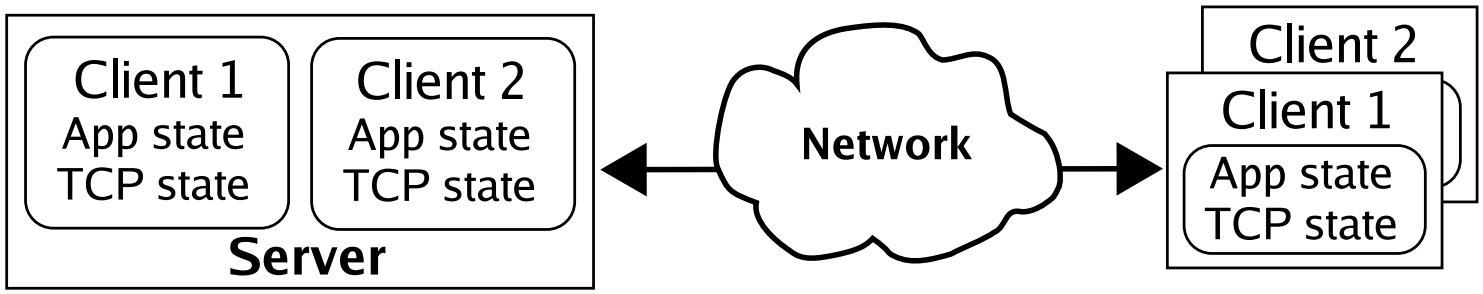
Trickles: A Stateless Network Stack for Improved Scalability, Resilience, and Flexibility

Alan Shieh
Andrew C. Myers
Emin Gün Sirer

Department of Computer Science
Cornell University

State placement is important

- In typical network stacks, both servers and clients hold per-connection state



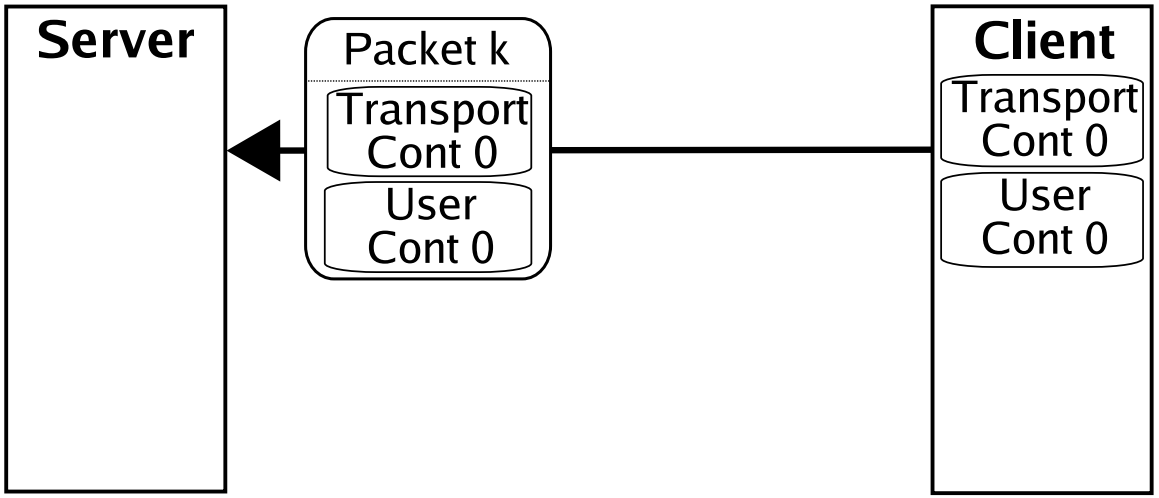
- State binds connections to endpoints
  - Limits flexibility
  - Poses barrier to migration
- State requires resources
  - Limits scalability
  - Increases vulnerability to DoS attack

Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data

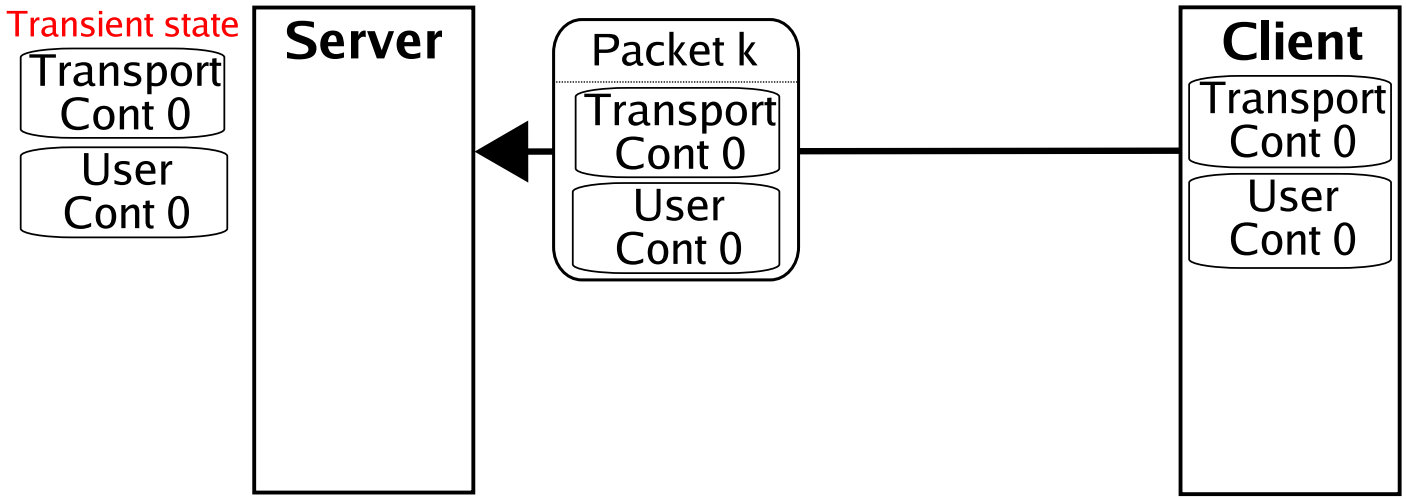
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



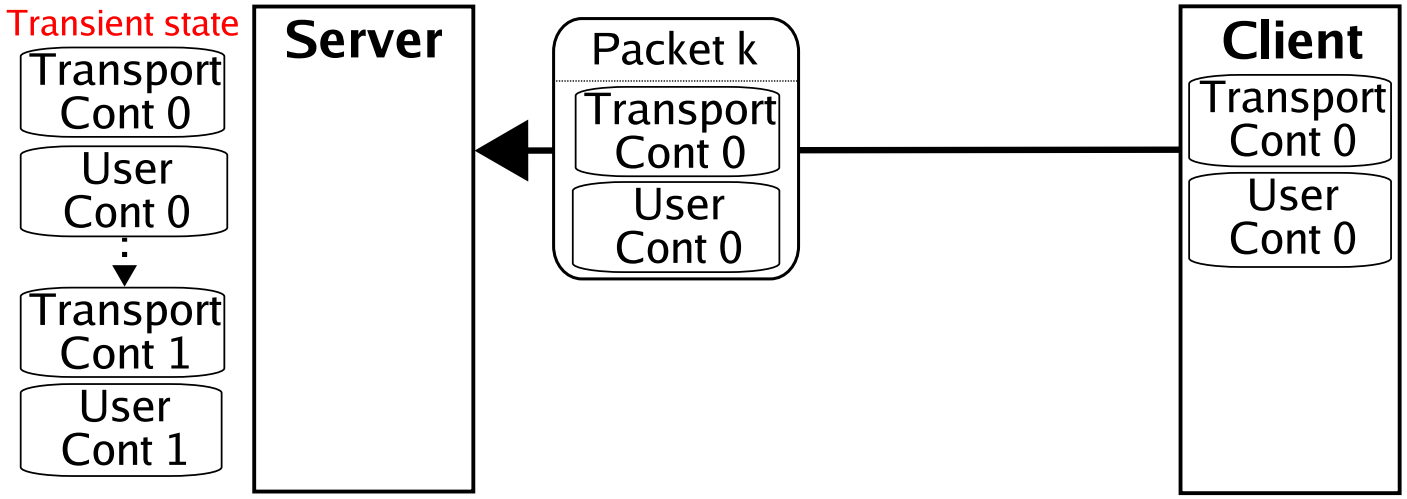
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



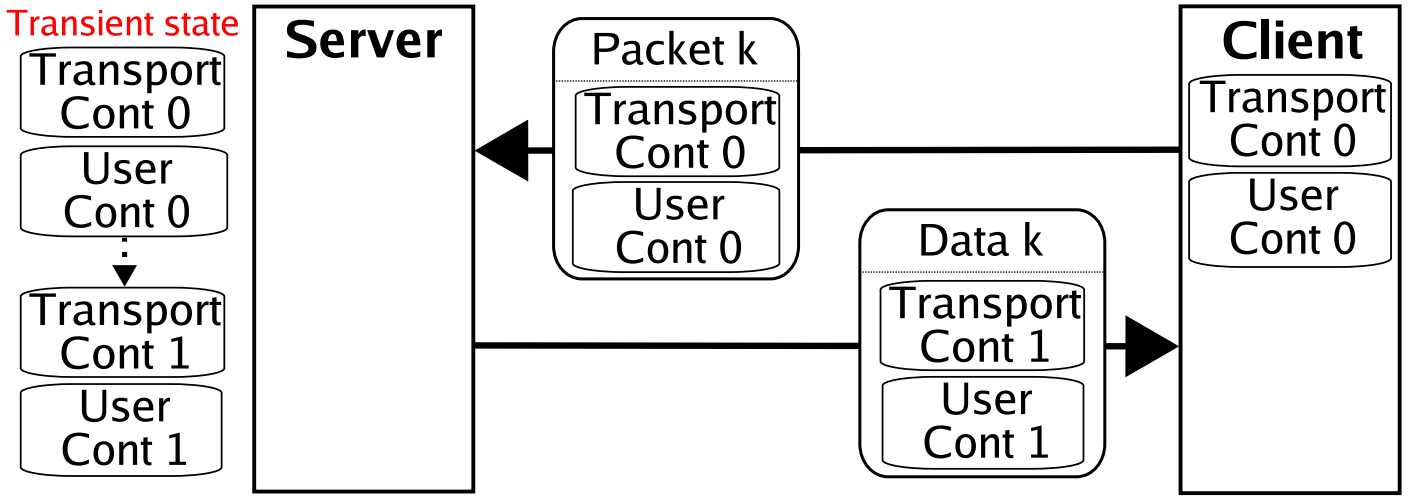
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



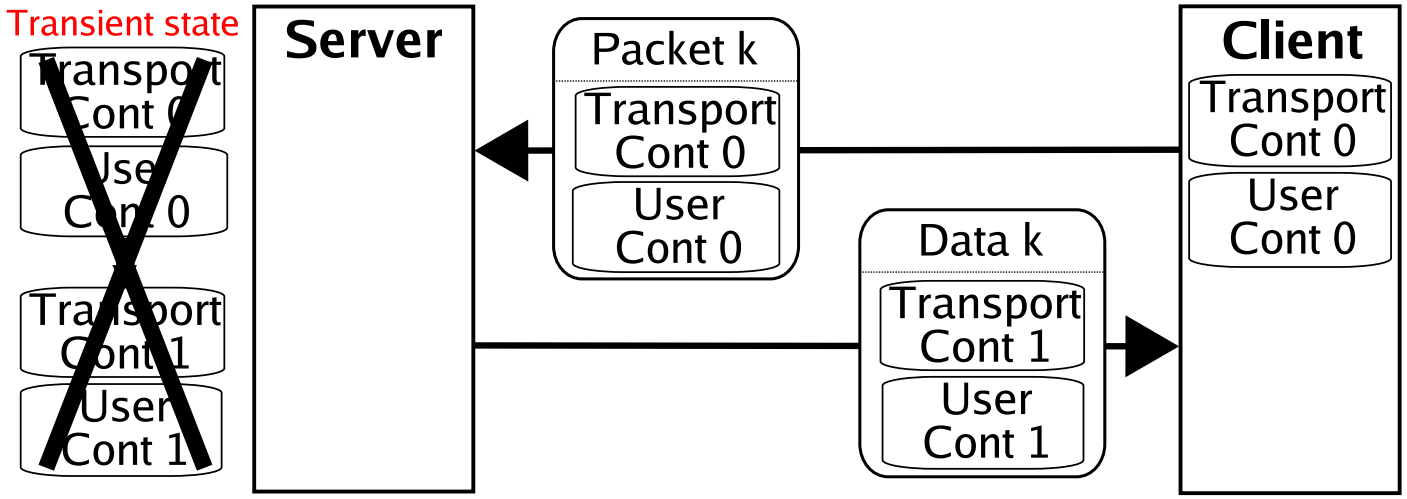
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



