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

SHIEH, A., MYERS, A. C., AND SIRER, E. G. 2005.

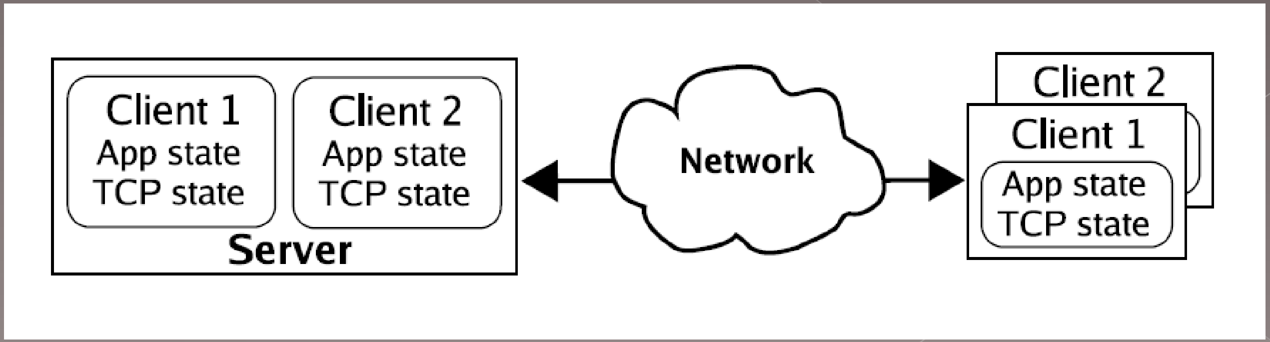
Irene K. Haque
EECS 345 – Winter 2010
Northwestern University

Overview and Goals

I. Introduction
II. Stateless Transport Protocol
III. Trickles Server API
IV. Client-Side Processing
V. Implementation & Evaluation
VI. Conclusion & Discussion

Introduction: Traditional TCP Approach

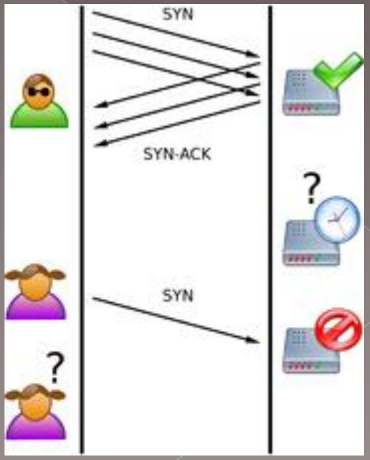
- Enables two systems to establish a connection and exchange streams of data
- Guarantees accurate delivery of data in sequential order
- Both systems hold per-connection state



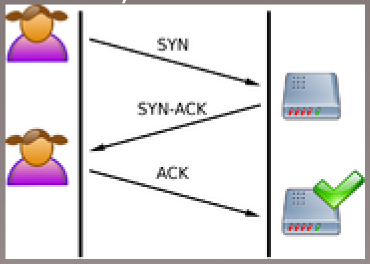
Introduction: Traditional TCP Approach

- Problems with State
  - State must be reconstructed if disconnected
    - Connection failover and recover is difficult and non-transparent
  - Per-client resources, thus limits scalability
  - Vulnerable to DoS attacks

SYN Flood

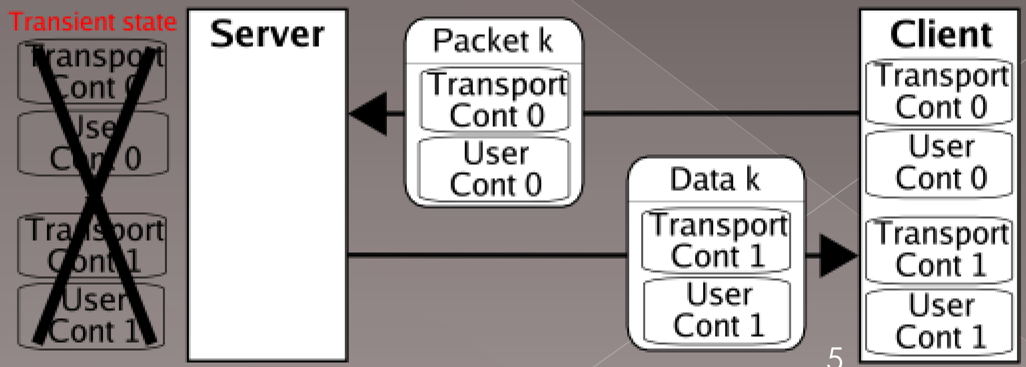


3 way handshake



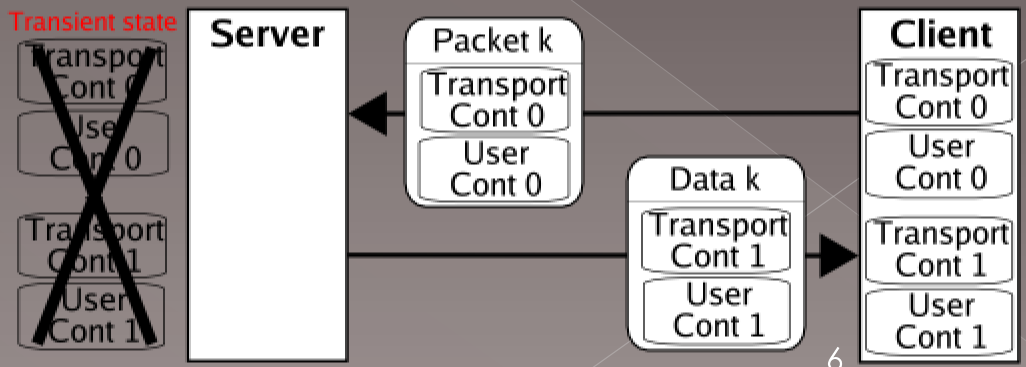
Introduction: Trickles Approach

- Make one end stateless (server)
- State kept on other end (client)
- Encapsulated state (continuations) are pushed from the server to client
- Client includes the continuation with subsequent requests



