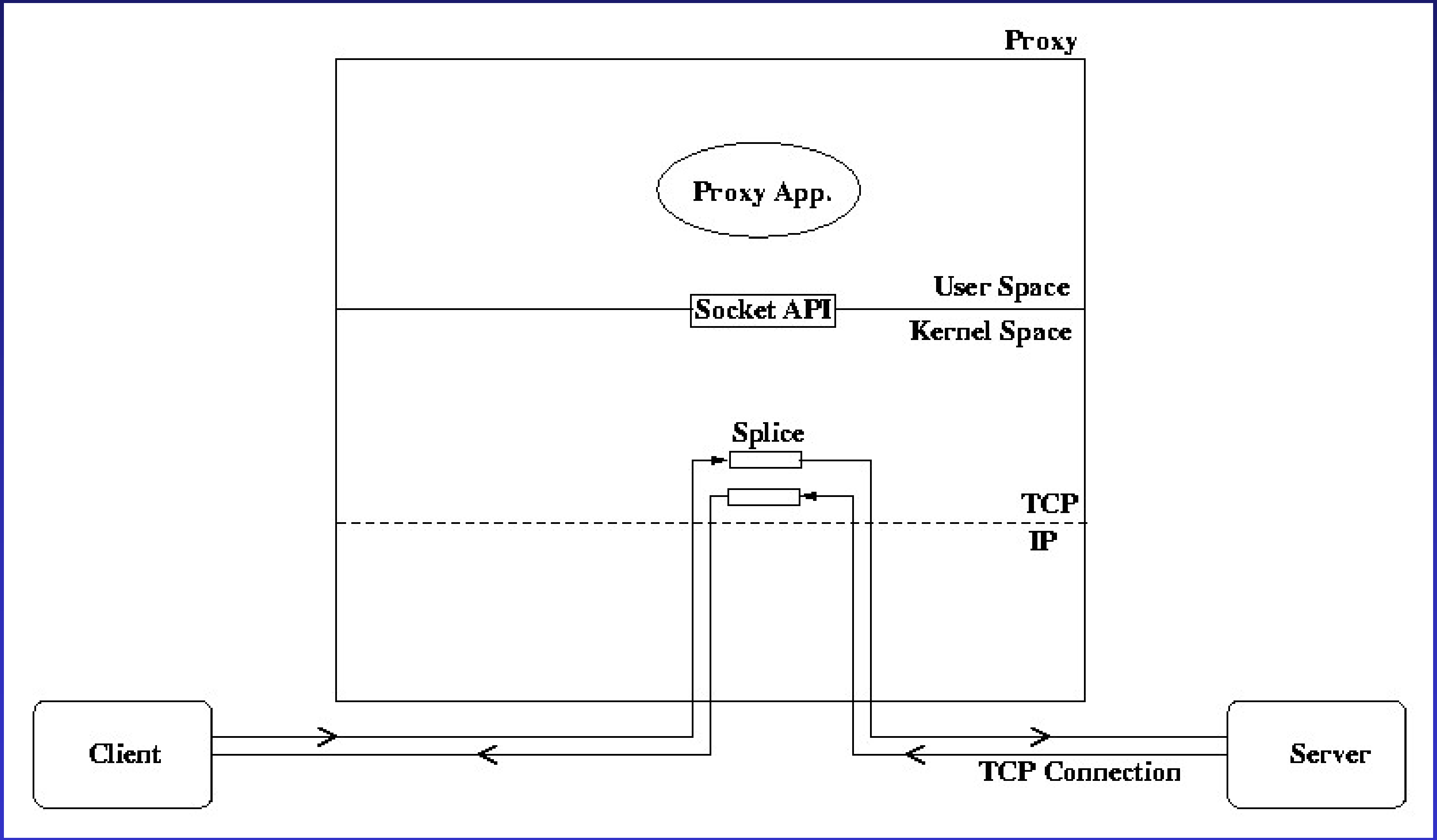
TCP Splice (1)

- Introduced to enhance the performance of web proxies
- Proxy relays data between client and server by manipulating TCP segment headers
- Advantages
  - Done entirely in the kernel
  - Latency and computational cost at proxy only a few times more than IP forwarding
  - End-to-end semantics of the client-server TCP connection preserved
  - No buffering at the proxy

TCP Splice (2)


Proxy
Proxy App.
User Space
Socket API
Kernel Space
TCP
IP
Client
Server
TCP Connection

TCP Splice (3)