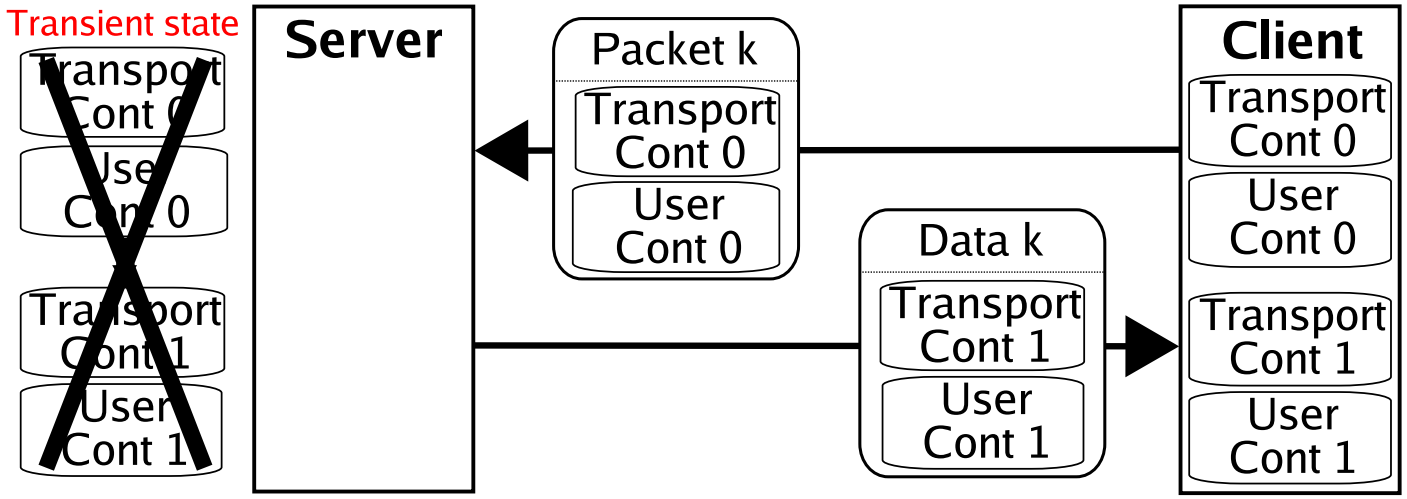
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



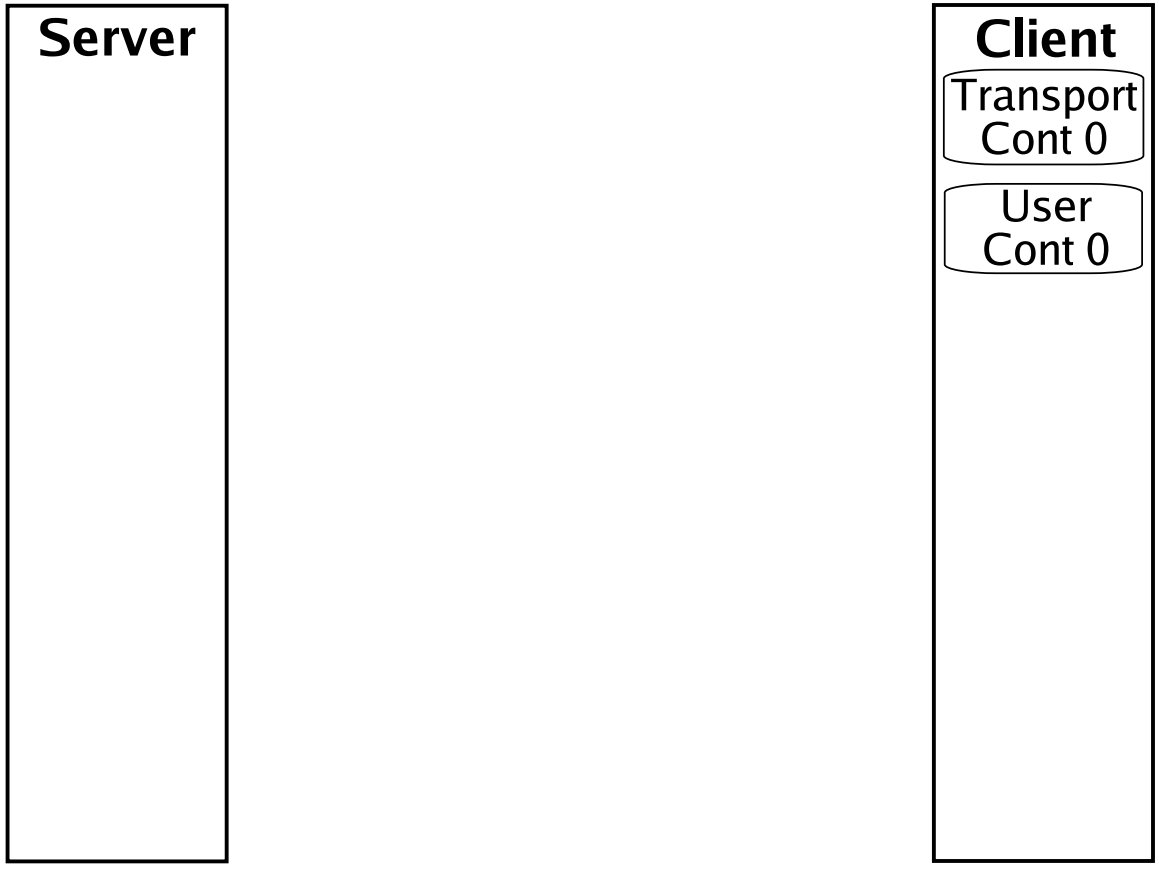
Trickles approach: Redistribute state

- Make one endpoint (server) completely stateless
  - Continuations represent a suspended computation
  - Encapsulate state with continuations
  - Migrate continuations to other endpoint
  - Continuation state periodically updated
- **Transport continuations**: congestion control
- **User continuations**: application data



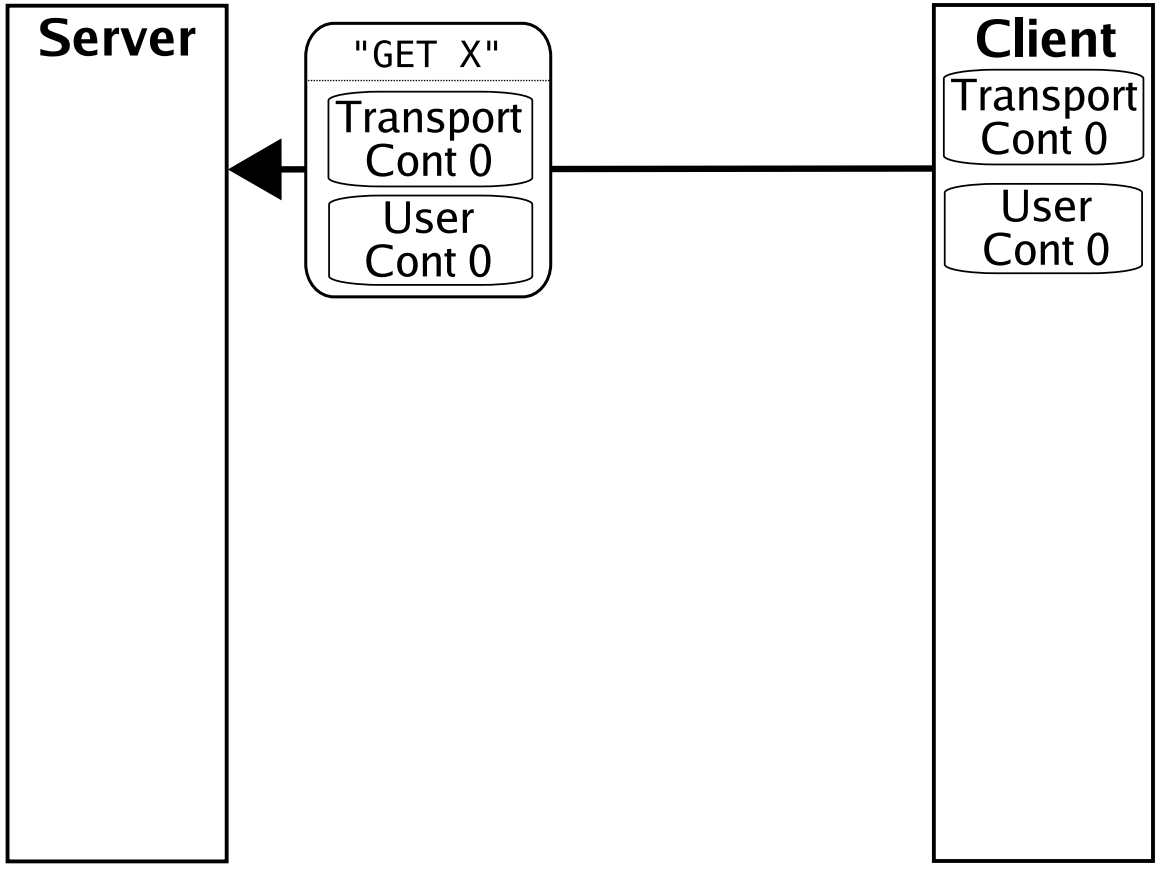
Continuation passing

- Packets contain previously-generated continuation
- Data packets contain updated continuations



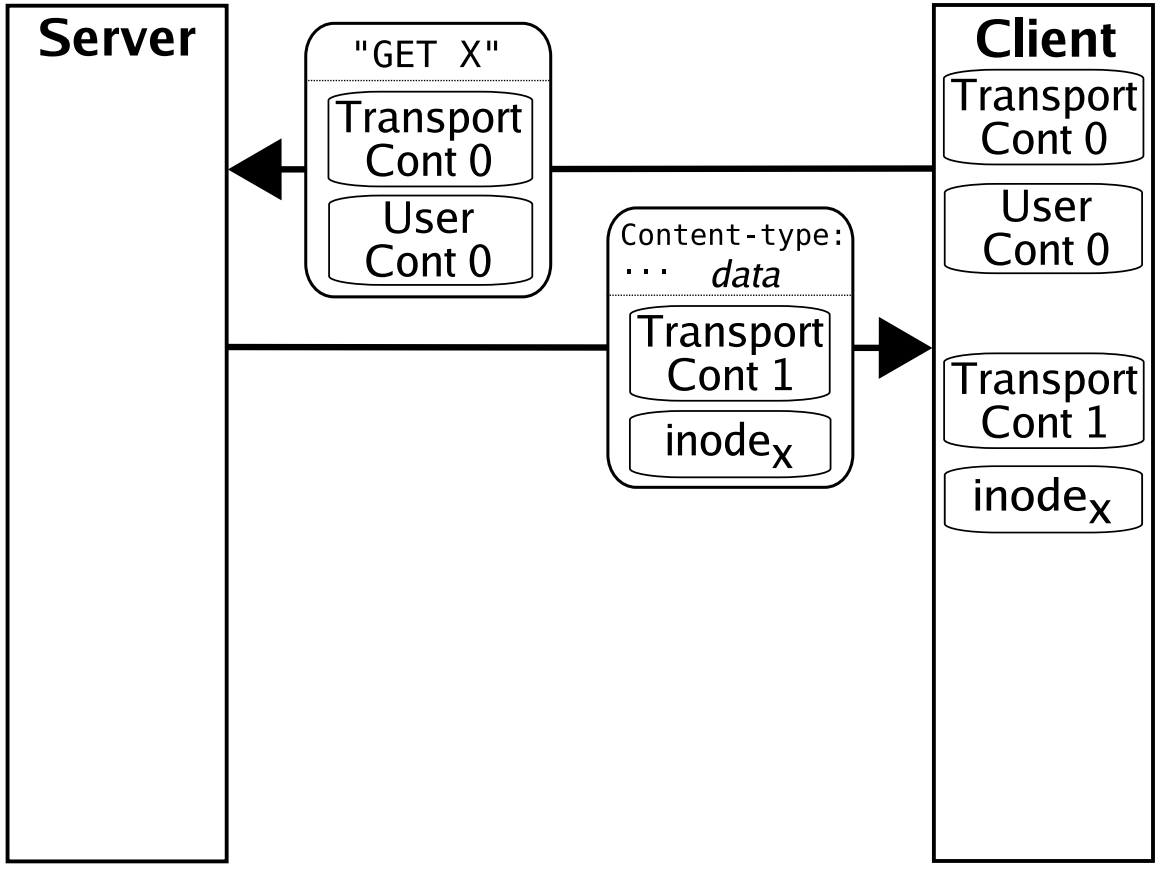
Continuation passing

- Packets contain previously-generated continuation
- Data packets contain updated continuations



