Approaches to Support Differentiated Quality of Web Services

Ryu Sook Hyun

Email : shryu@postech.ac.kr

DP&NM Lab.

POSTECH

Contents

Introduction (1)

- The WWW is rapidly increasing, with the number of users expected to reach 320 million by the year 2002.
- The diversity of Web applications continues to increase.
  → The quality of service is an increasingly critical issue in Web services.
- QoS requirements
  - Refer to non-functional requirements, such as performance or availability requirements.
  - Deal with how an application or service will behave at run-time.

  → May differ for different invocations of a service, based on factors such as the user or time of delay.

  → This is referred to as a differentiated QoS.

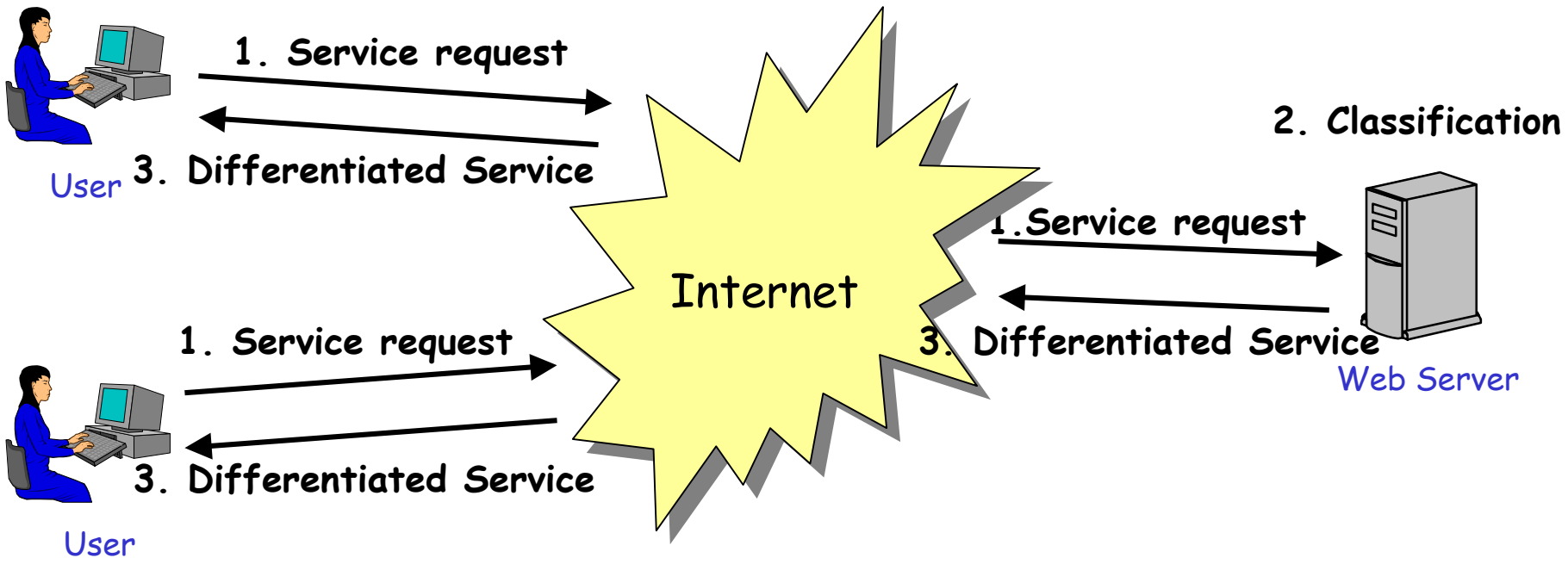
Introduction (2)

- Most Web Servers handle incoming requests on a first-come, first-serve basis.

  →do not provide differentiated QoS.

- How can Web Servers support differentiated QoS?

Differentiated Service Outline



General Web Server & Differentiated Web Server

- General Web Server
  - Handles incoming requests on a first-come, first-serve basis
  - Premium users cannot be protected from overload in server
  - Do not provide Differentiated QoS
- Differentiated Web Server
  - The incoming requests are classified into different categories
  - Different levels of service are applied to each category
  - The requests of premium users are serviced first
  - Provide Differentiated QoS

Related Work

	My work	WebQOS (HP)	Nikolaos's work	Jussara's work
User-Level Approach	O	O	O	O
Kernel-Level Approach	O	X	X	O
Portability	O	O	X	X
# of Differentiation Levels	2	3	2	2
URL (Classification)	O	O	O	O
Client IP (Classification)	O	O	O	X
User Private Key (Classification)	O	X	X	X
User Authentication (Classification)	O	X	X	X

