

# Handling Persistent Connections in Overloaded Web Servers

**Thiemo Voigt<sup>a</sup>**

Swedish Institute of Computer Science (SICS),  
Uppsala University

**Per Gunningberg**

Uppsala University

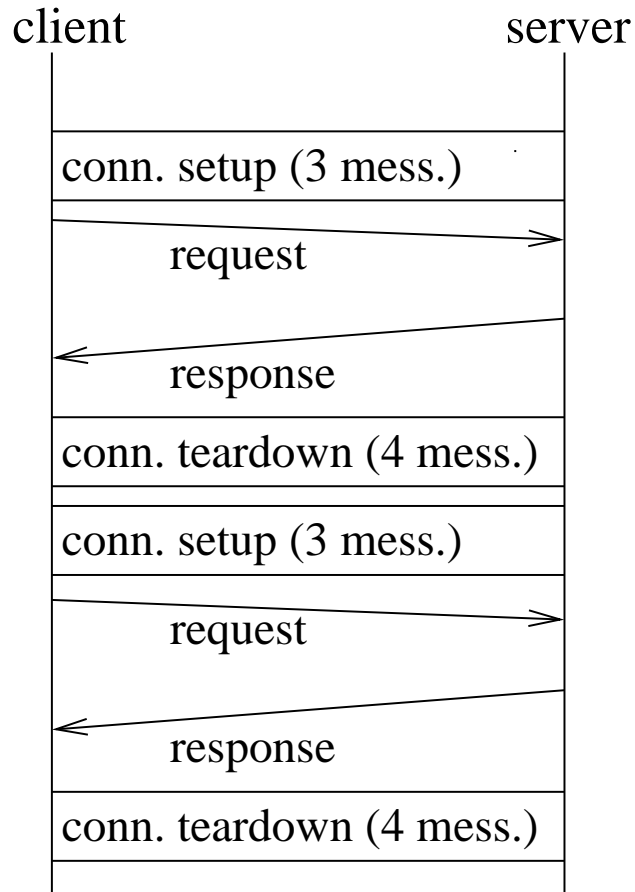
---

<sup>a</sup>This work is partially funded by the national Swedish Real-Time Systems research initiative ARTES ([www.artes.uu.se](http://www.artes.uu.se)).

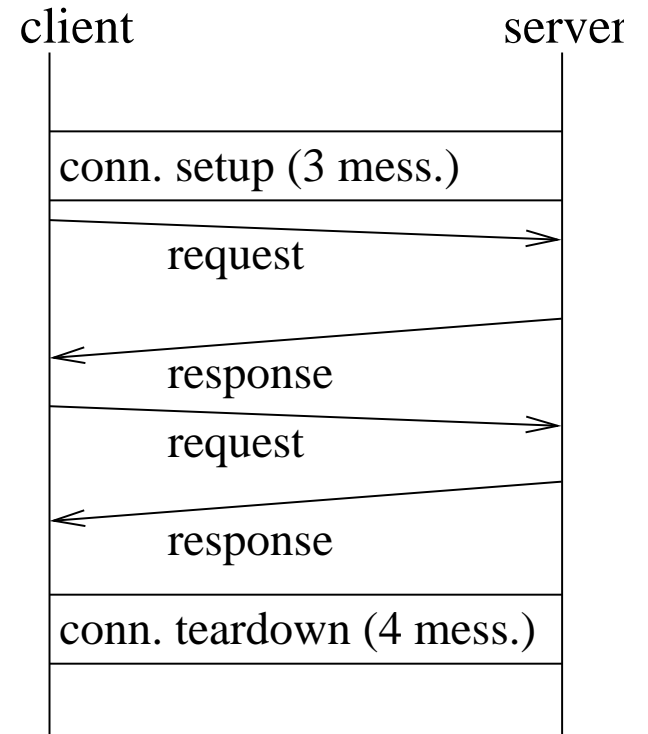
# Motivation

- web server overload can cause unpredictable service
- web server admission control and service differentiation schemes deployed
- requests accepted/rejected based on information found in HTTP header
- requests rejected when queue lengths exceed thresholds or when requests are not compliant with token bucket based policers

# Persistent Connections



(a) no persistent connections



(b) persistent connections

# Motivation

- persistent connections reduce client latency and server overhead
- resource demands of future requests on persistent connections unknown when admission decision is made
- first request does not reveal information about amount of work entering the system
- too optimistic → overload risk  
too conservative → unnecessary rejections

goal of this work:

avoid uncontrollable overload while maximizing access

# Outline

- Approach for dealing with persistent connections
- Kernel-based architecture
- Experiments to evaluate the approach
- Conclusions