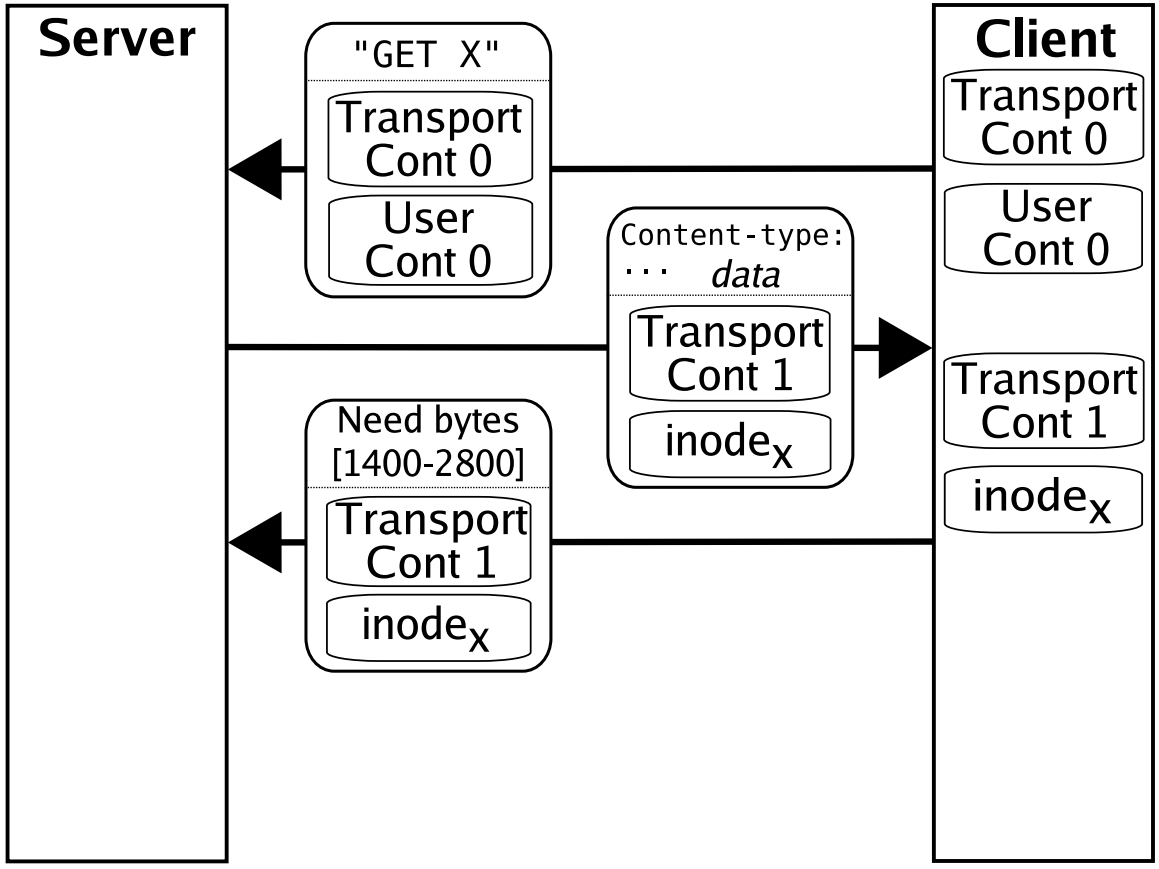
Continuation passing

- Packets contain previously-generated continuation
- Data packets contain updated continuations



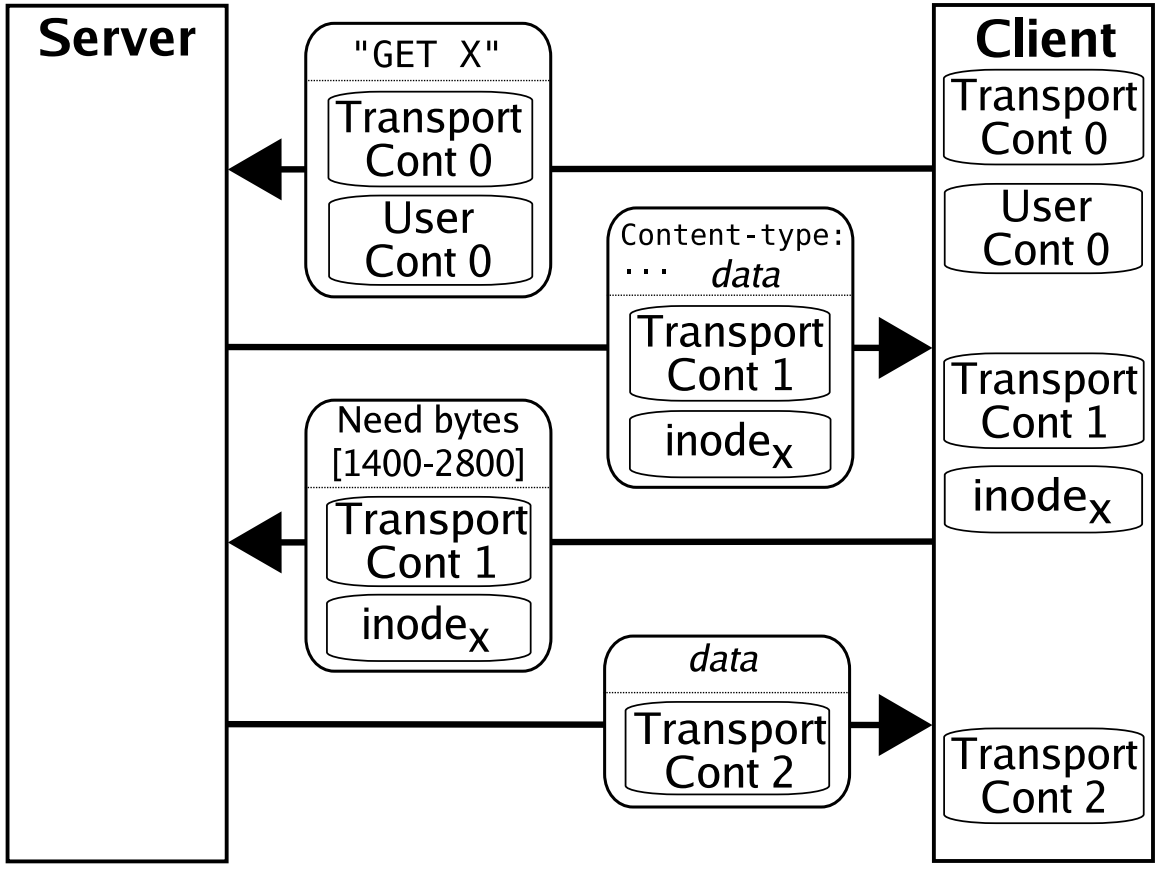
Continuation passing

- Packets contain previously-generated continuation
- Data packets contain updated continuations



Continuation passing

- Packets contain previously-generated continuation
- Data packets contain updated continuations



Architectural motivation

- TCP+Sockets is inherently stateful
  - Every TCP server requires per-connection state

- Trickles replaces TCP and sockets
  - New protocol
  - New server API
  - Backwards compatible on client
  - Server-side stateless
  - Connection-oriented
  - Per-client session state

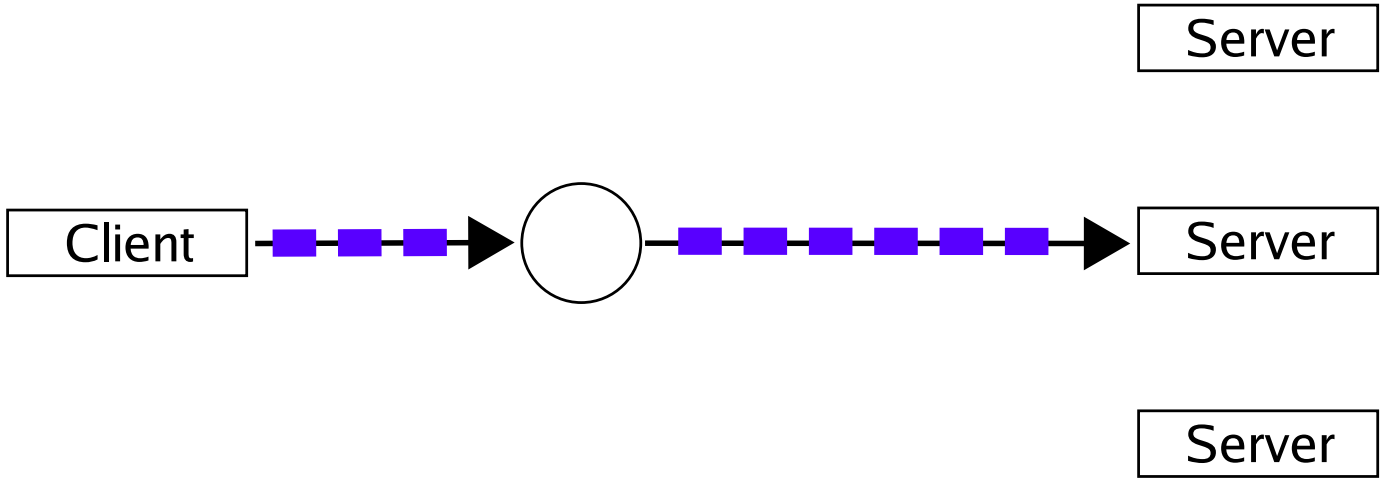
Advantages of statelessness

- Improved scalability
- Any server replica can service any packet
  - Transparent failover
  - Load balancing
  - Anycast services

Transparent failover

- Transparent failover
- Load balancing
- Anycast services

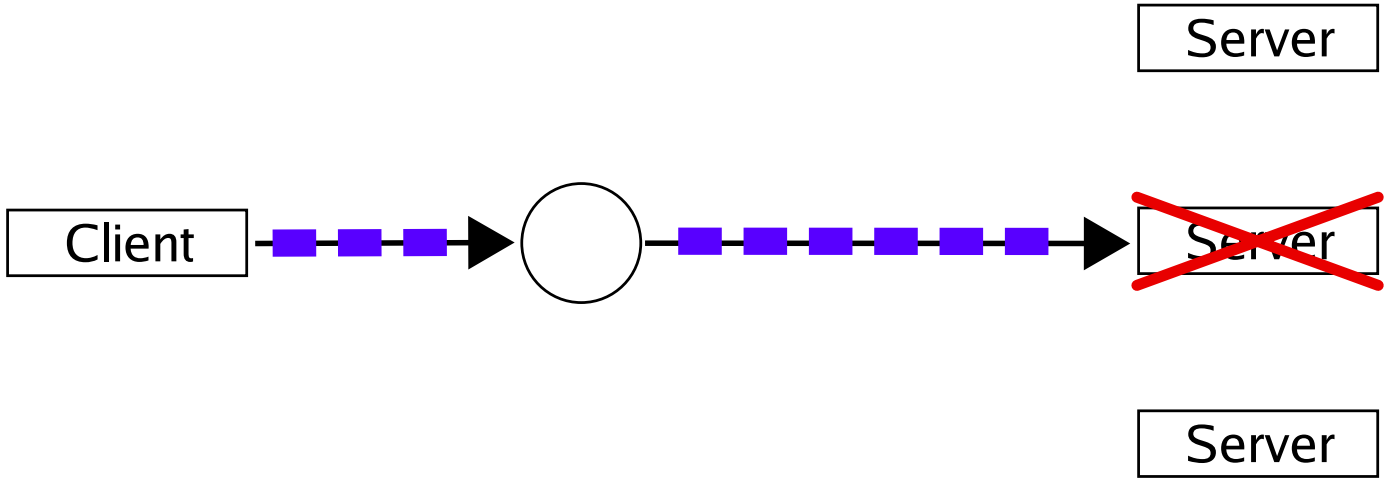
- Network layer provides transparent failover



Transparent failover

- Transparent failover
- Load balancing
- Anycast services

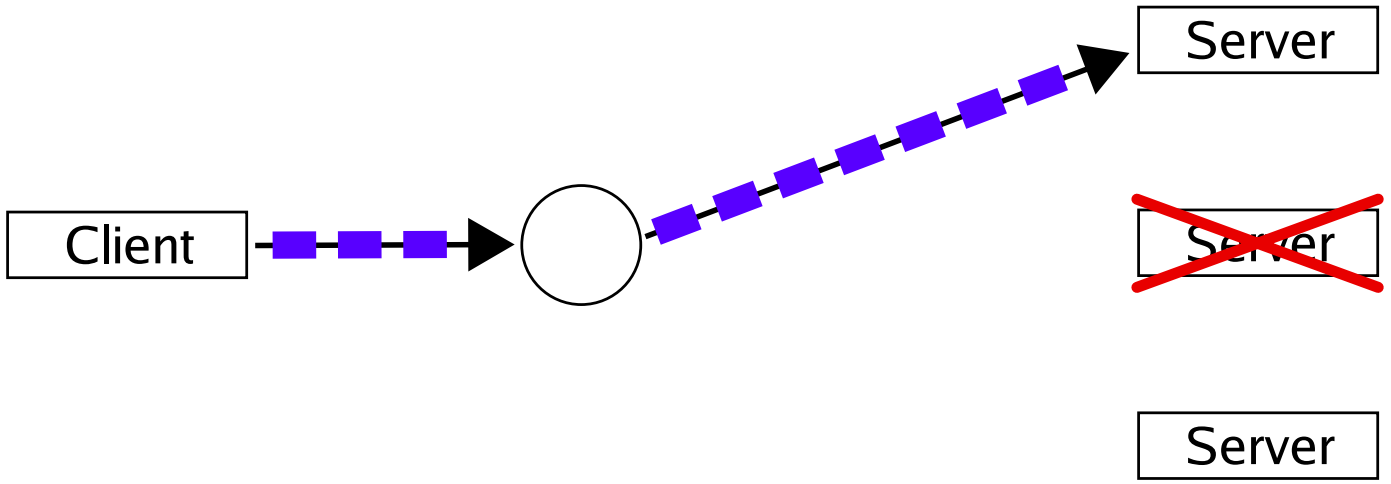
- Network layer provides transparent failover