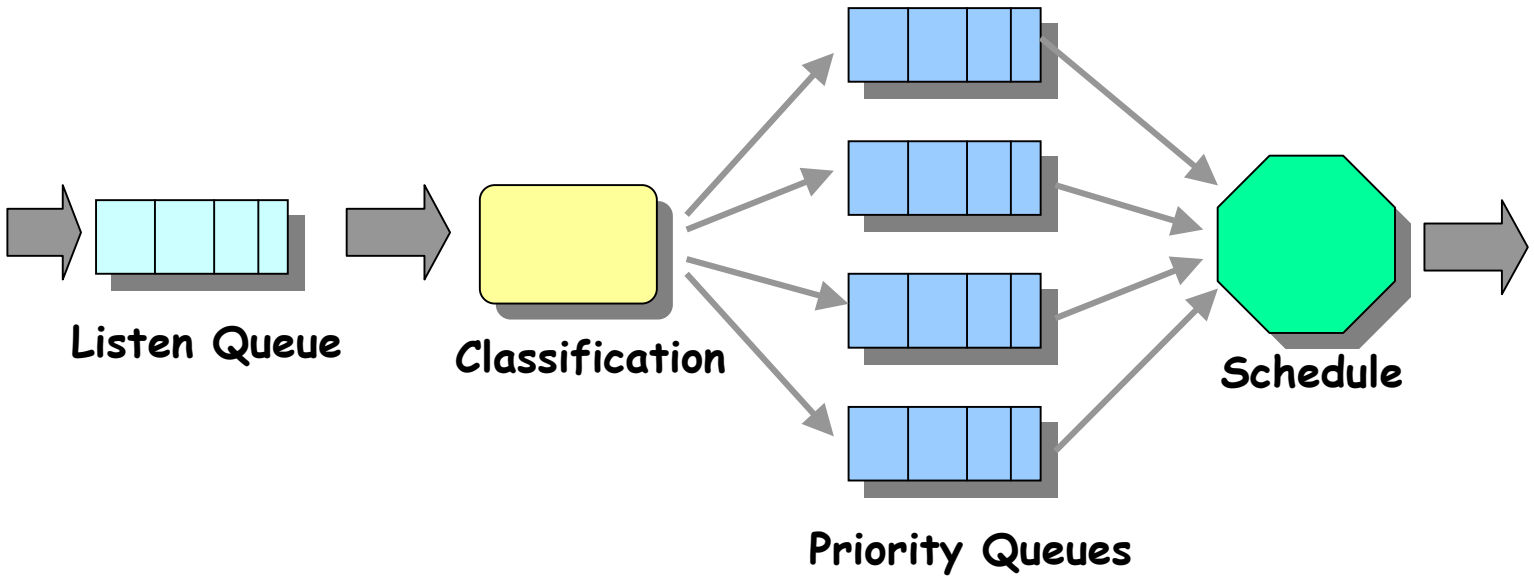
Functional Requirements

- Classification of Requests
  - The classification methods are as various as can be thought of
- Scheduling
  - Throughput remains constant during peak demand
  - Error rate remains constant during peak demand

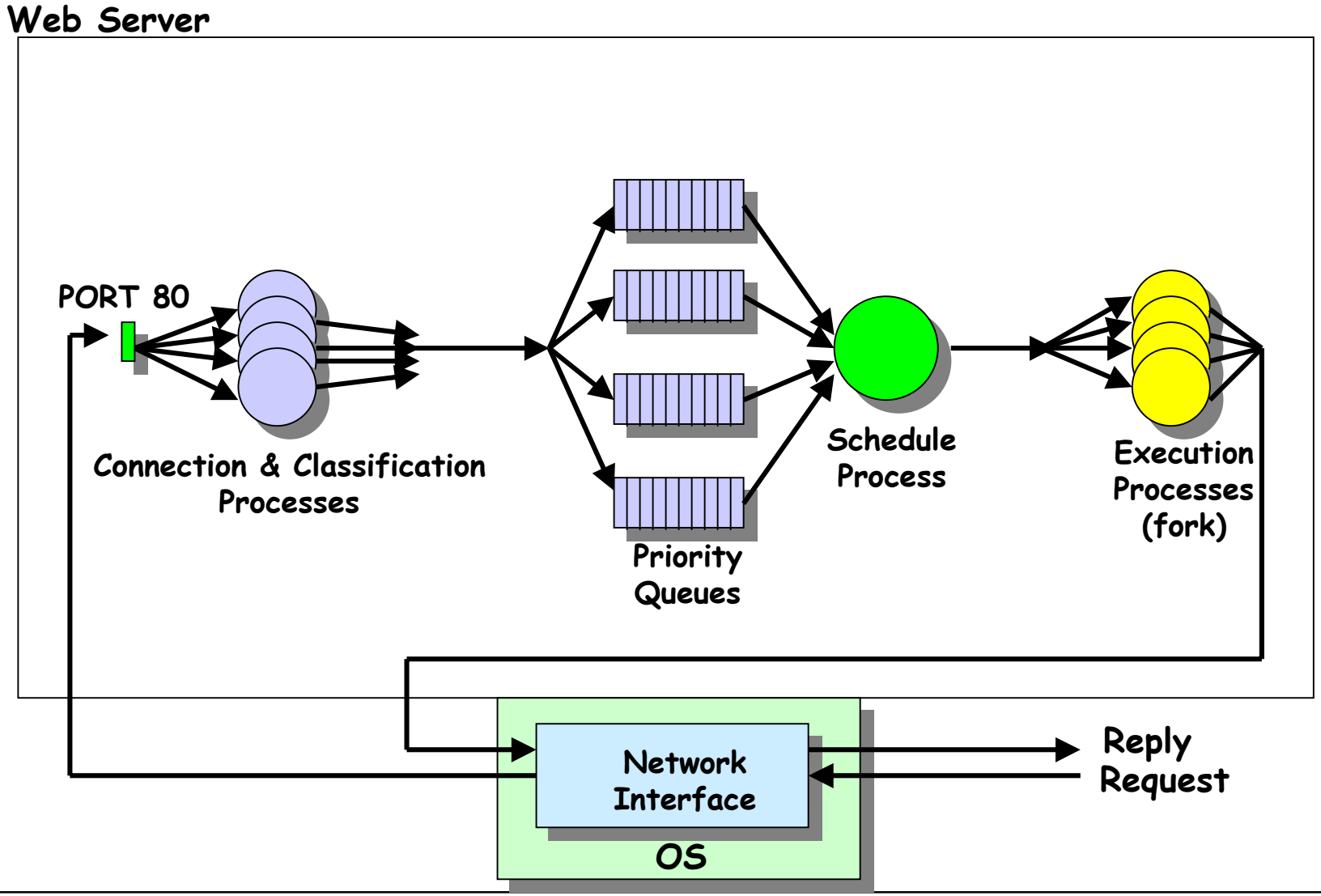
Non-Functional Requirements

- Portability
  - Portable on various Web servers and operating systems
- Resource Requirements
  - Must use as little CPU, memory usage as possible