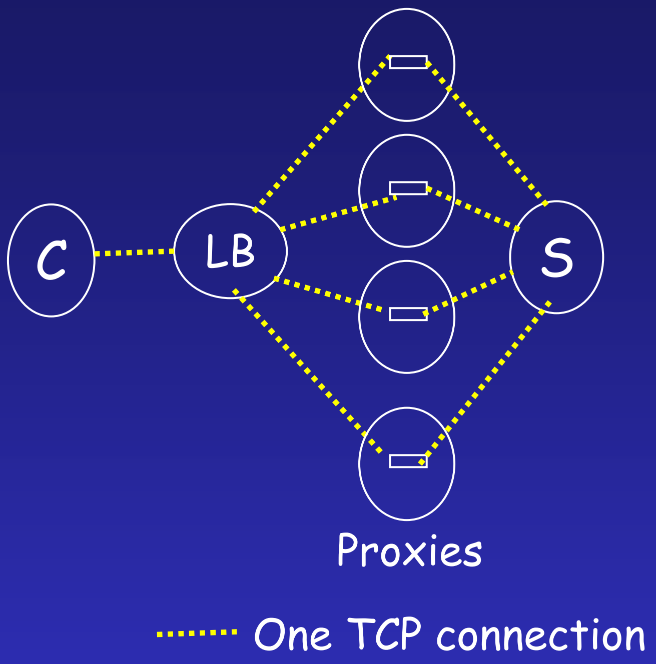


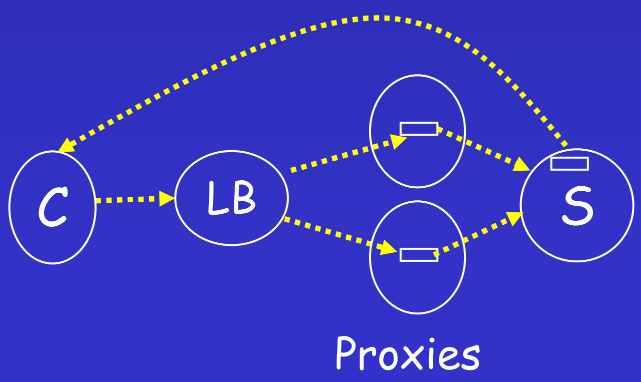
send_seq_num = (recv_seq_num – init_recv_seq_num) + init_send_seq_num

offset

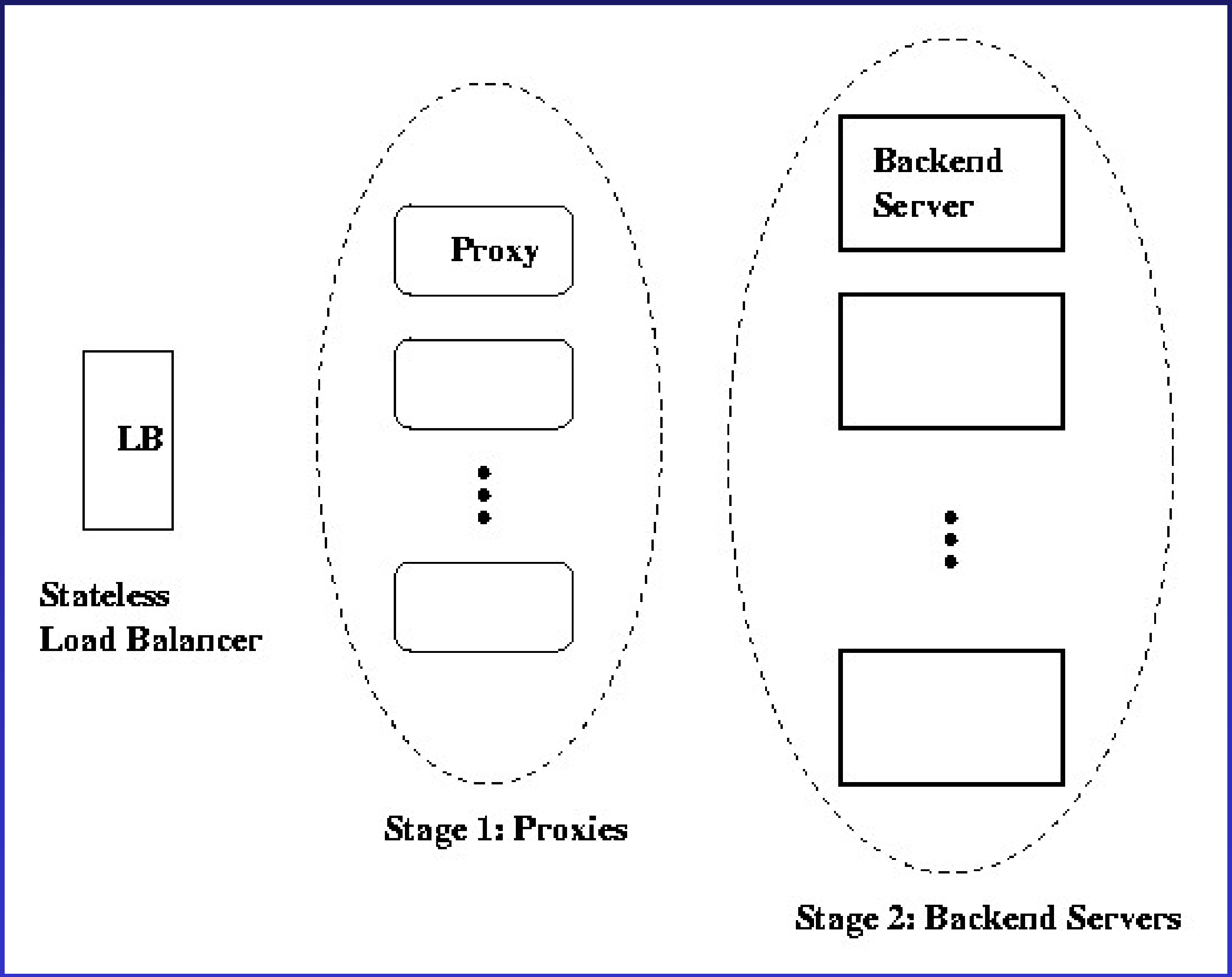
Enhancements to TCP Splice

- Recently, we proposed the following generic enhancements to TCP splice to address
  - Fault tolerance
  - Scalability
- Replicated and Parallel Splice
  - Same connection can be spliced through different machines
- Split-Splice
  - Traffic in the two different directions of the same connection can be spliced at different machines