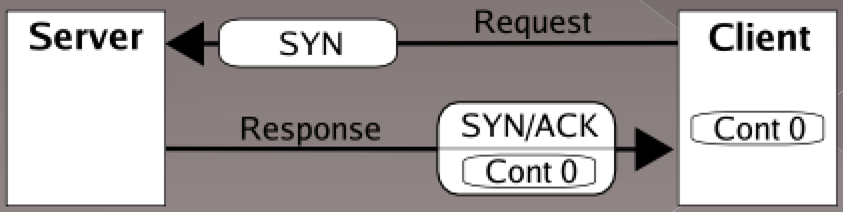
Introduction: Trickles Approach

- Continuations
  - Encapsulate server-side state
  - Piggyback on request and data packets
  - Secured with tamper-resilient MAC
  - Enables any server replica to handle the request



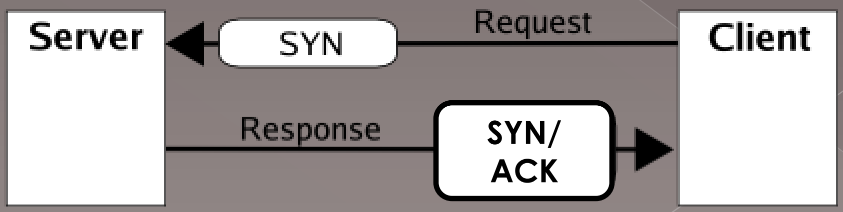
Introduction: Advantages of Trickles

- Effective utilization of resources
  - Improved Scalability
  - Resistant to Denial of Service attacks
- Services are easily replicated to other Servers
- Backwards-Compatible with TCP



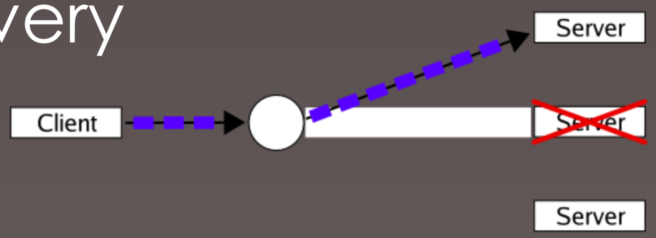
Introduction: Advantages of Trickles

- Effective utilization of resources
  - Improved Scalability
  - Resistant to Denial of Service attacks
- Services are easily replicated to other Servers
- Backwards-Compatible with TCP

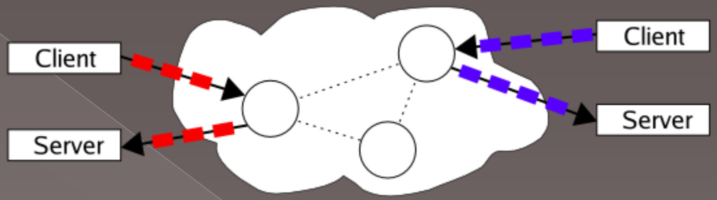


Introduction: Advantages of Trickles

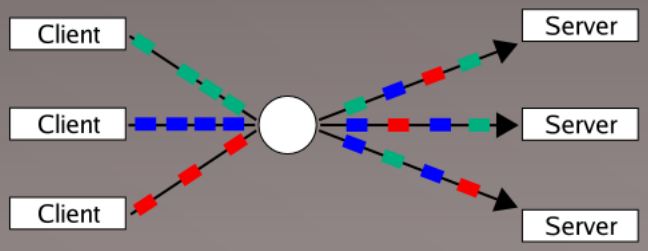
- Transparent failure recovery

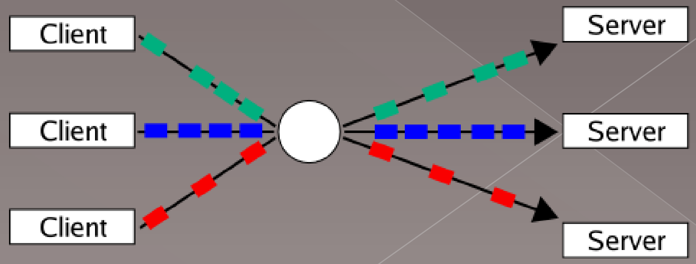


- Geographic anycast



- Packet-Level Load Balancing (Trickles vs. TCP)