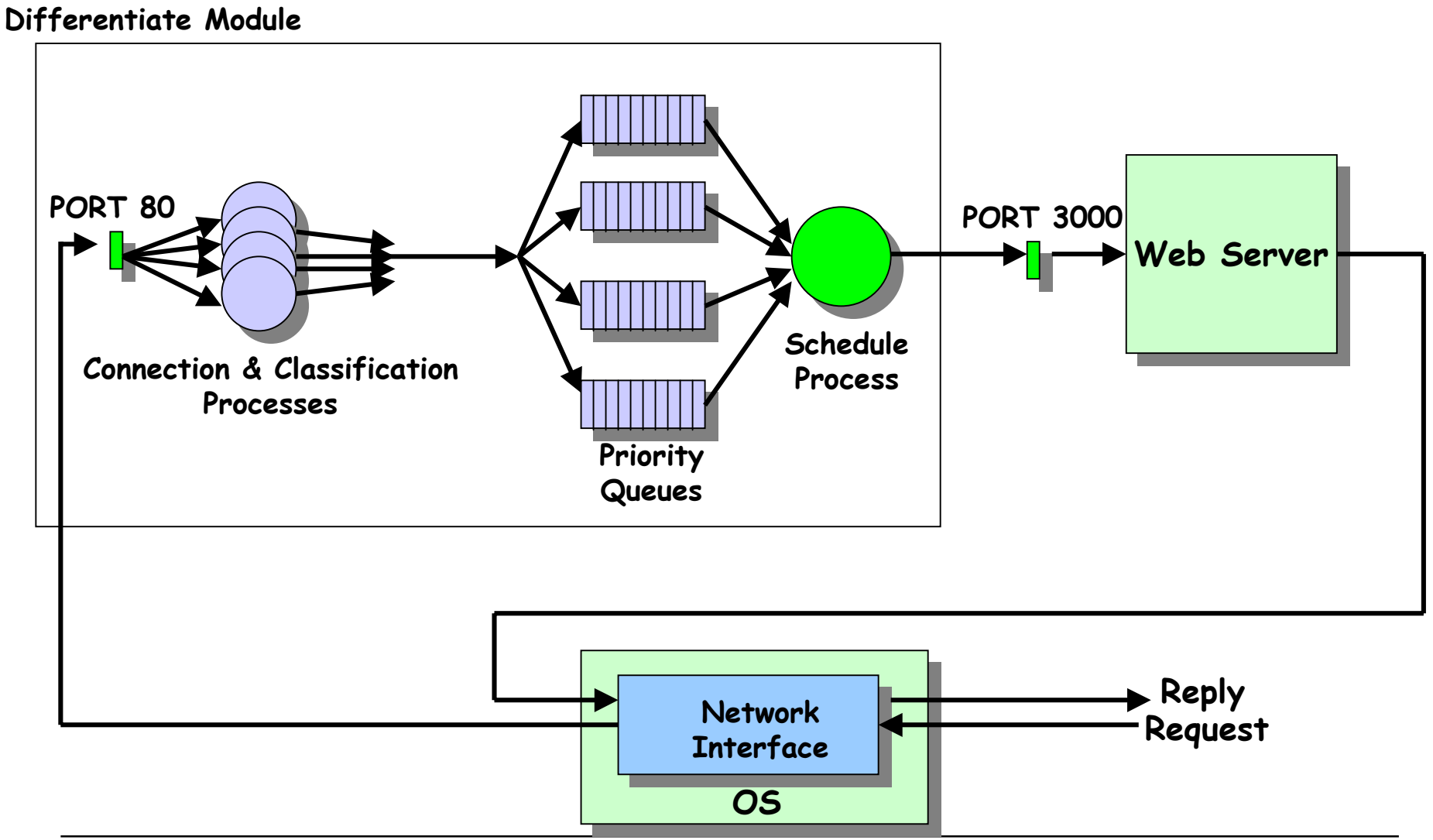
Differentiated Service System Architecture