Transparent failover

- Transparent failover
- Load balancing
- Anycast services

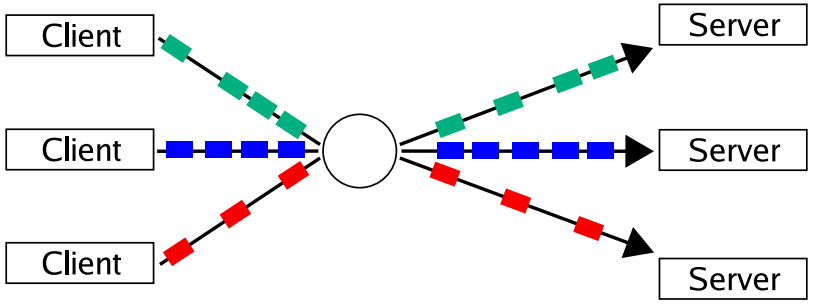
- Network layer provides transparent failover



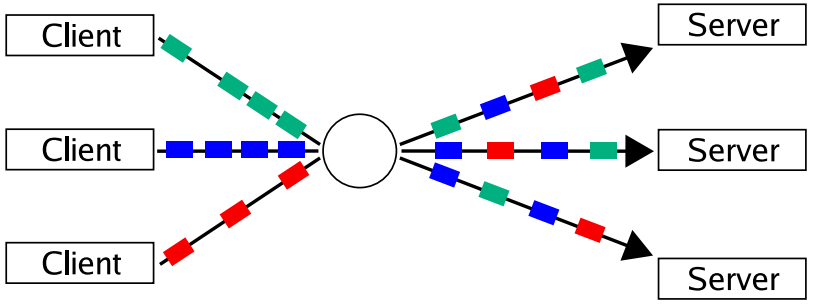
Load balancing

- Transparent failover
- Load balancing
- Anycast services

- TCP: connection granularity
  - Load balance at connection time



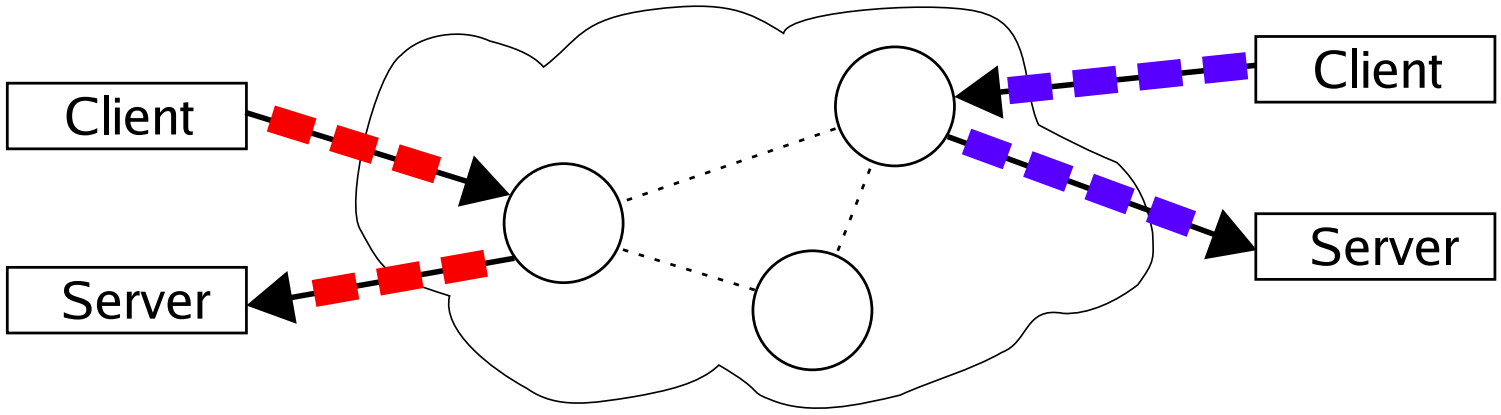
- Trickles: packet granularity
  - Load balance on any packet



Geographic anycast

- Transparent failover
- Load balancing
- Geographic anycast

- Network independently routes packets from clients to closest anycast node



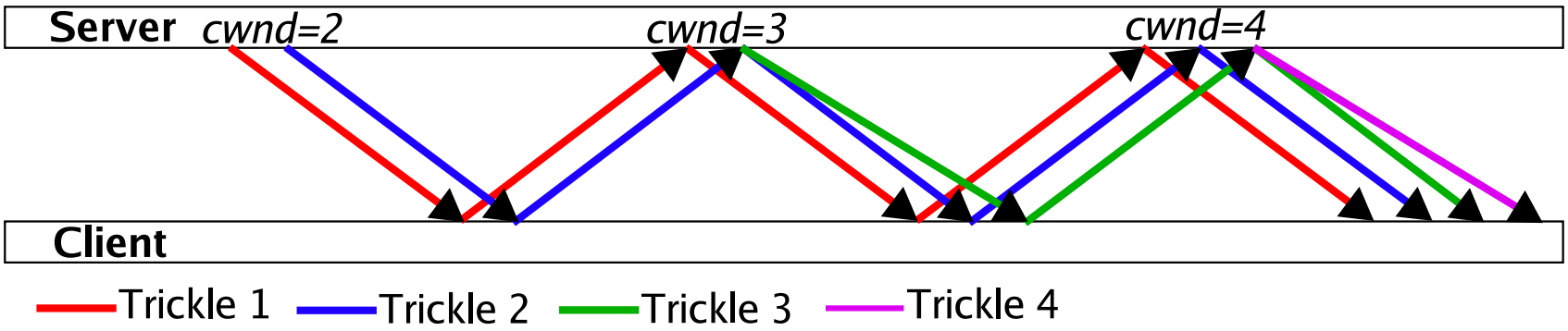
Target applications

- Application properties
  - Compact server-side state
  - Client/server with large numbers of clients
  - Data transfer occurs predominantly from server to client
- Dynamic and static web servers
- Network services based on flexible redirection

Designing a stateless protocol

The trickle abstraction

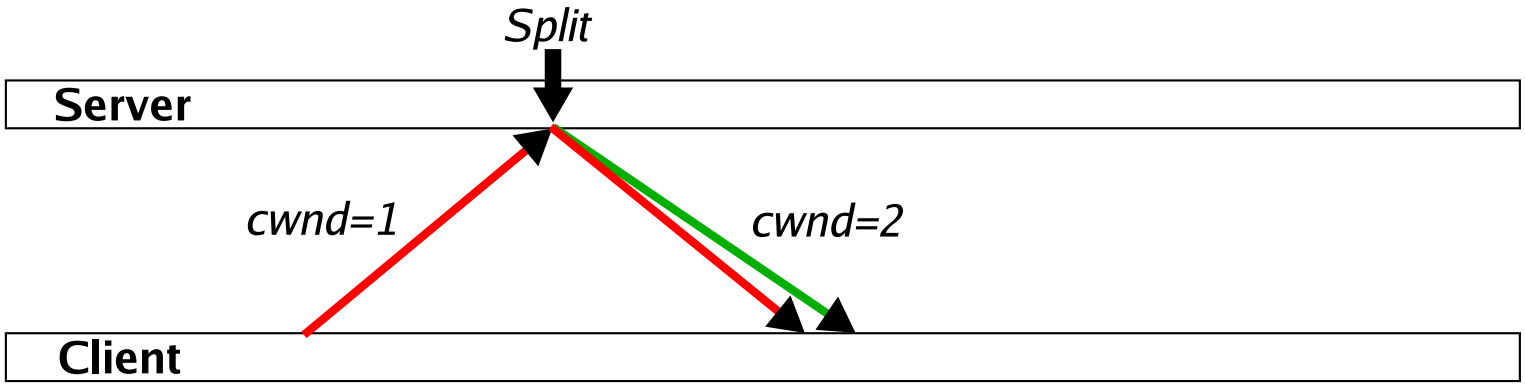
- Trickles protocol has multiple active states at any time
  - One continuation per in-flight packet



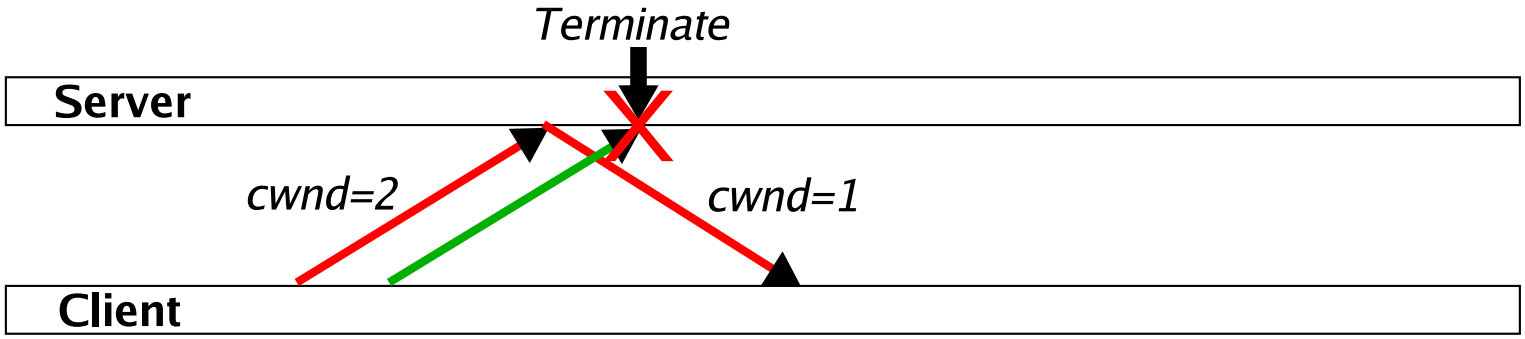
- A **trickle** is a sequence of request/response pairs
  - Captures the data and control flow properties
  - A Trickles connection is partitioned into a set of *cwnd* trickles, one trickle per in-flight packet
- *cwnd* corresponds to the number of trickles

Trickle operations

- **Splitting** increases the window



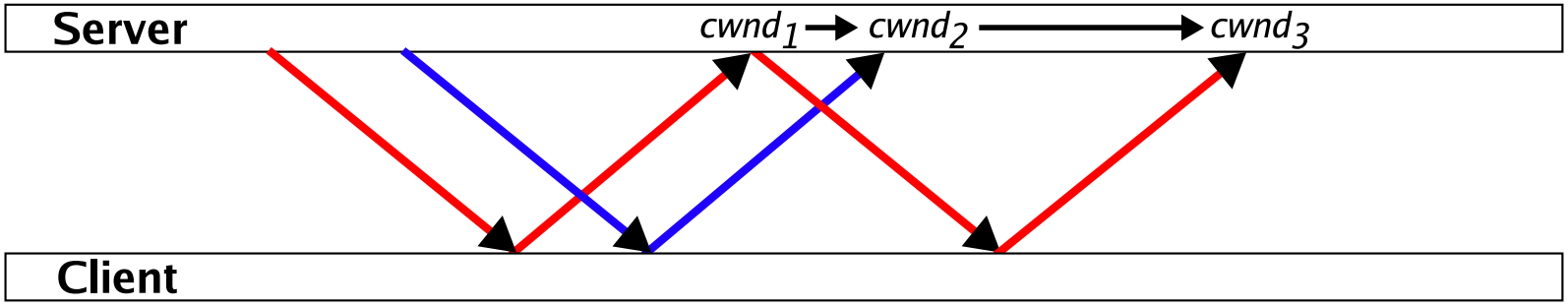
- **Terminating** decreases the window



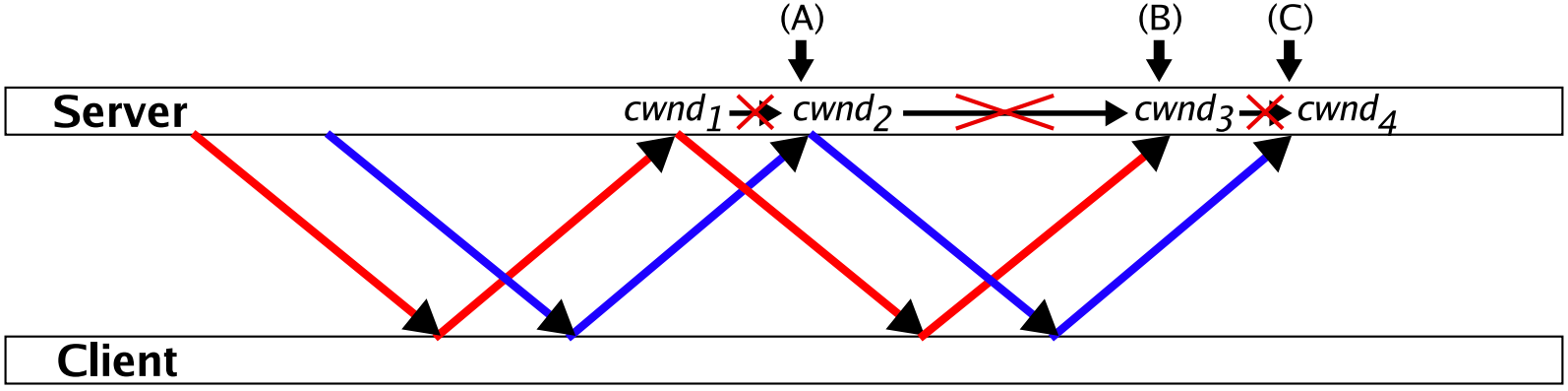
- Congestion control reduces to determining when to **split** and **terminate**