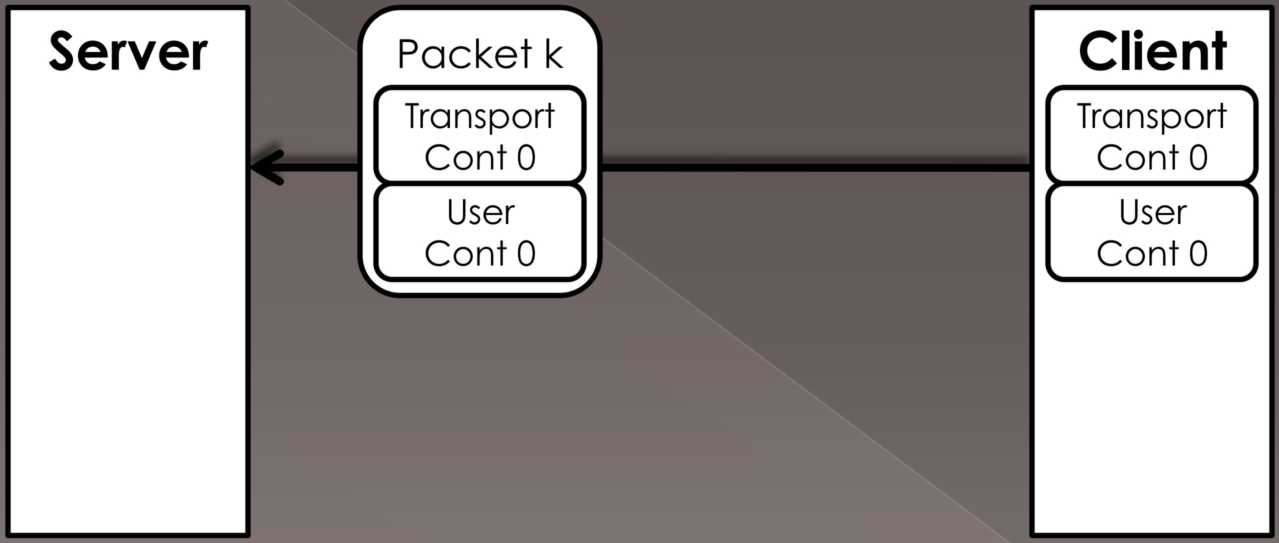


Stateless Transport Protocol: Continuations

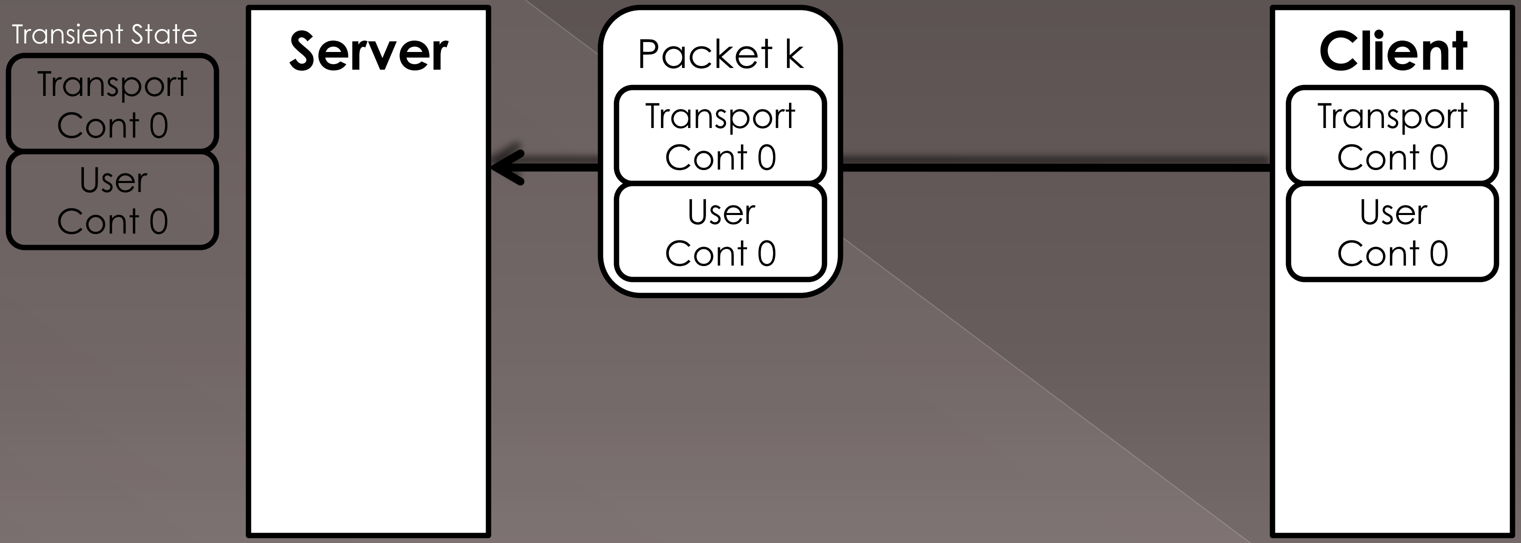
- Provides the necessary information for the server to resume processing
- Client Maintained
- Transport Continuations – Kernel Level/ TCP Congestion Control
- User Continuations – Application Level Data

Stateless Transport Protocol: Continuations

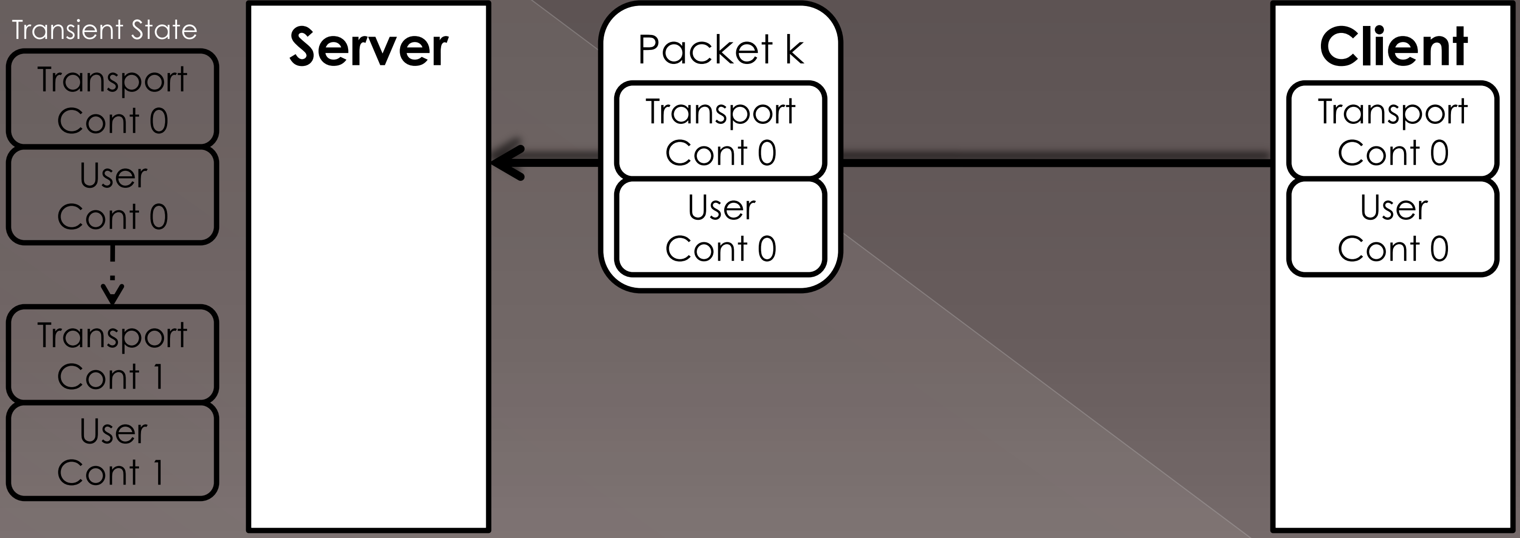
Stateless Transport Protocol: Continuations