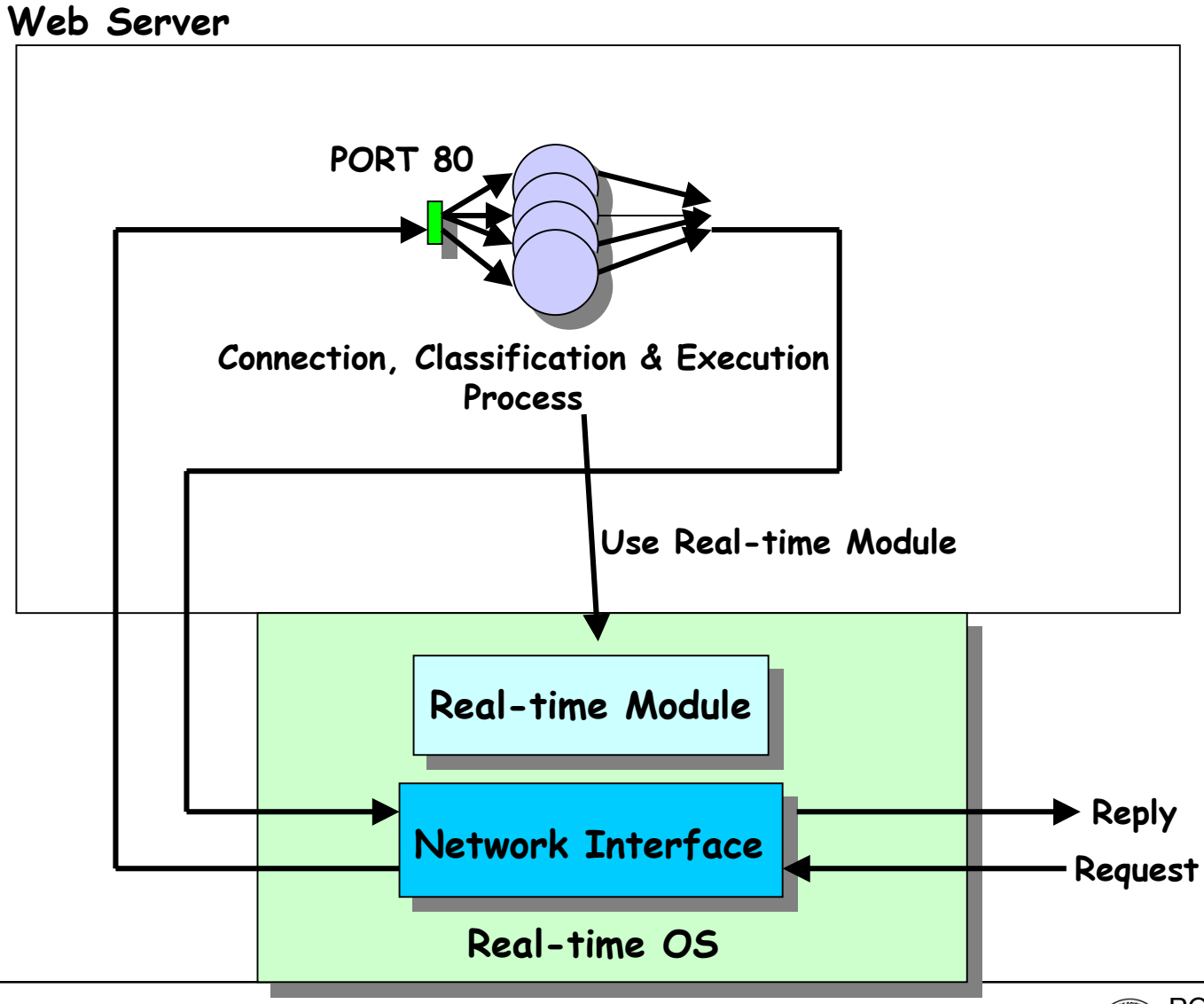
User-Level Approach (1)



User-Level Approach (2)

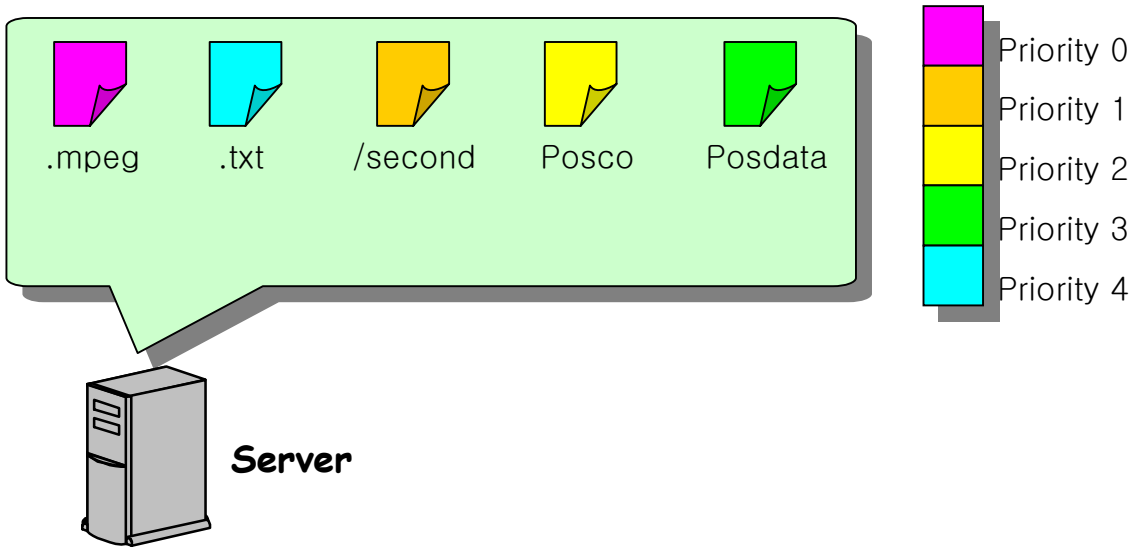


Kernel-Level Approach



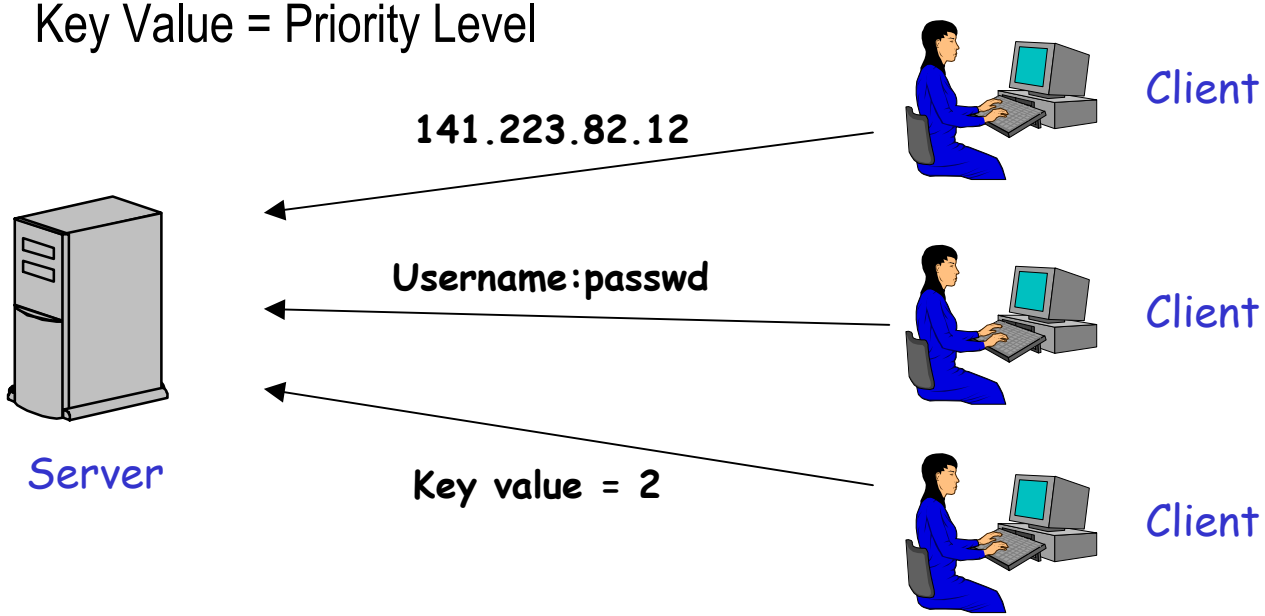
Classification Approaches (1)