Our Web Server Architecture

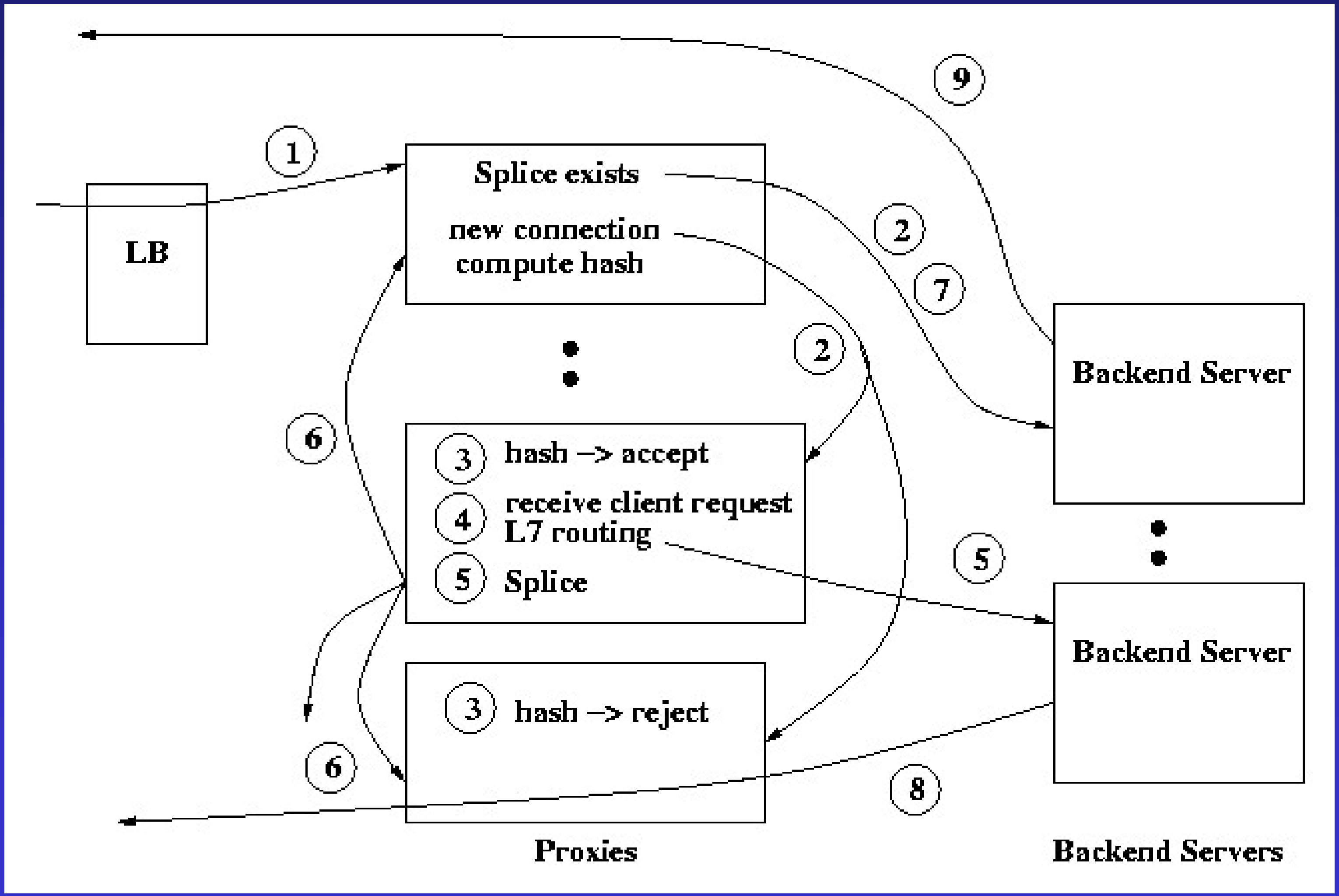


Logical View

Load Balancer

- IP level load balancer (LB)
- No modifications to the packets
- No connection state information is kept
- Completely stateless -> fault tolerance trivial
- Has service IP address
- Right now round-robin algorithm
- Heartbeat mechanism between LB and proxies
  - For failure detection
  - For communicating proxy workload (in future)
- Can be combined with router or proxies