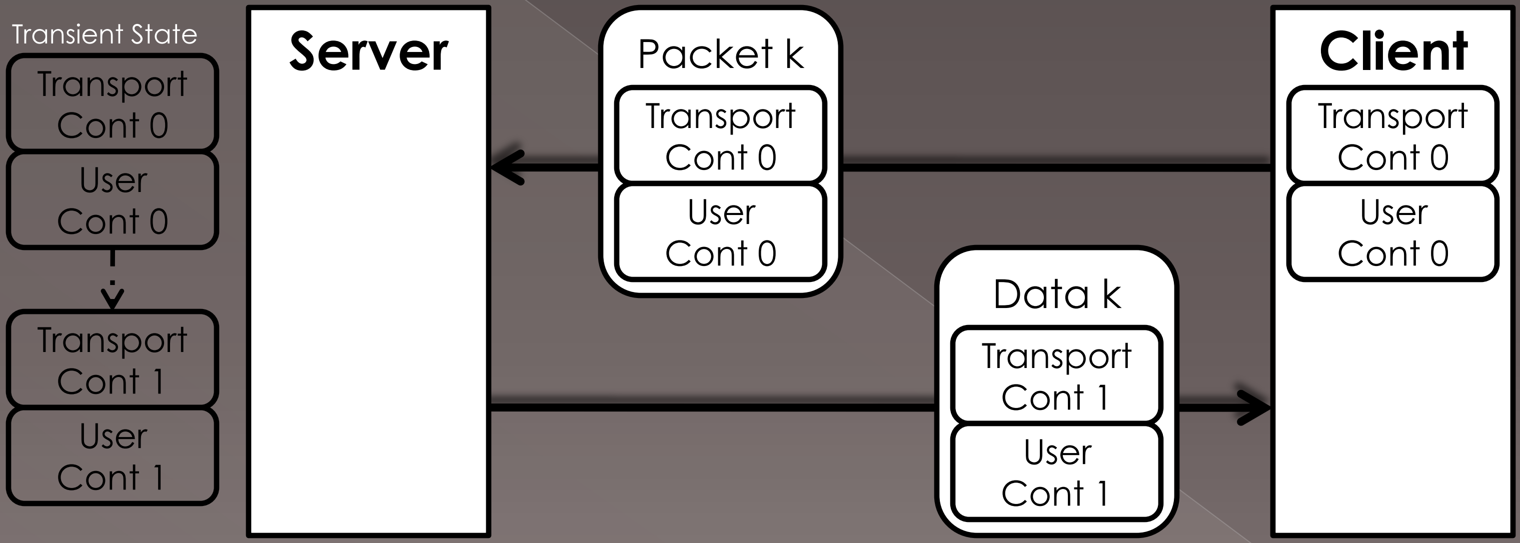
Stateless Transport Protocol: Continuations



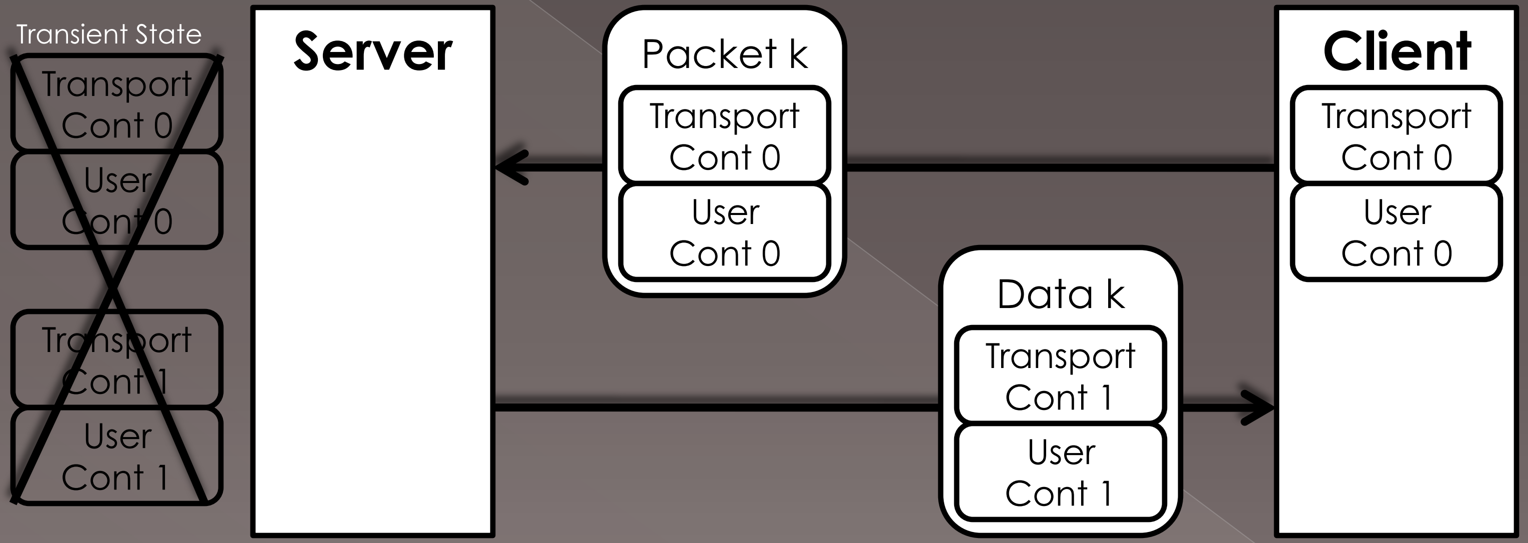
Stateless Transport Protocol: Continuations