- Server-based approach
  - Use content information in Server
    - URL



Classification Approaches (2)

- Client-based approach
  - Use Client Information
    - Client IP
    - User authentication
    - User Private Key
    - Key Value = Priority Level



Priority-driven Scheduling methods in User-Level Approach

- Strict priority
- Weighted priority
- Shared capacity
- Fixed capacity
- Earliest deadline first

→ We use strict priority scheduling method

→ Strict priority scheduling method is very simple and requires less CPU capacity, memory volume than other priority scheduling methods

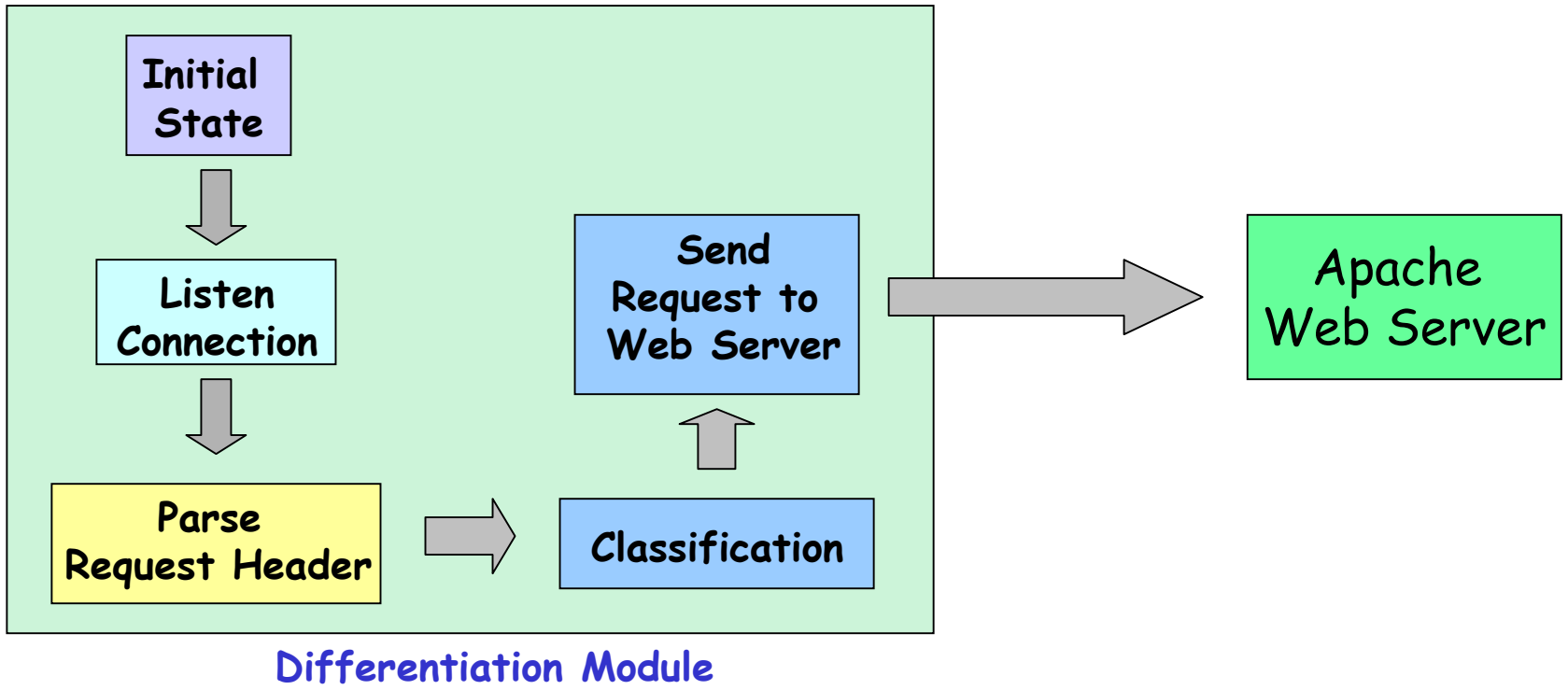
Development Environment

	User-Level Approach	Kernel-Level Approach
OS	Linux Kernel 2.2.14	Linux Kernel 2.2.14
Realtime Kernel	None	Soft Realtime Kernel (Monstavista Realtime Scheduler)
Web Server	Apache 1.3.12	Apache 1.3.12
Language	C, PHP	C, PHP