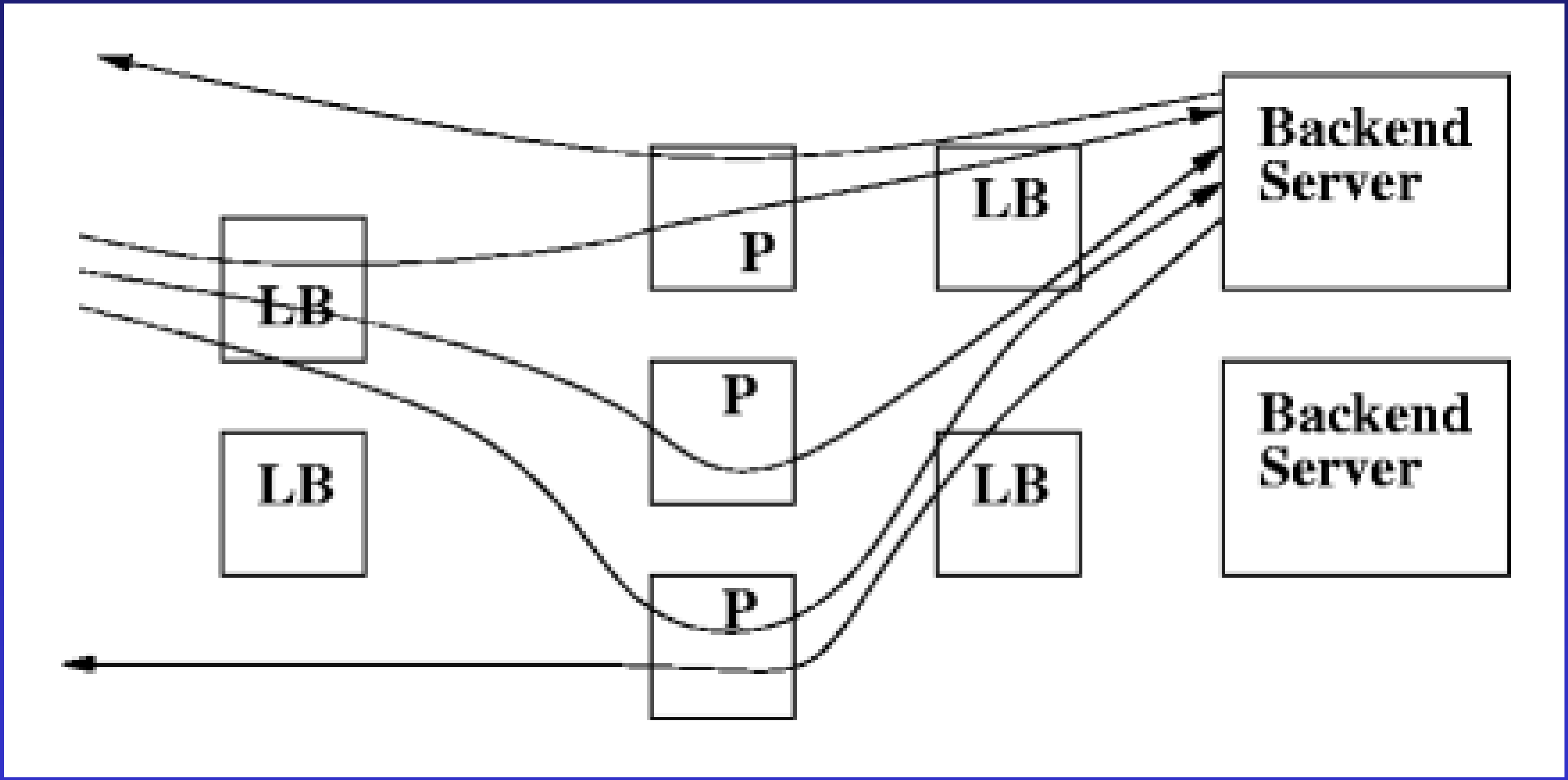
Sequence of Steps involved in Handling a Request



Web Server Configuration 1 (1)

- Separate Proxy and backend server machines
- Traffic in both directions passes through the proxies
- No OS changes required on the backend servers

Web Server Configuration 1 (2)