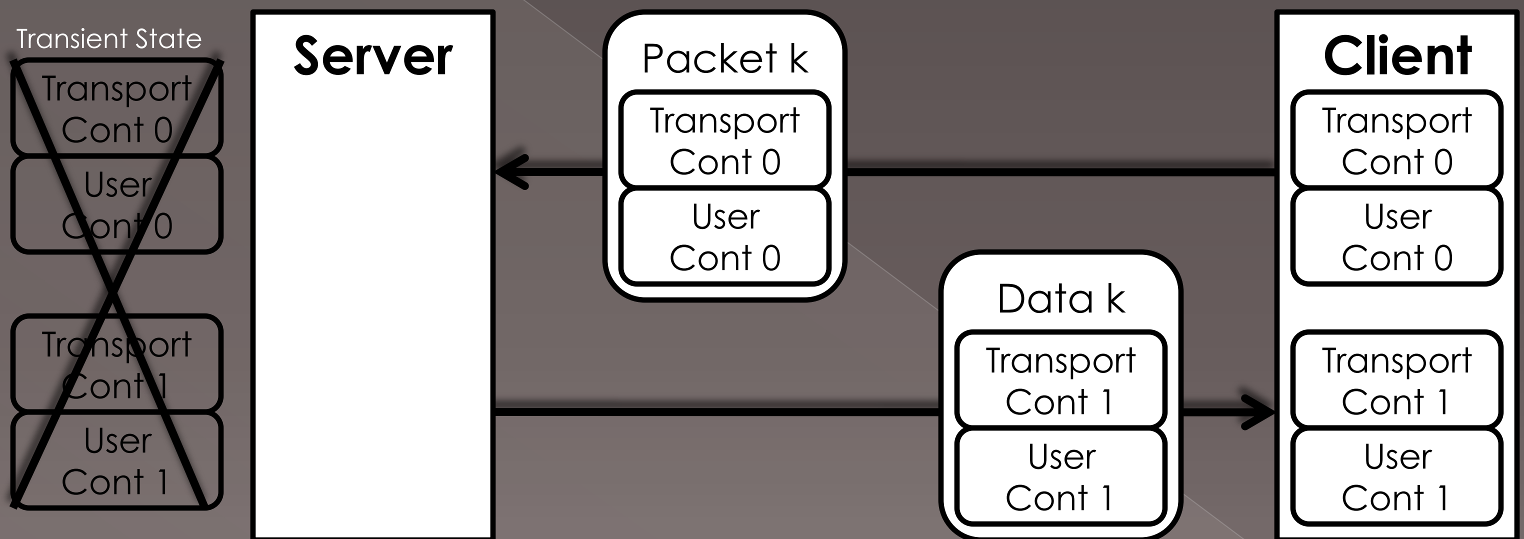
Stateless Transport Protocol: Continuations



Stateless Transport Protocol: Continuations



Stateless Transport Protocol: Continuations

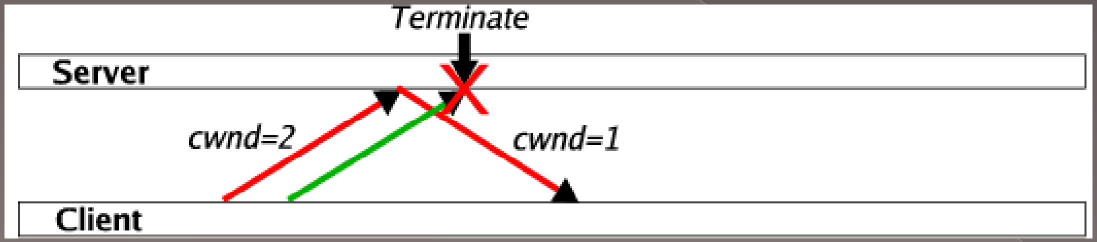


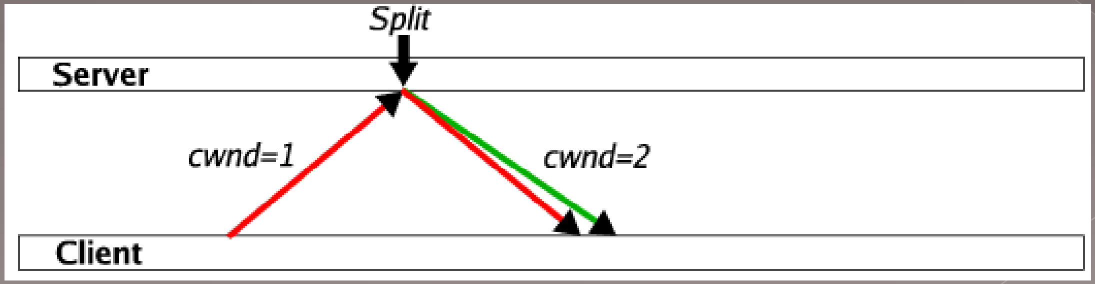
Stateless Transport Protocol: Security

- Maintaining state integrity
  - MAC prevents tampering with protected state in transport continuations
  - Range of unique nonces attached to each packet used to compute SACK proofs
- Protection against Replay
  - Requires some state, but independent of the number of connections
  - Hash table keyed on transport MAC

Stateless Transport Protocol: Trickle Abstraction

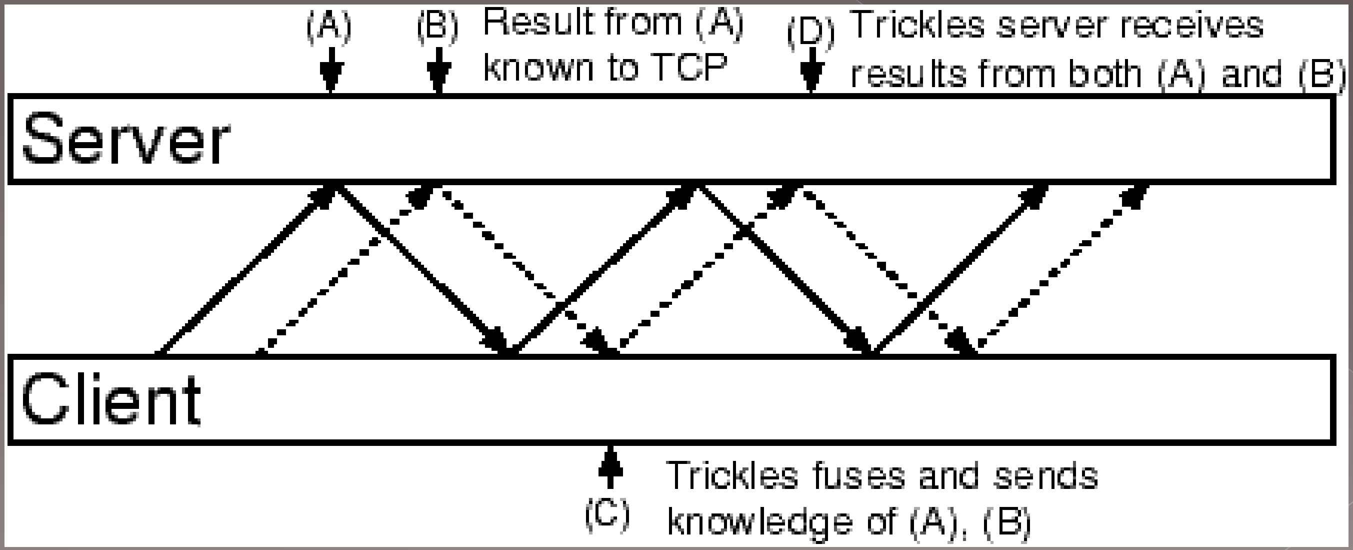
- A sequence of requests and responses
- Congestion control determines when to split and terminate by calculating current window size