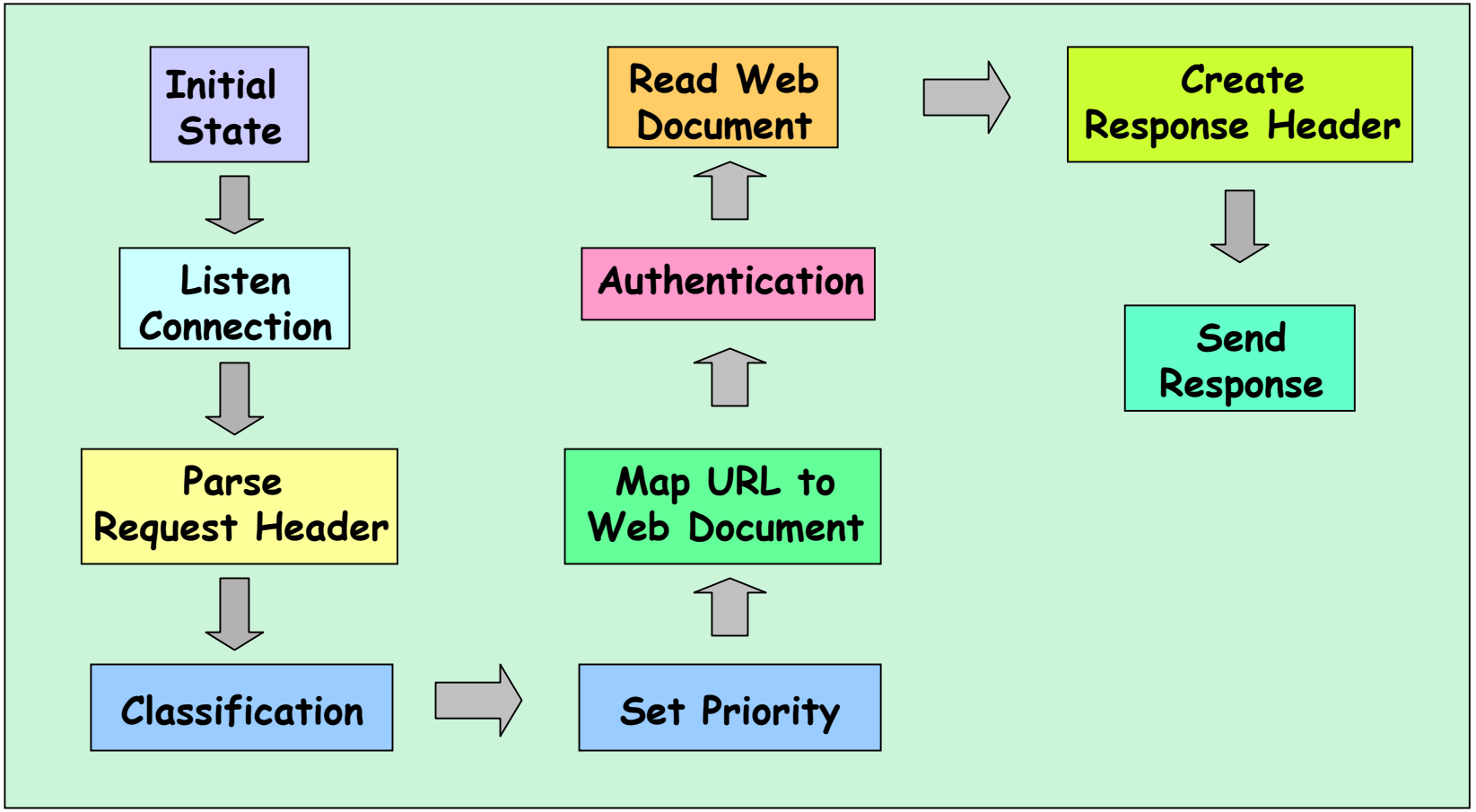
Montavista Realtime Scheduler

- Soft Realtime Kernel Scheduler
- Priority driven Real-time process scheduler
- Priority Levels : 128
- Multilevel-FIFO, Multilevel-Round Robin scheduling method
- Separation between Real-time and Non-Real-time Processes