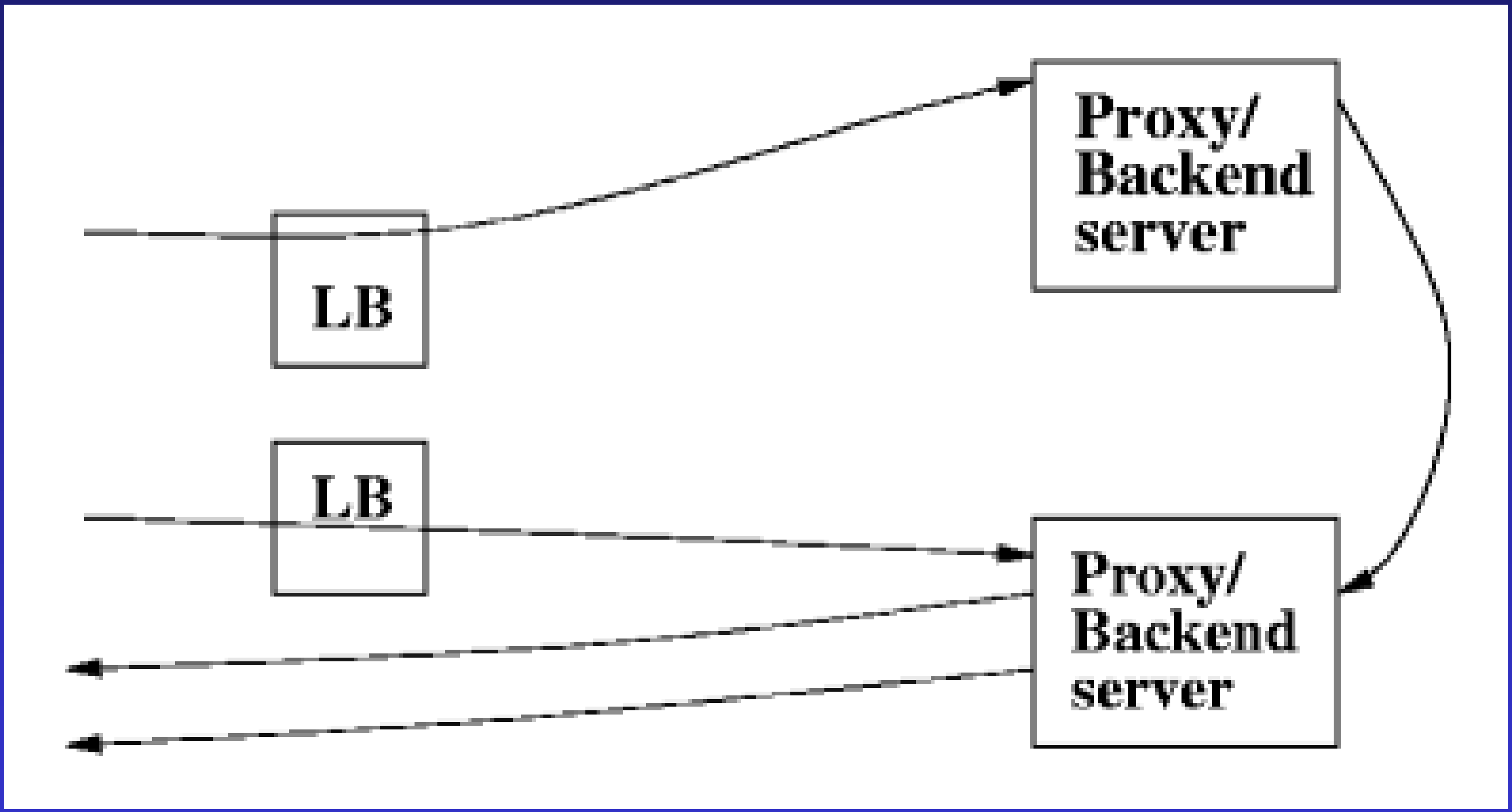
Data Path

One TCP connection

Web Server Configuration 2 (1)

- Co-located Proxy and backend server machines
- Separate proxy machines not required
- Saves HW
- OS changes required on the servers

Web Server Configuration 2 (2)