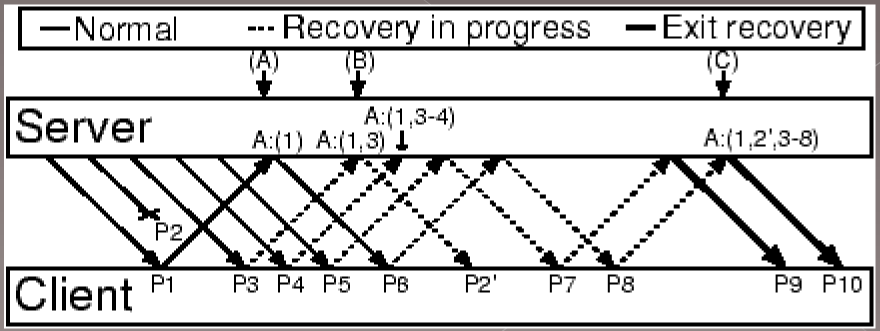
Stateless Transport Protocol: Dataflow Constraints

- Round-trip delay in state updates
- Prefix Property – given SACK proof L, proof L' sent after contains prefix L



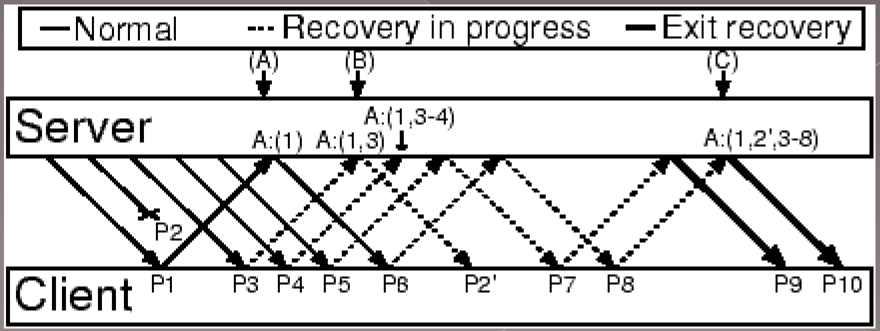
Stateless Transport Protocol: Congestion Control

- Emulates TCP Reno cwnd – 3 modes
  1. Slow Start/Congestion Avoidance
     - When cwnd is increased, the trickle is split
       - Slow Start: increase on every packet
       - Congestion Avoidance: increase every cwnd packets



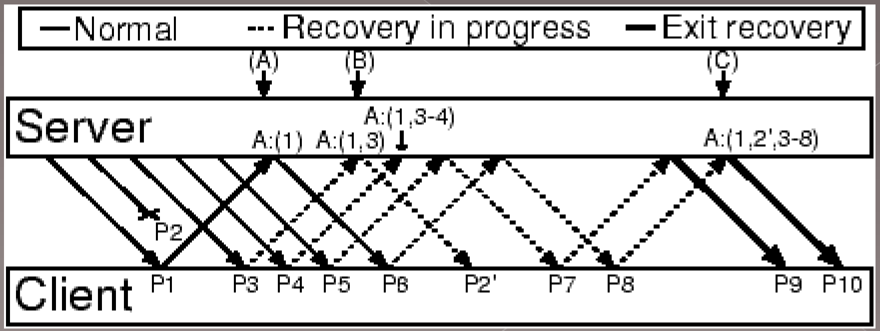
Stateless Transport Protocol: Congestion Control

- Emulates TCP Reno cwnd – 3 modes
  2. Fast Retransmit/Recovery
  - Entered when SACK contains loss
    - Retransmits lost packet
    - cwnd is halved and terminates trickles
  - When finished, enter congestion control mode



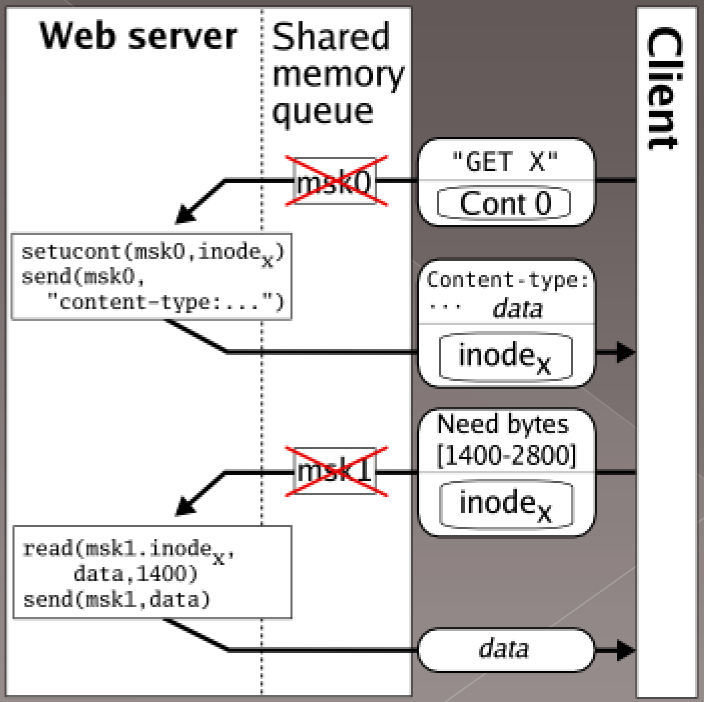
Stateless Transport Protocol: Congestion Control