Stateless information constraints

- TCP: persistent state

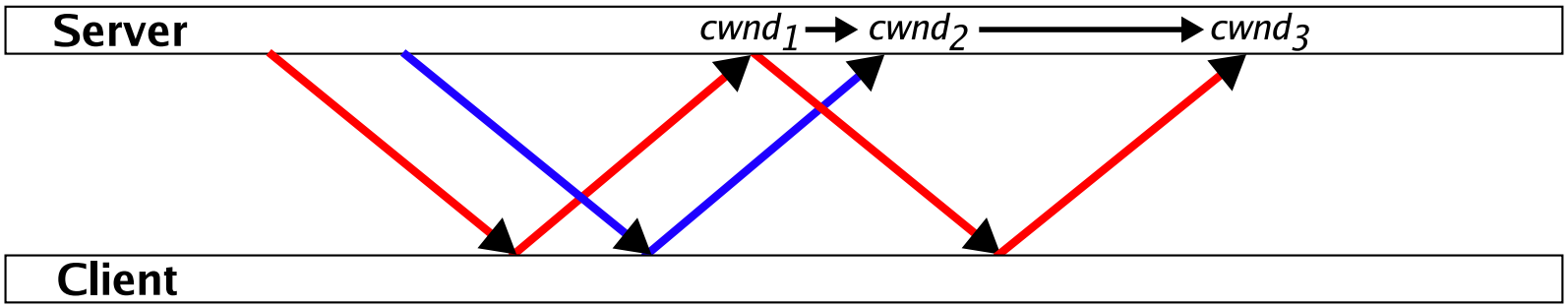


- Trickles:
  - Server only knows state sent by client
  - State does not flow between different trickles
  - Client merges state from multiple trickles

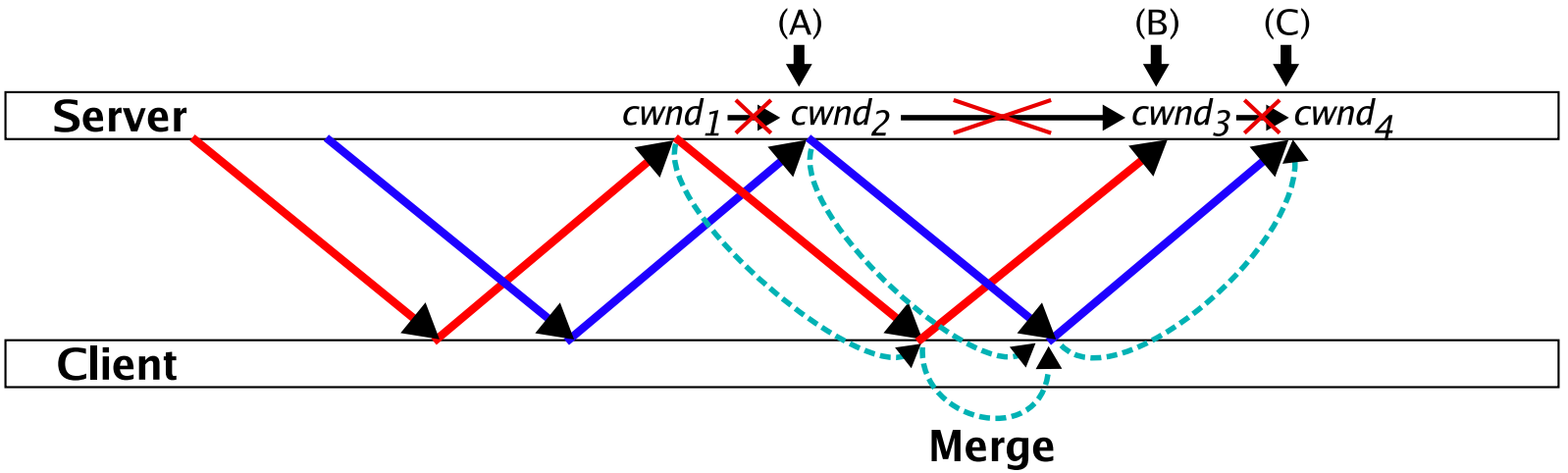


Stateless information constraints

- TCP: persistent state

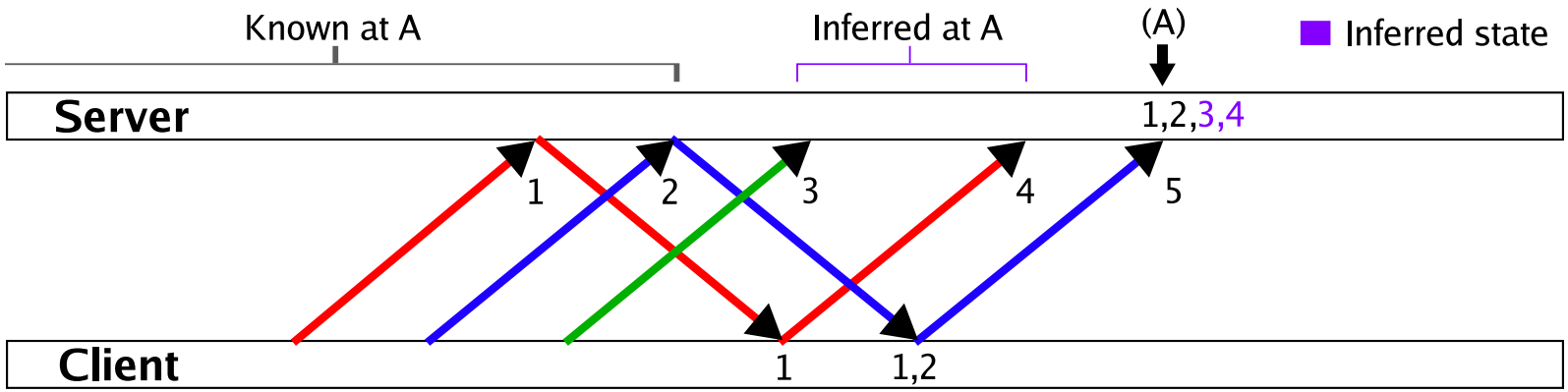


- Trickles:
  - Server only knows state sent by client
  - State does not flow between different trickles
  - Client merges state from multiple trickles



Solution: State inference

- Server infers state on each packet
  - Assume inputs in previous packets
  - Infer state from input



- Use merged and inferred state to process packet
- Recover if incorrect inference detected later

Protocol phases

- Trickles protocol phases match TCP Reno
  - Slow start
  - Congestion avoidance
  - Fast retransmit/recovery

Slow start and congestion avoidance

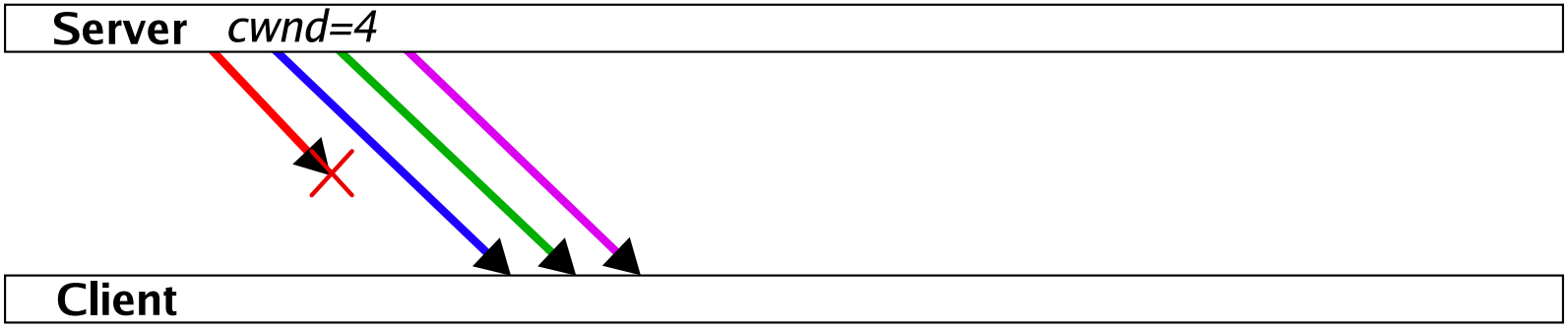
- Goal: Increase and utilize congestion window
  - Slow start: increase on every packet
  - Congestion avoidance: increase every *cwnd* packets
- Split trickle to generate the same number of response packets as TCP
- Simulate TCP action at each packet (conceptually)

Closed form optimization

- Naïve simulation is inefficient
- Use equivalent closed-form expression to simulate loss-free epochs in $O(1)$-time
- Parameters are used to compensate for losses between epochs
  - Initial sequence number
  - Initial *cwnd*
  - *ssthresh*

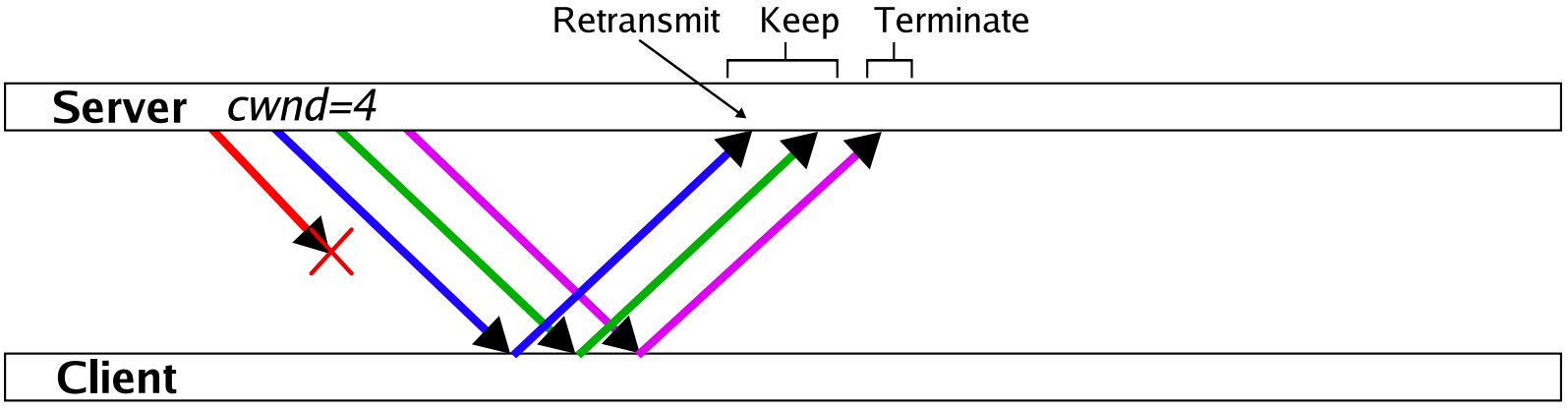
Fast retransmit/recovery

- Goal: Retransmit lost data and halve *cwnd*
  - Compute *cwnd* / 2
  - Avoid repeated/omitted actions
- Infer consistent view of where losses occurred
- All trickles use same deterministic plan
  - Each trickle executes its share of plan



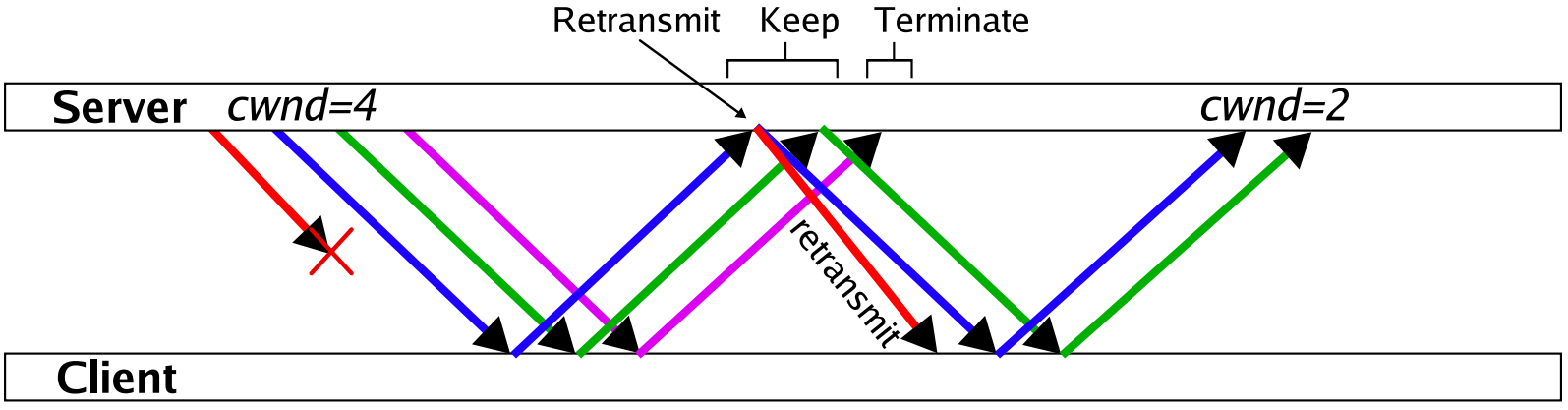
Fast retransmit/recovery

- Goal: Retransmit lost data and halve *cwnd*
  - Compute *cwnd* / 2
  - Avoid repeated/omitted actions
- Infer consistent view of where losses occurred
- All trickles use same deterministic plan
  - Each trickle executes its share of plan