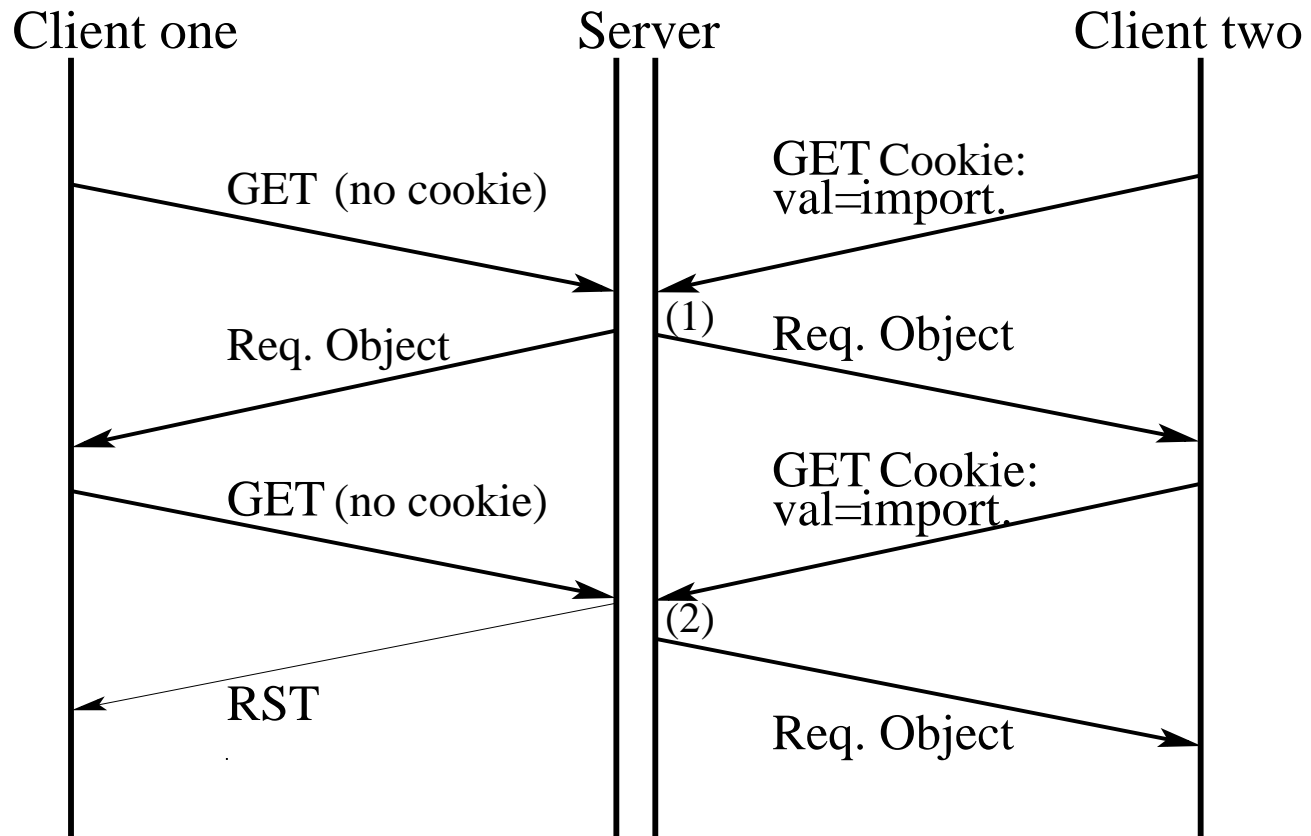
# Approach

- admission control of first request using token bucket based policers
- no control of following requests on same persistent connection
- during overload: reset “unimportant” connections, but keep alive important connections
- use cookies to code importance of connections

# Why Cookies?

- all information to determine importance of connections embedded in cookie.
- established technique, widely used
- can contain long-lasting information, e.g. identify preferred customers.
- no changes to the web server or clients (browsers) necessary.