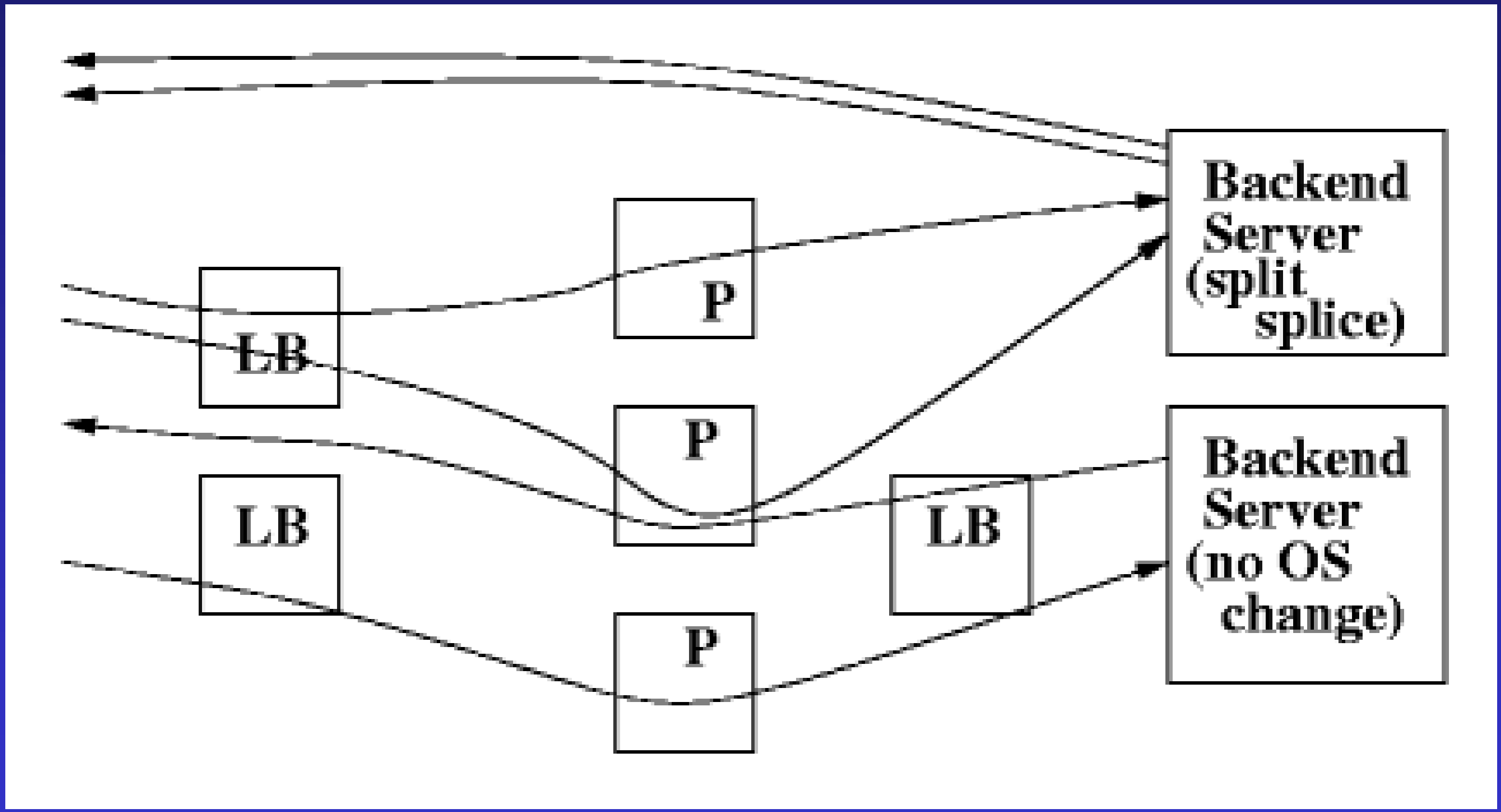
Data Path

One TCP connection

Web Server Configuration 3 (1)

- Mixed configuration: OS changes are required on some machines
- Backend servers where OS can be changed do split-splice

Web Server Configuration 3 (2)