Fast retransmit/recovery

- Goal: Retransmit lost data and halve *cwnd*
  - Compute *cwnd* / 2
  - Avoid repeated/omitted actions
- Infer consistent view of where losses occurred
- All trickles use same deterministic plan
  - Each trickle executes its share of plan



Security

- Preserve integrity with tamper-resistant MAC
- Preserve confidentiality with encryption
- Keep recent packet history to prevent replay of old continuations
- Use range nonces on SACKs to prevent clients from hiding losses

Efficient range nonces

- Prevent client from hiding losses in SACKs
- Nonce attached to packet k by server, computed from secret sequence r_j

$$p_k = r_k \oplus r_{k+1}$$

  - O(1) generation time, O(1) size per nonce
- Nonce from client ACKing packets 1 to n

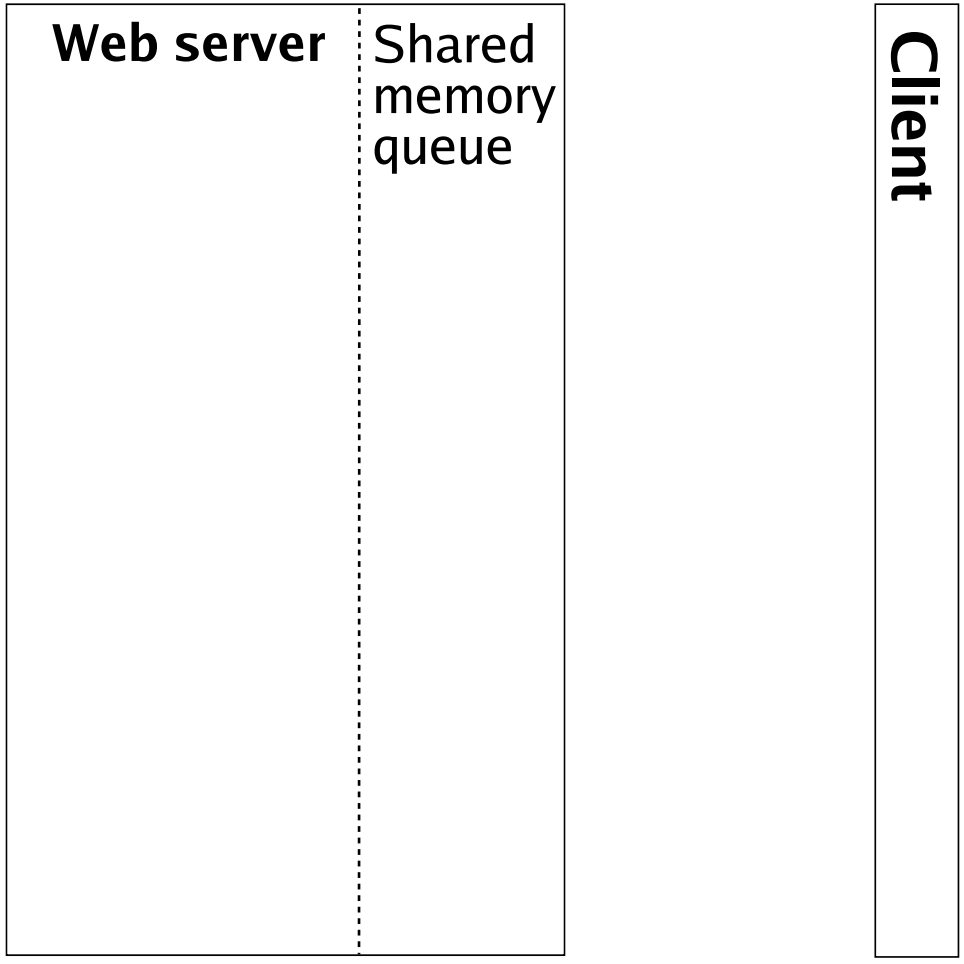
$$\begin{aligned} p_{1,n} &= p_1 \oplus p_2 \oplus \ldots \oplus p_n \\ &= (r_1 \oplus r_2) \oplus (r_2 \oplus r_3) \oplus \ldots \oplus (r_n \oplus r_{n+1}) \\ &= r_1 \oplus (r_2 \oplus r_2) \oplus (r_3 \oplus r_3) \oplus \ldots \oplus (r_n \oplus r_n) \oplus r_{n+1} \\ &= r_1 \oplus r_{n+1} \end{aligned}$$

  - O(1) validation time, O(1) size per range

Trickles server API

Stateless, event-based server API to replace sockets

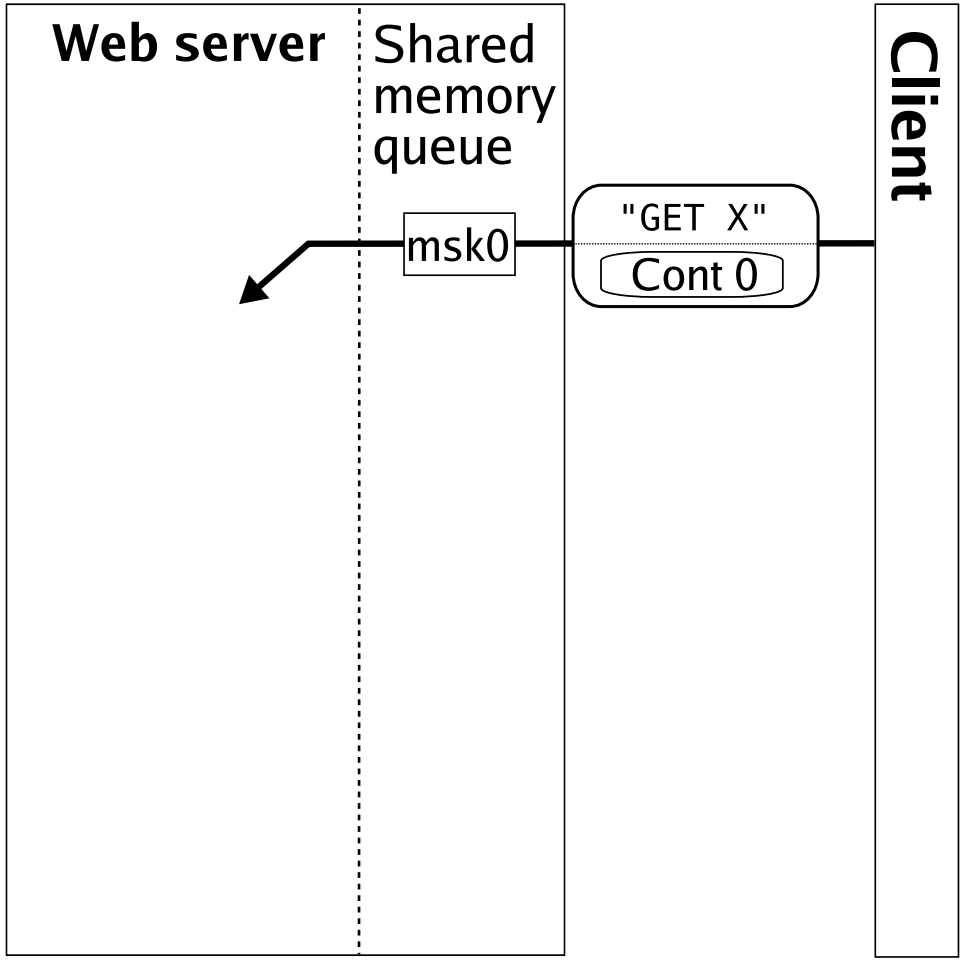
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

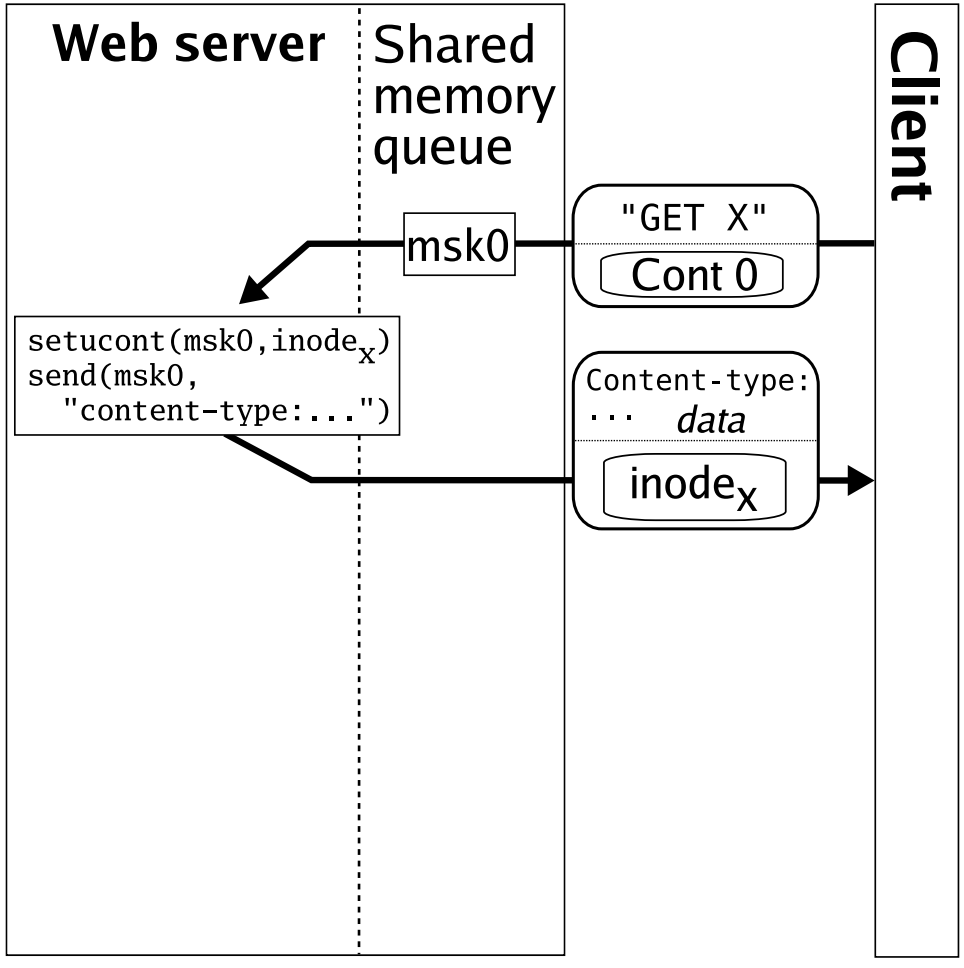
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

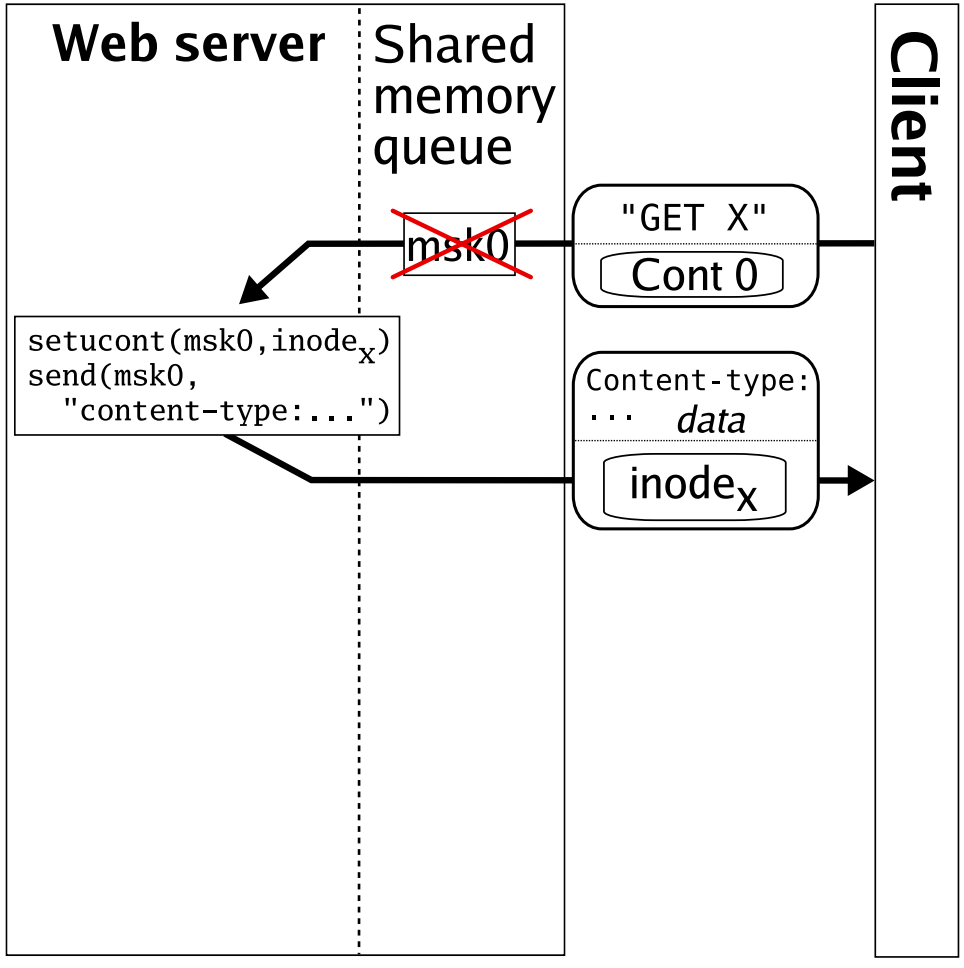
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