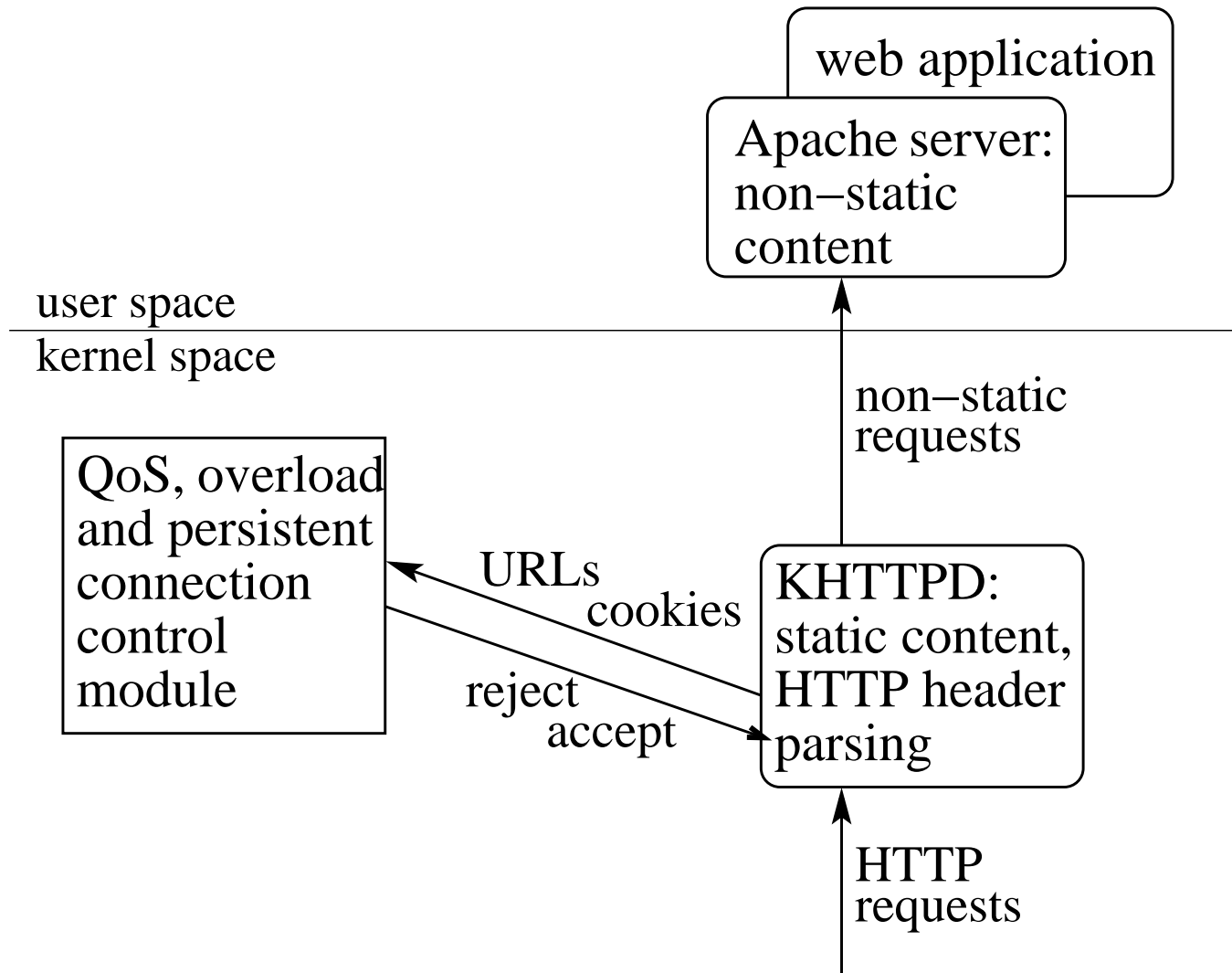
# Cookie Example



(1) normal condition

(2) overload

# Kernel-based Architecture



## Kernel-based Architecture

- cookie-based connection control active when CPU-utilization is higher than threshold
- when cookie-based connection control is active:
  - “right cookie” → keep connection alive
  - else → abort connection
- approach works as well in middleware or user-space

# Experimental Setup

- isolated network, server and hosts connected via switch
- client populations emulated using one traffic generator (S-Client) instance per client population
- client population requests one or more files on same persistent connection (“session”)
- some of the requests carry “right” cookie in HTTP headers (“protected requests”)

# Experiments: Overload Control

two emulated client populations:

- (client\_pop<sub>one</sub>)'s sessions: 6 requests (last 4 protected), 50 sessions/sec requested
- Bursty population: sessions start with same first request (treated same way as client\_pop<sub>one</sub>'s first request), plus more requests (unprotected) which causes server overload

cookie control enabled	throughput non-bursty client pop.	session compl. time
no	40.3 sess/sec	9.9 sec
yes	37.8 sess/sec	2.1 sec

Overload protection using cookie control

# Experiments: Service Differentiation

three emulated client populations:

- ( $\text{client\_pop}_{pref}$ )'s sessions: 6 requests (all reqs. protected)
- ( $\text{client\_pop}_{two}$ )'s sessions: 6 requests (last 4 reqs. protected)
- Bursty population causing overload

Throughput (sessions/sec)		
throughput	$\text{client\_pop}_{pref}$	$\text{client\_pop}_{two}$
overall	25.3	19.4
during bursts	21.1	9.4
during non-bursts	28.0	24.8

Service Differentiation using cookie-based control

# Limitations

- KHTTPD handles static requests only
- simple string comparison (*strcmp*)
- turning cookie-based control on/off can potentially lead to oscillations
- there need to be “unimportant” sessions
- users might disable cookies

# Conclusions

- problem of unknown resource demand of persistent connections
- architecture that protects web servers from overload
- importance of persistent connections encoded in cookies
- experiments that show that the approach can prevent uncontrollable server overload and provide service differentiation

# Filter Rules

URL	ACTION
*noaccess*	<drop >
/index.html	< rate=50, burst=5 >
/cgi-bin/*	< rate=10, burst=2 >

application-level filter rules

(HTTP header-based connection control)

cookie	ACTION
val=important	keep-alive
client=Fred	keep-alive

cookie rules

(cookie-based persistent connection control)

## Related Work

- *WebQoS* [1999]: middleware for service differentiation. Cookies used to identify sessions.
- Bhatti *et al.* [2000] use cookies to code durations of sessions
- Cherkasova and Phaal [1999] deploy admission control with aim to allow session completion
- Aron *et al.* [1999] support persistent connections in cluster-based web servers
- Almeida *et al.* [1999]: preferred requests executed as higher priority processes