- Emulates TCP Reno cwnd – 3 modes
  3. Retransmit Timeout
     - Client kernel triggers timeout
       - resets cwnd to original value
       - Sets ssthresh to half of cwnd (before first lost)
     - When finished, enter slow start mode



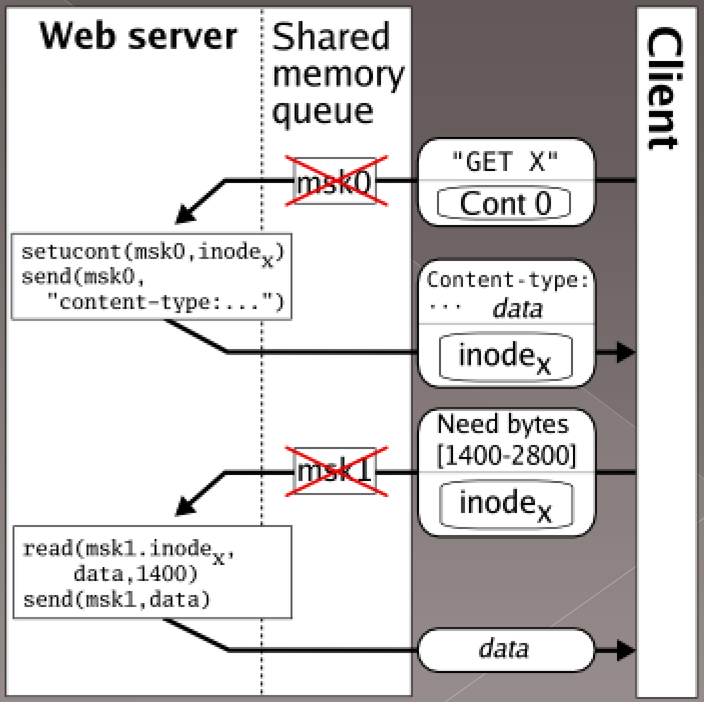
Trickles Server API: Event Queue

- Stored in shared memory
- Packets generate events/minisockets



Trickles Server API: Minisockets

- Represents the remote end-point
- Send/Receive data
- Destroyed after event is processed
- Includes user continuation and congestion control



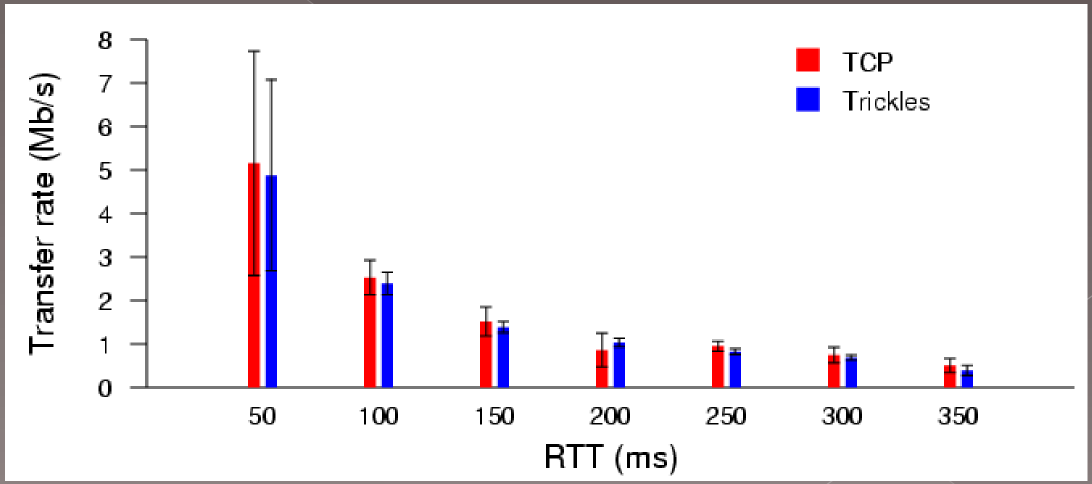
Client-Side Processing: Client Kernel

- Client application is not aware of Trickles, but uses a Berkeley sockets interface
- Kernel maintains transport protocol
  - Creates requests from transport continuations
  - SACK Proofs
  - Triggers Retransmit Timeout
- Manages user continuations
  - Input
  - Output

Implementation

- Linux
  - 15,000 lines of code
  - AES Encryption
- PlanetLab
  - Real Internet Conditions

Average throughput for a 160kB file

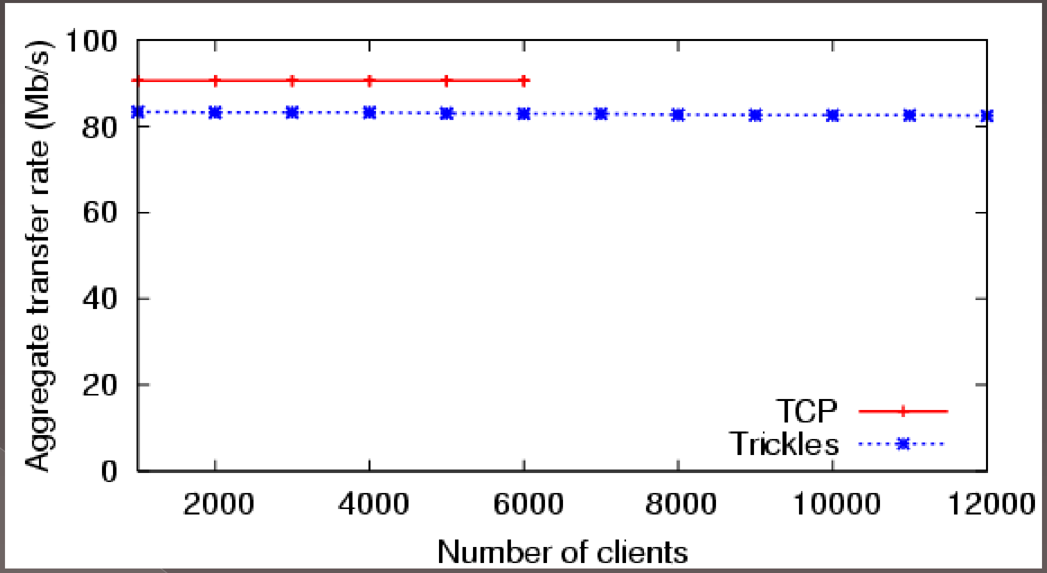


Each bar represents the average of all PlanetLab nodes that are within a 50ms bucket, sorted by latency