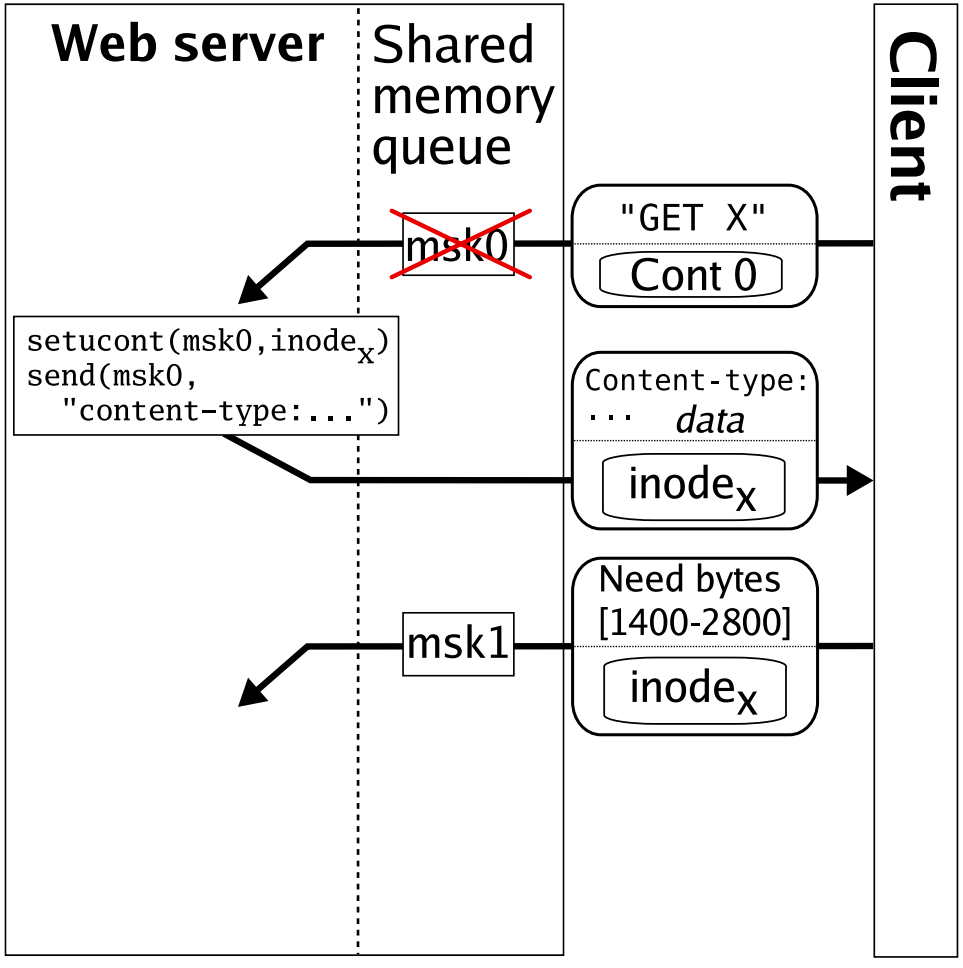
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

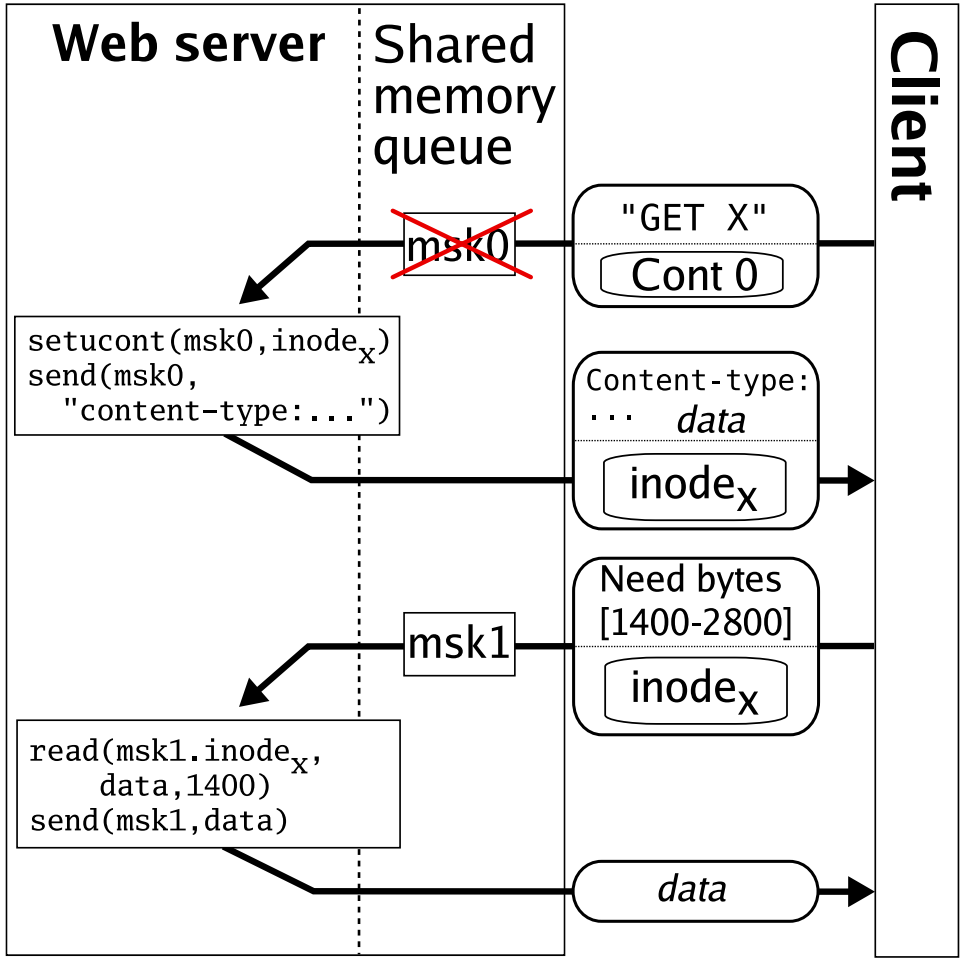
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

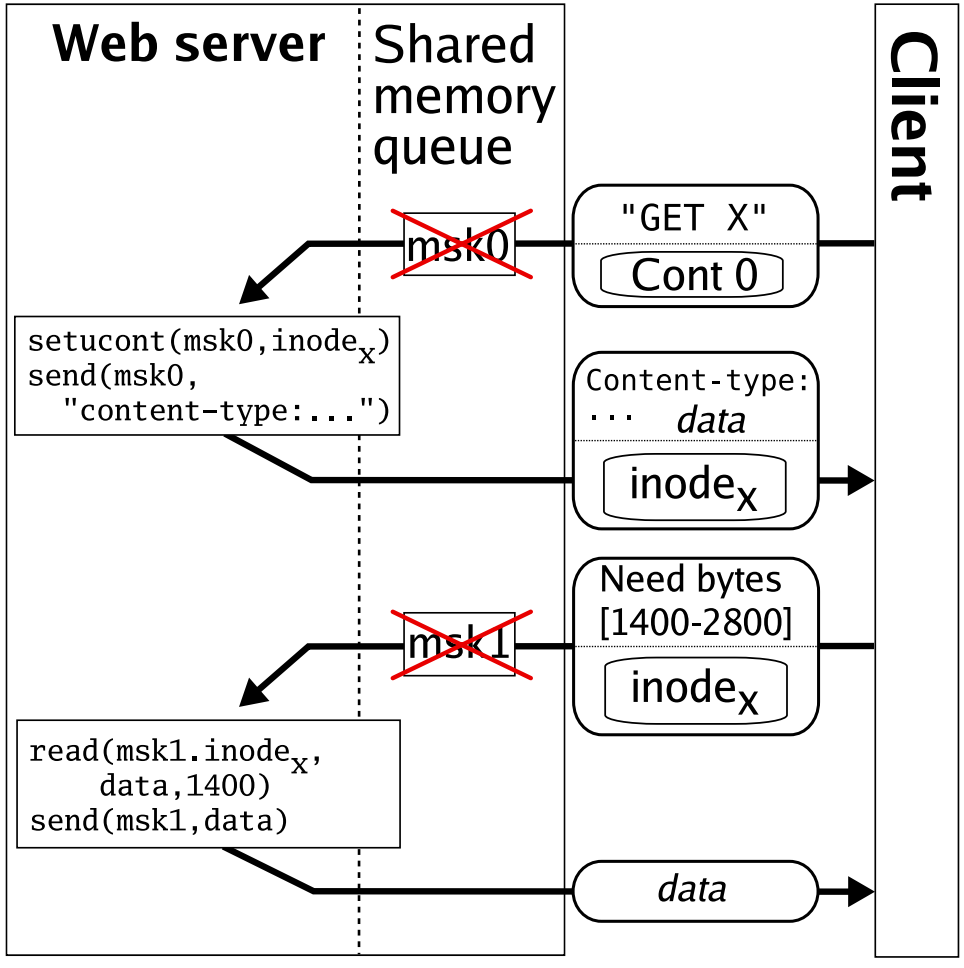
- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Trickles server API

Stateless, event-based server API to replace sockets

- Packets generate events
- Each event is a minisocket
- Minisockets provide application with:
  - identifier for client endpoint
  - user continuation
  - congestion control status



Implementation

- Linux implementation
  - No per-connection state on server
  - Deployed on PlanetLab
- Comparable performance to TCP
- Backwards-compatible
  - Same wire format
  - TCP-friendly
  - Same client API

Example applications

- Endpoint services
  - Web server
  - Watermark server
- Network services
  - Failover
  - Load balancing
  - Anycast

Evaluation

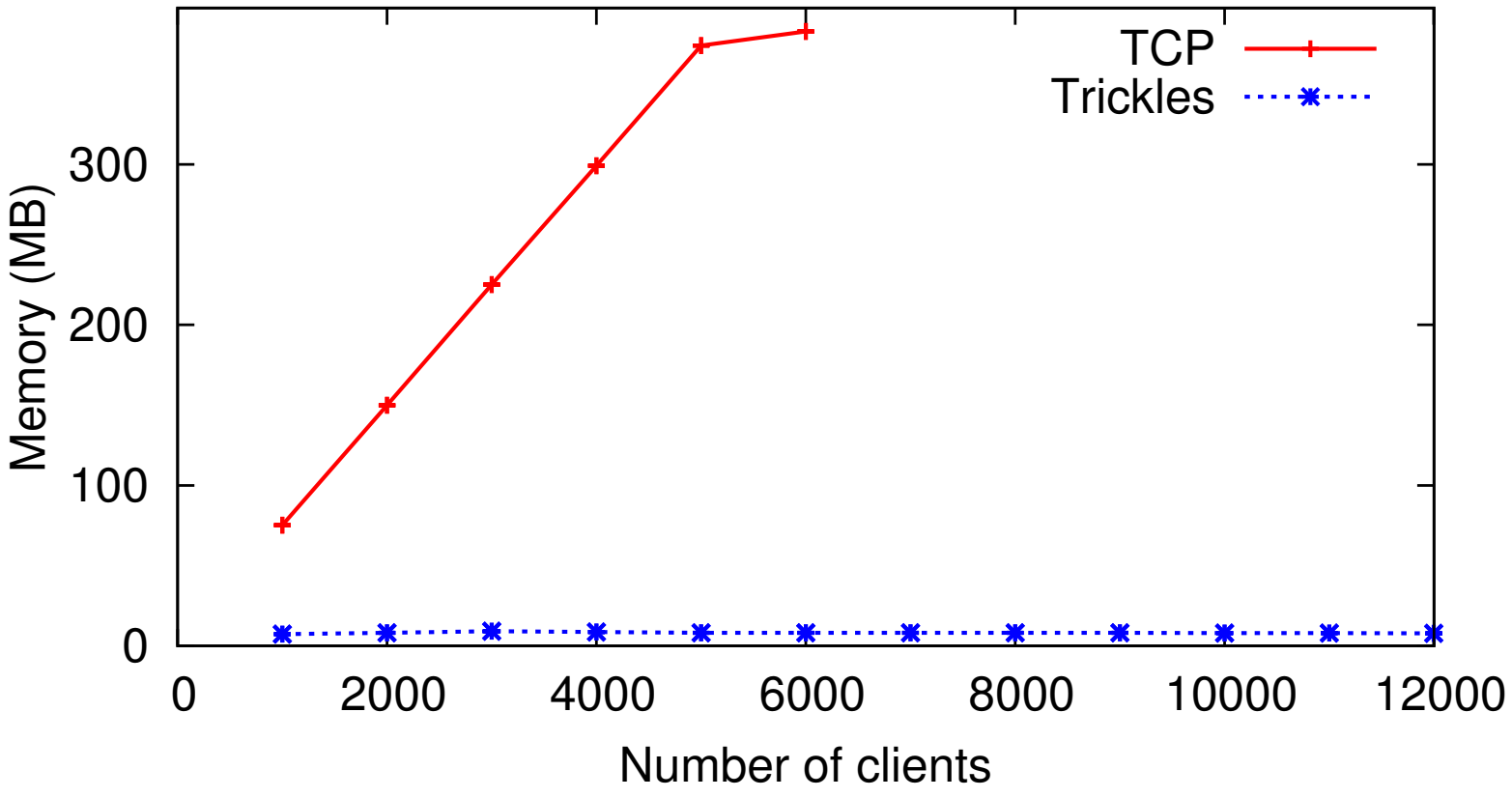
Evaluation

- LAN microbenchmarks

Improved scalability	Memory utilization
Comparable performance	Network performance comparison
TCP-friendly	Interaction with TCP
Reasonable processing overhead	Gigabit CPU utilization