→ Data Path

Two TCP connections

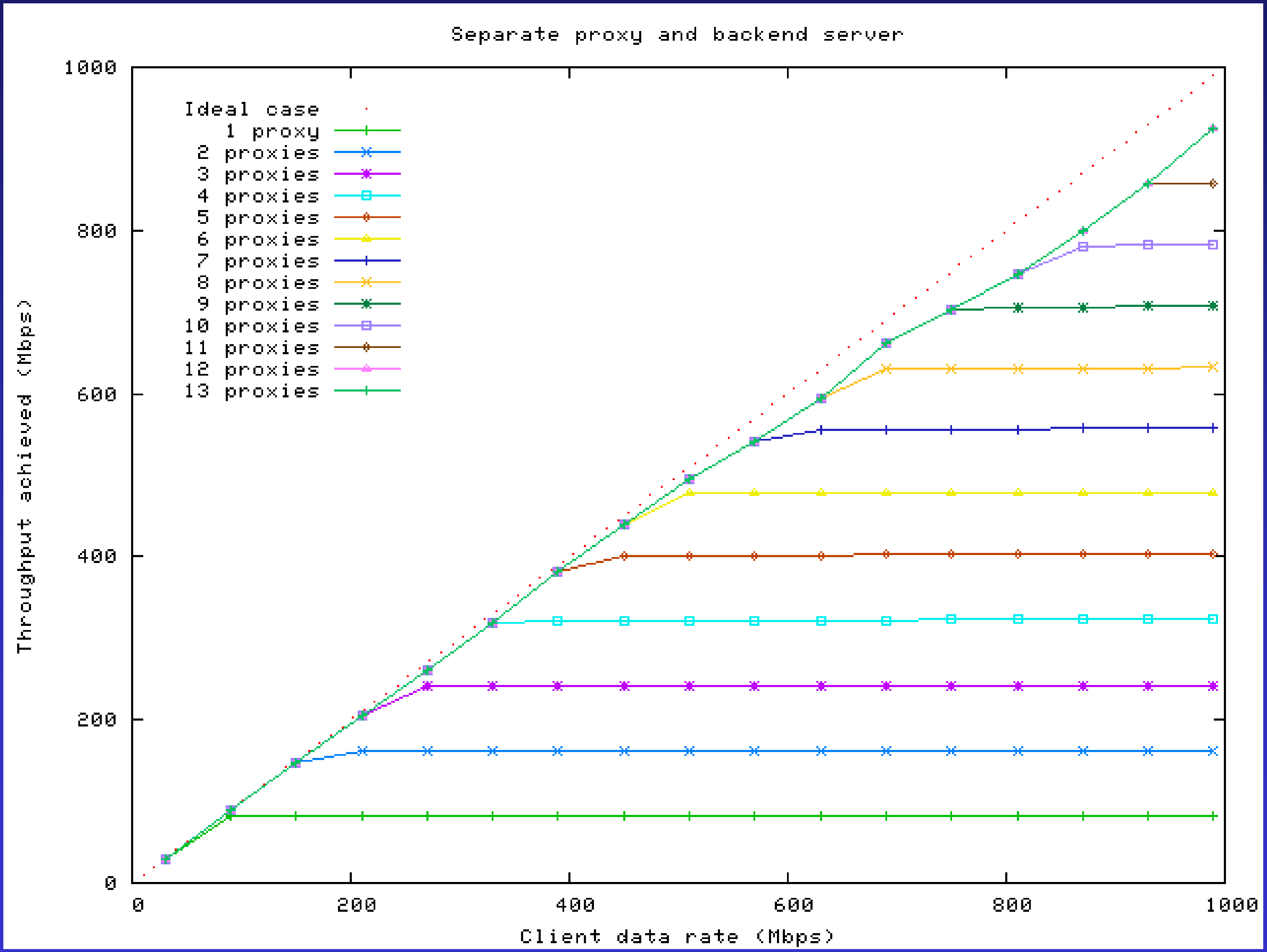
Simulations

- A discrete event simulator was written to simulate the architectures
- Goal of the simulations is to show scalability
- Connections simulated are assumed to be already established and spliced
- Some simulation parameters
  - Splicing Cost: 25 μ sec
  - Client – Proxy link delay: 25 ms
  - Proxy – Server link delay: 0.7 ms
  - TCP buffer: 256kB
  - Packet size: 1460 B
  - Q processing Rate: 1 Gbps

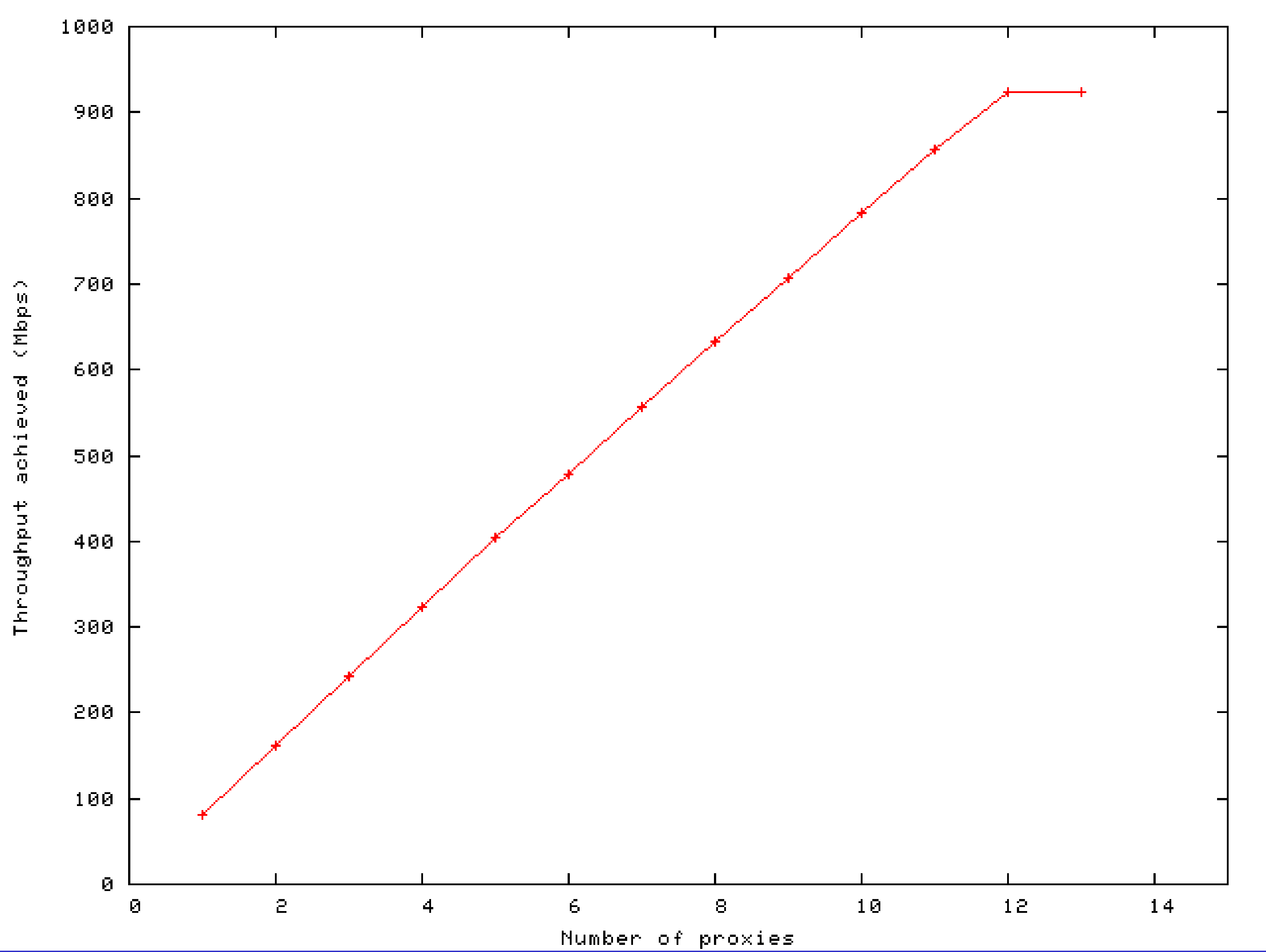
Separate proxy and back-end server (1) (Web Server Configuration 1)

- Goal: show scalability of architecture
- Client
  - data is generated from 30 Mbps to 990 Mbps in intervals of 60 Mbps
  - 50 MB is transferred at each data rate
- Proxies
  - Varied from 1 to 13
- 1 Backend server used