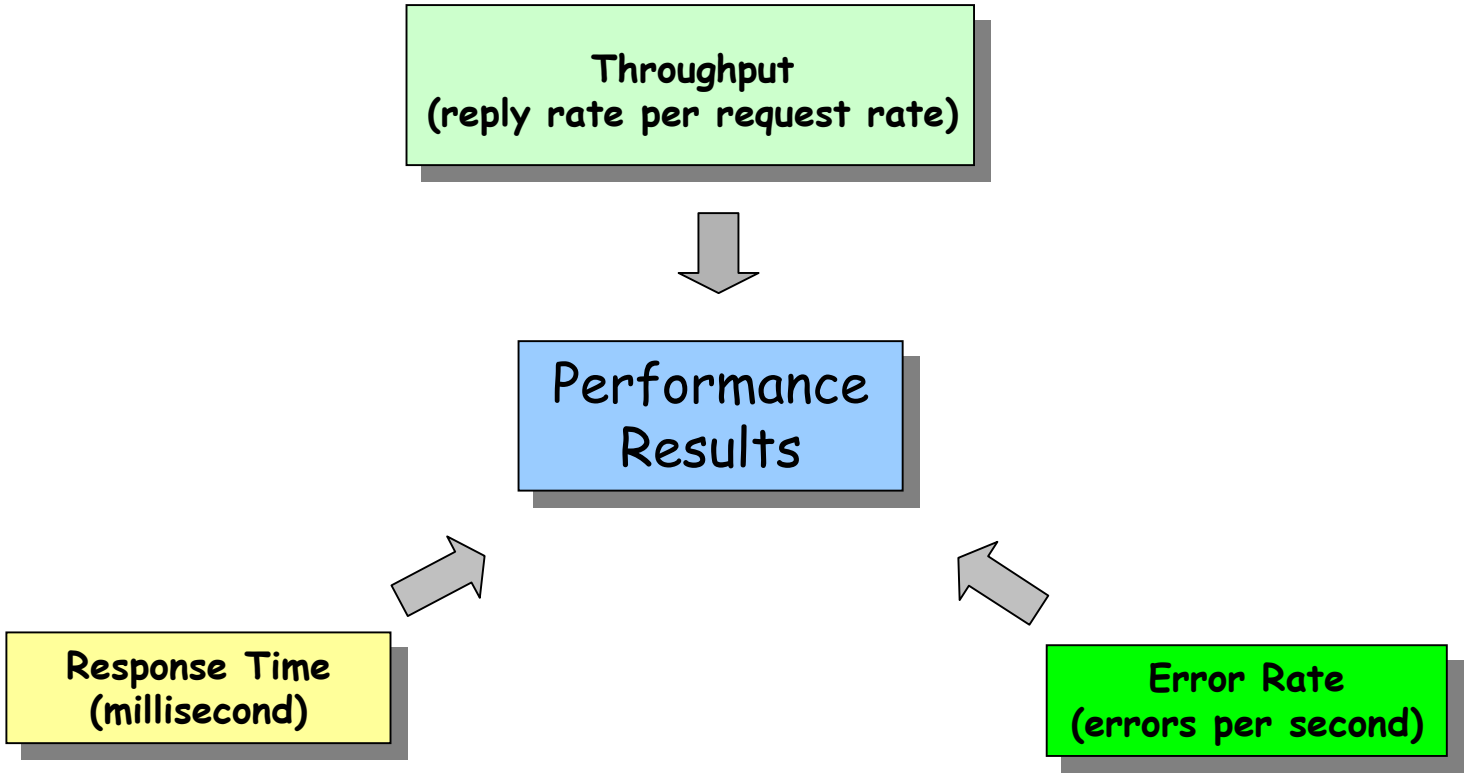
User-Level Approach Flow



Kernel-Level Approach Flow

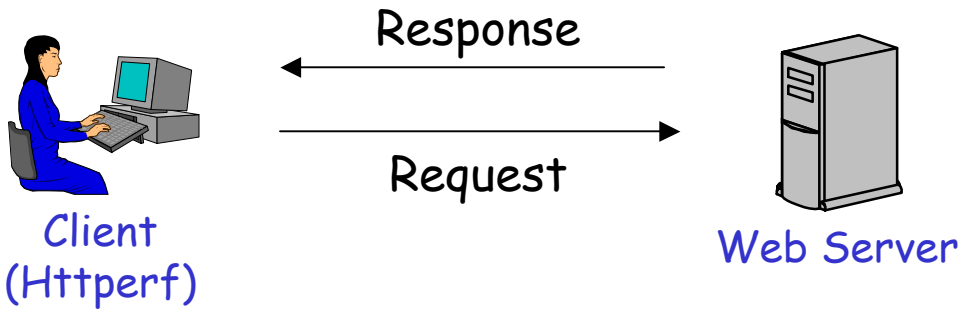


Performance Metrics



Httperf Example

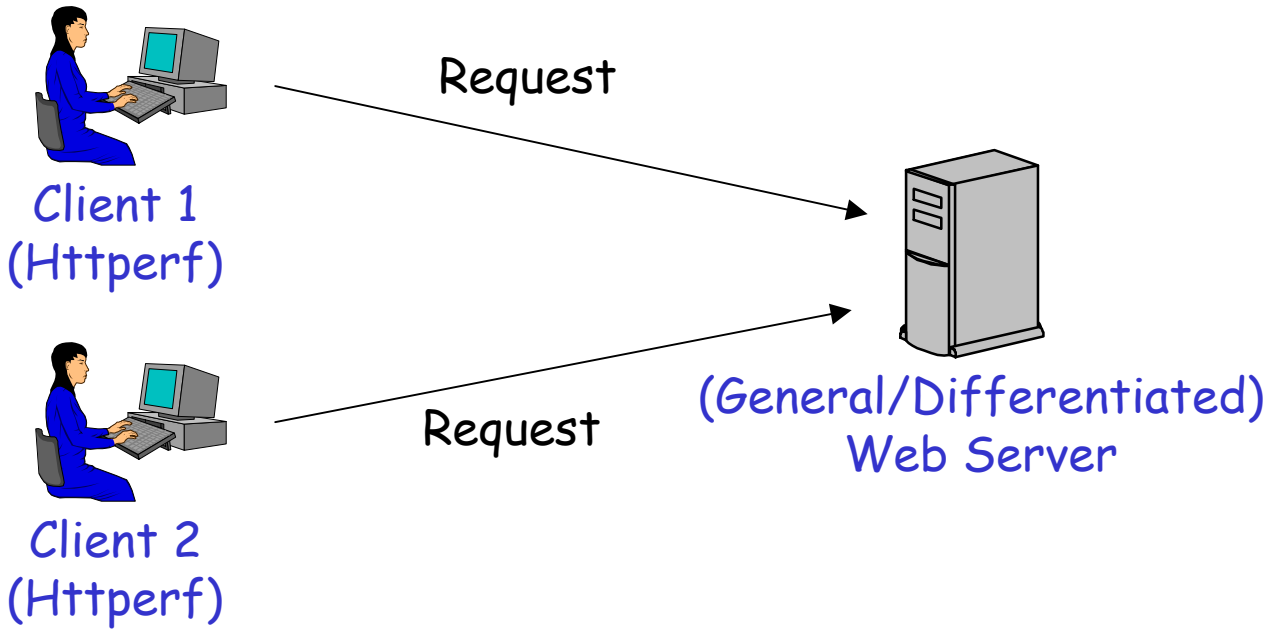
- Httperf –server hostname –port 80 –uri /test.html –rate 150 –num-conn 25000 –num-call 1 –timeout 5



Httperf Results

- Request part
  - Request rate (req / s)
  - Request size (B)
- Reply part
  - Reply rate (replies / s)
  - Reply time (ms)
  - Reply size (B)
- Errors part
  - Total number of errors
  - Type of errors (client-timeout, socket-timeout, connection refused, fd-unavailable, ftab-full)