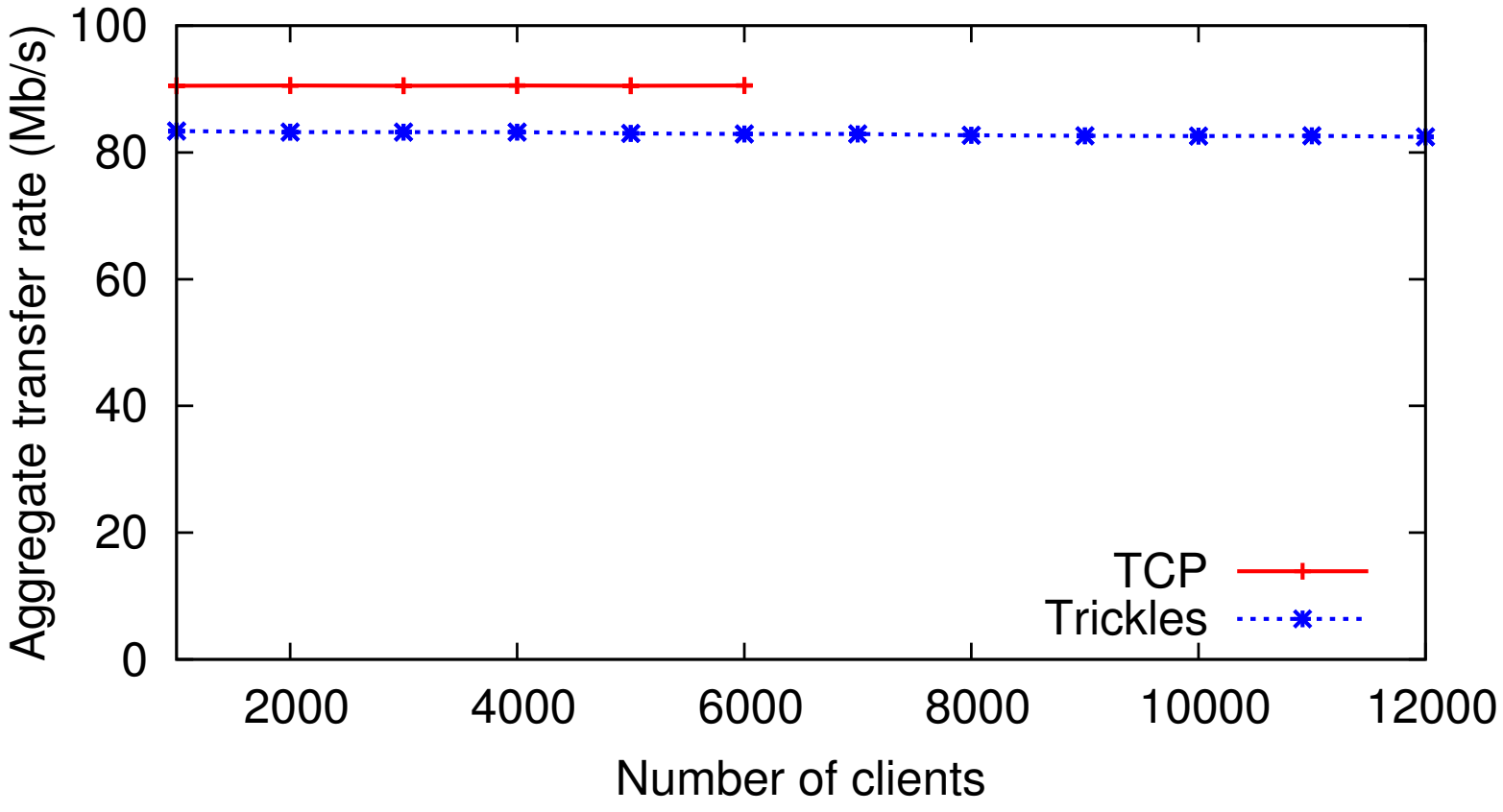
- Network redirection services
  - Transparent failover
  - PlanetLab throughput
  - Fine-grained load balancing
- Optimizations

Server-side memory utilization



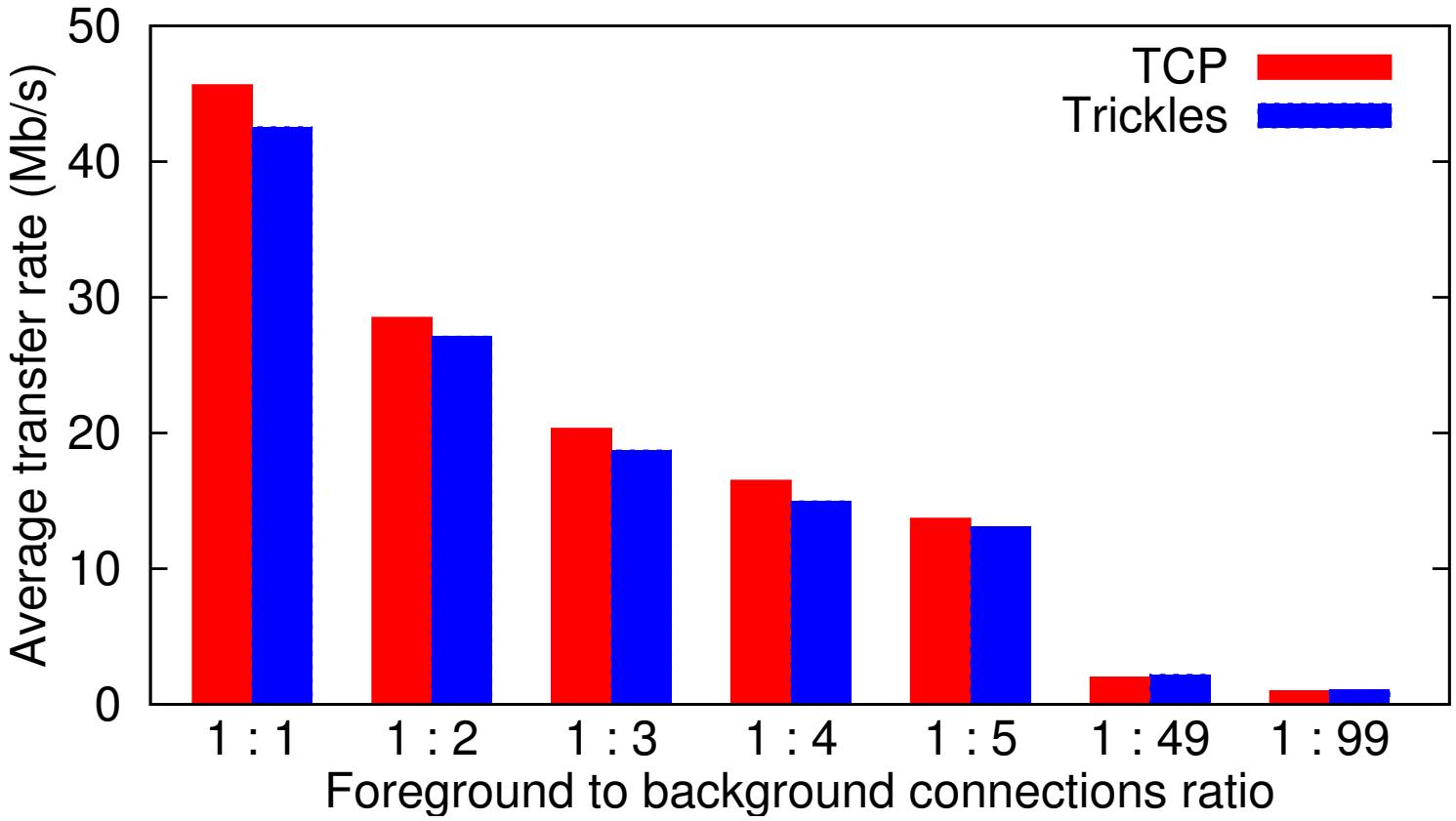
Trickles memory overhead is considerably lower, reducing vulnerability to DoS.

Aggregate throughput



Trickles achieves roughly comparable performance to TCP

Interaction of Trickles and TCP



Trickles competes fairly with TCP

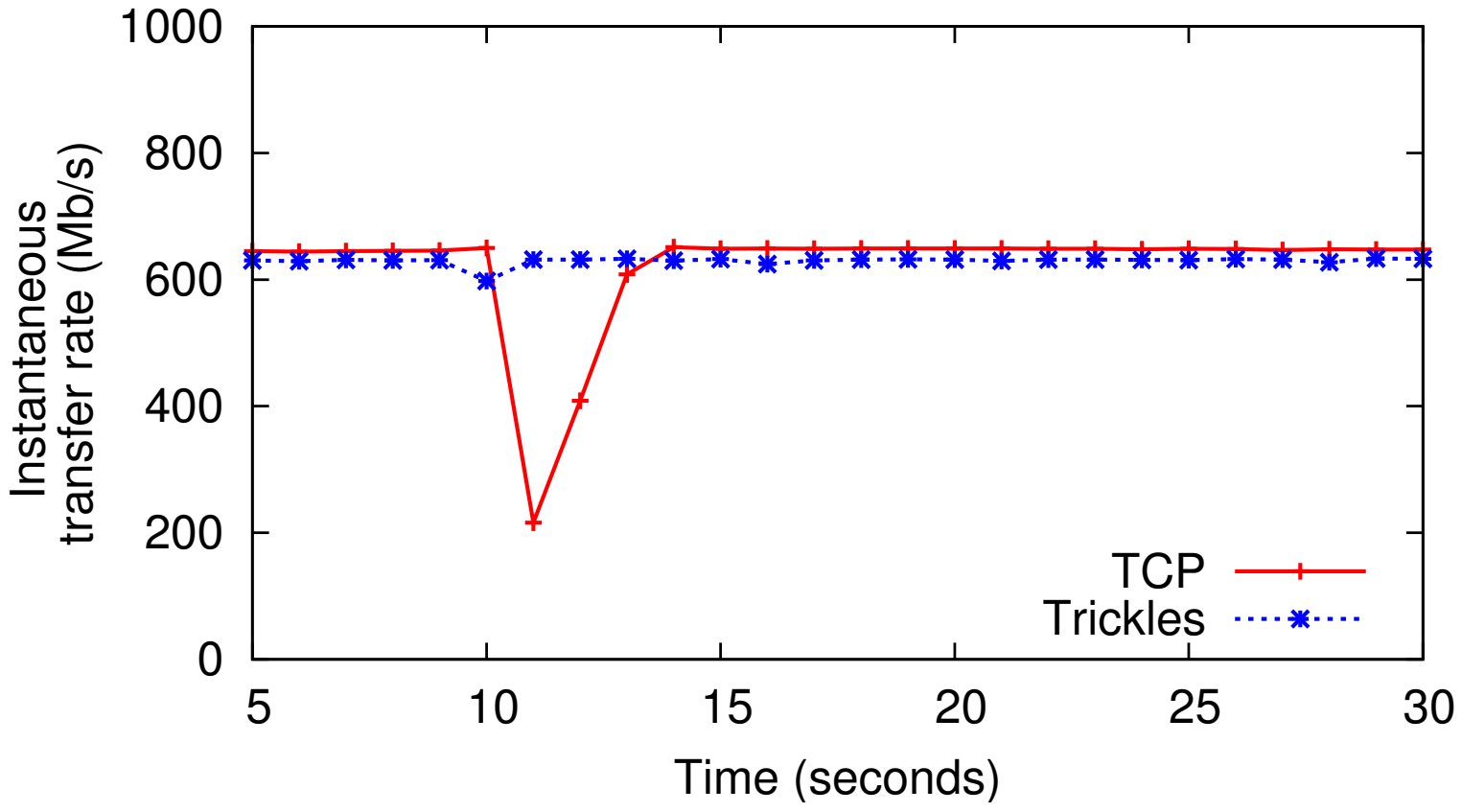
Evaluation

- LAN microbenchmarks

Improved scalability	Memory utilization
Comparable performance	Network performance comparison
TCP-friendly	Interaction with TCP
Reasonable processing overhead	Gigabit CPU utilization

- **Network redirection services**
  - Transparent failover
  - PlanetLab throughput
  - Fine-grained load balancing
- Optimizations

Transparent failover



Trickles recovers more quickly than TCP

Optimizations

- State caching
- Delta-encoding of continuations
- SKIP
- Parallel requests

Summary

- Trickles enables stateless connection-oriented services
  - Scale well
  - Resist DoS attacks
  - Require less resources
- Flexible network redirection makes possible qualitatively different kinds of services
  - Packet-level failover
  - Fine-grained load balancing
  - Geographically distributed anycast