Separate proxy and back-end server (2)



Separate proxy and back-end server (3)



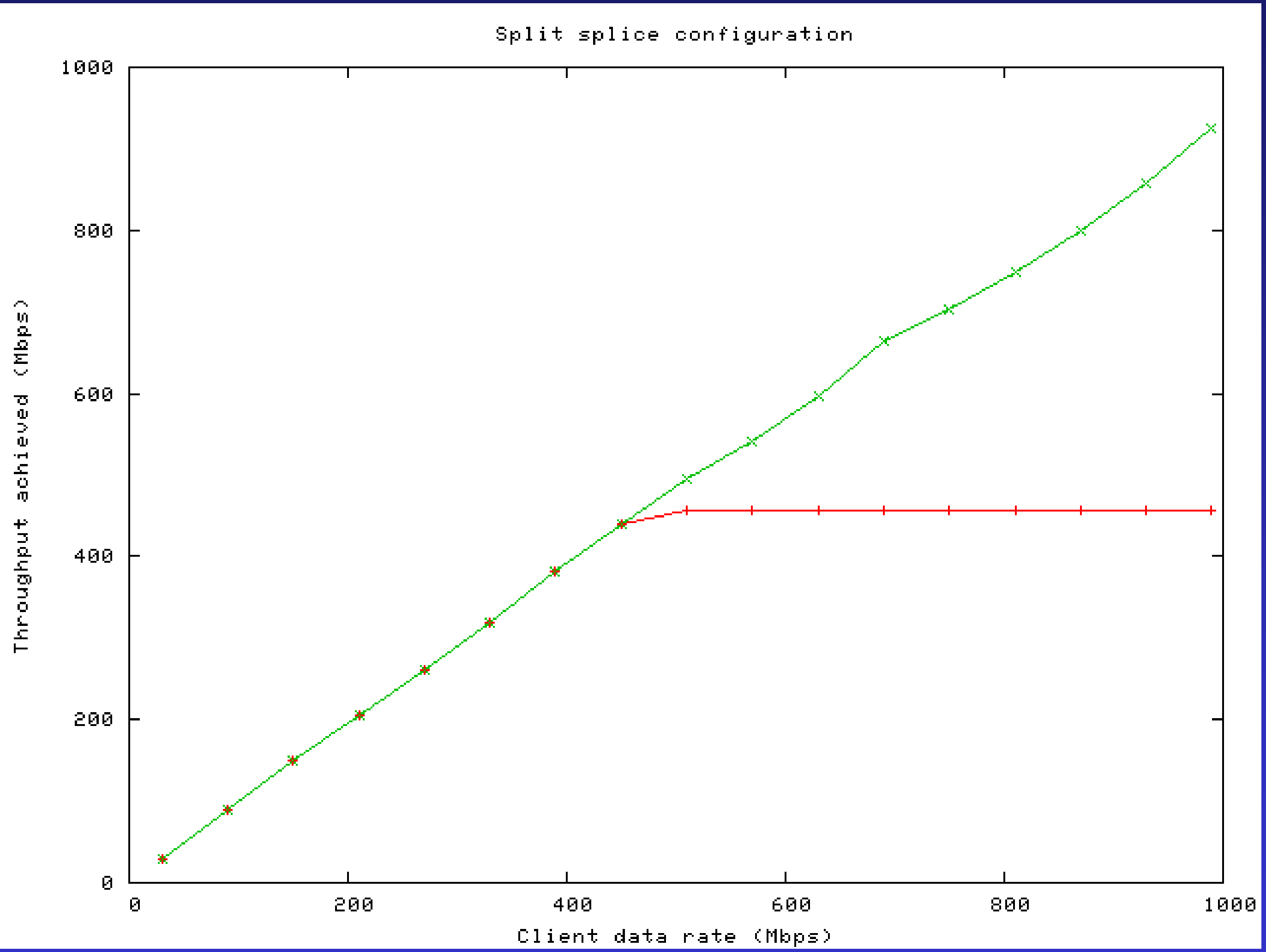
Co-located proxy and back-end server (Web Server Configuration 2)

- The same experiments were repeated for this scenario
- Proxy – Backend server link delay was made zero
- Almost identical to separate proxy and backend server scenario since link delay was already small

Split Splice (1)

- Same experiments were repeated for split splice
- Data generated at a backend server, spliced and sent directly to client

Split-Splice (2)



Splicing cost: 25 μ sec

Splicing cost: 2 μ sec

Conclusions and Future Work

- Presented web server architectures based on enhanced TCP splice
- Used simulations to evaluate these architectures
- In particular, simulated three different configurations
  - Scale well
  - Connections preserved even on proxy failure
  - Proxy not a bottleneck
- Linux prototype implementation paper to be presented at SRDS '06
- In future, make architecture tolerate backend server failures while preserving client connection

Backup Slides

Proxies

- Perform TCP splicing
- Before a connection is spliced, all packets of a particular connection are received by the same proxy through use of hashing and multicast groups (details in SRDS '06 paper)
- Proxy establishing splice distributes splicing state info to other proxies
- Once a connection is spliced, a packet of that connection can be spliced at *any* proxy