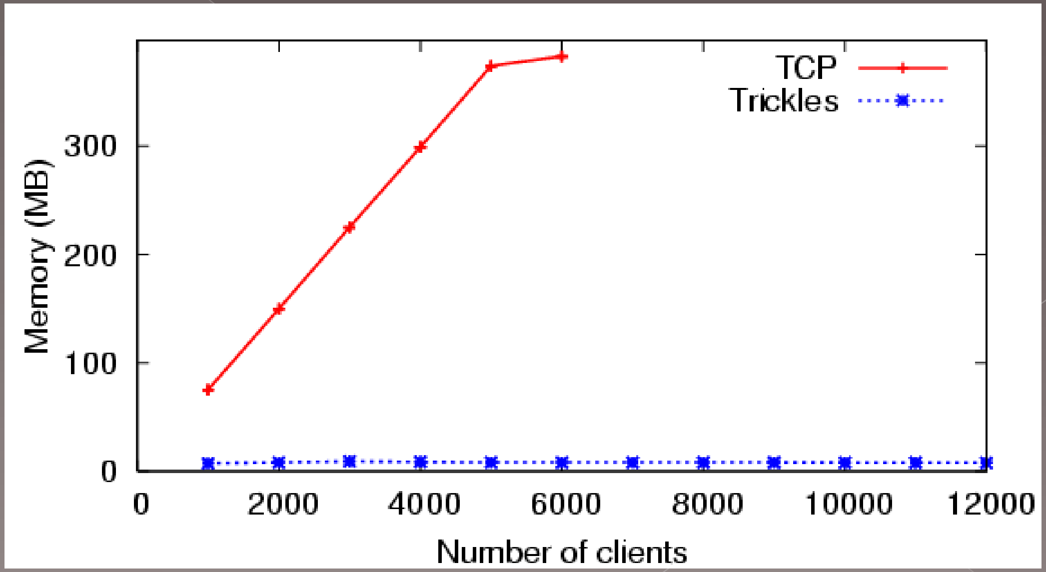
Evaluation:

- Aggregate Throughput

clients connects to a single server over a 100Mb link



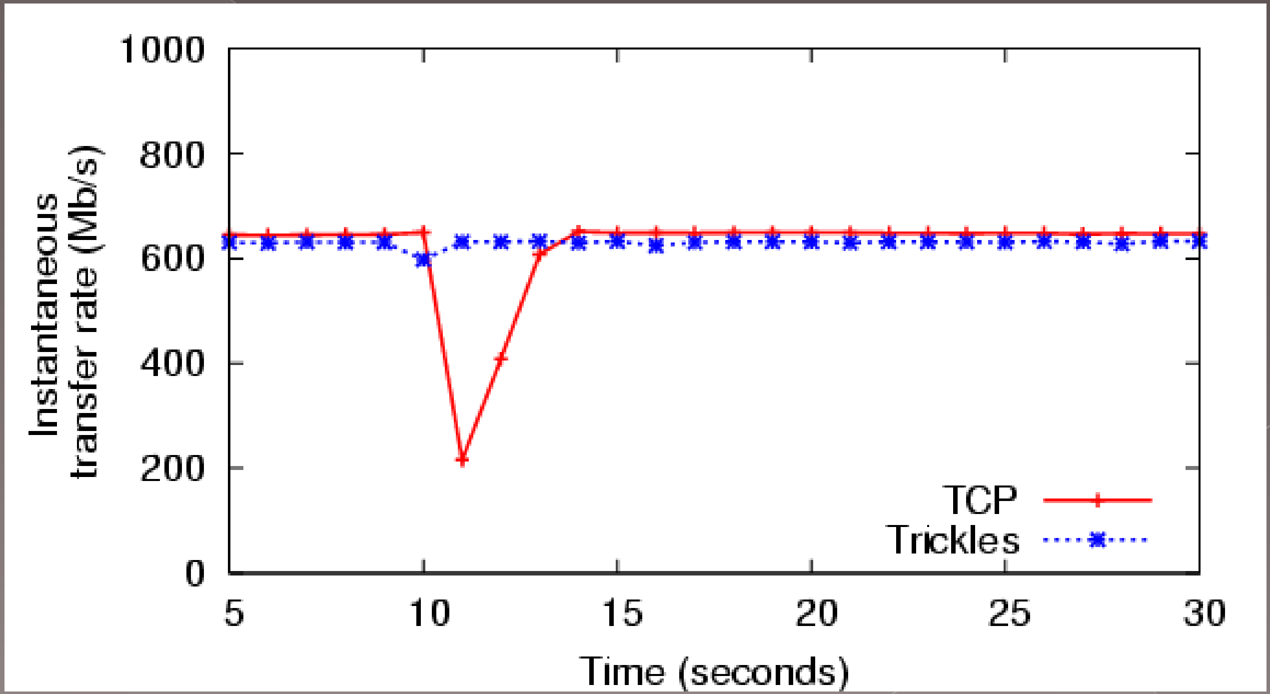
- Memory Utilization



Evaluation:

- Instantaneous Failover

Disconnection occurs at t = 10 seconds.



Conclusion

- Trickles is similar to TCP in efficiency and reliability, but with better resource allocation
- Offers packet-level load-balancing, instantaneous failover, transparent connection migration
- Servers may be replicated and geographically distributed
- Trickles is backward compatible with existing TCP clients and servers

Discussion

- Any Disadvantages?
- Overhead costs
  - Transport continuation size is 75+12m
    - (m= number of loss events)
  - TCP header size is between 20 – 60 bytes