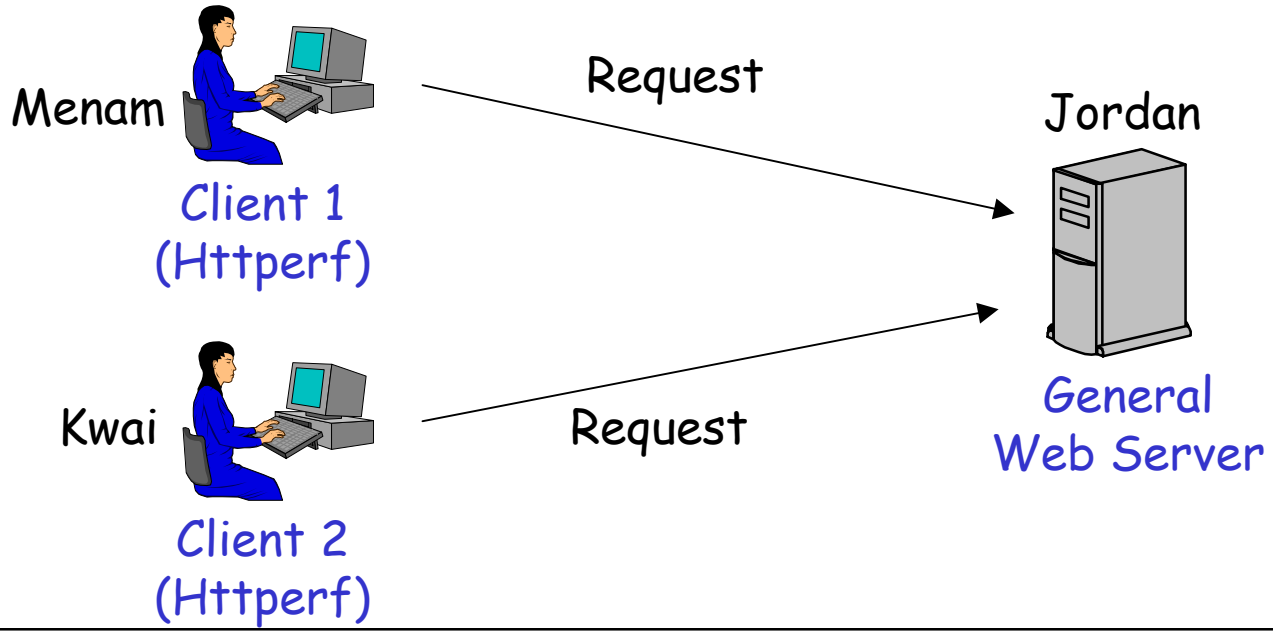
General Measurement Architecture



- Use Httperf (request generator & performance measurement)
- Change of request number per second

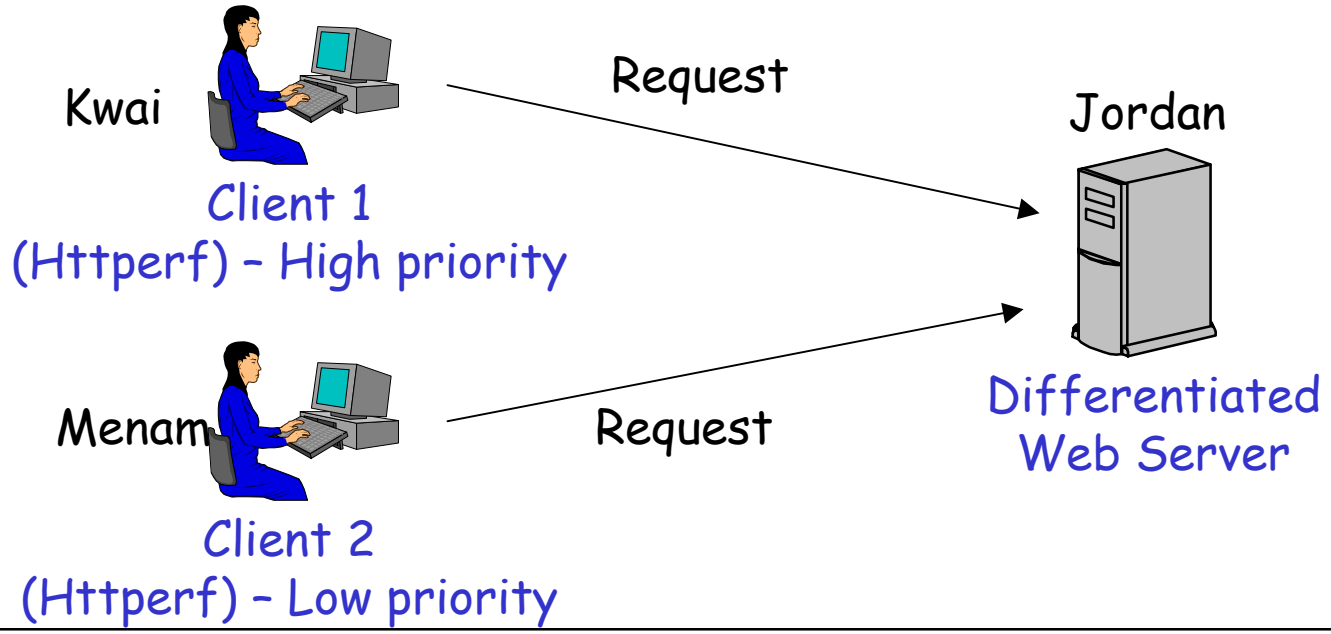
Measurement Testbed for General Web Server

- Menam : Pentium III-800, 256MB Ram, Red-hat Linux 6.2
- Kwai : Pentium II-233, 128MB Ram, Wow-Linux 6.2
- Jordan : Pentium 133, 128MB Ram, Mizi-Linux 1.1S



Measurement Testbed for Differentiated Web Server

- Menam : Pentium III-800, 256MB Ram, Red-hat Linux 6.2
- Kwai : Pentium II-233, 128MB Ram, Wow-Linux 6.2
- Jordan : Pentium 133, 128MB Ram, Mizi-Linux 1.1S

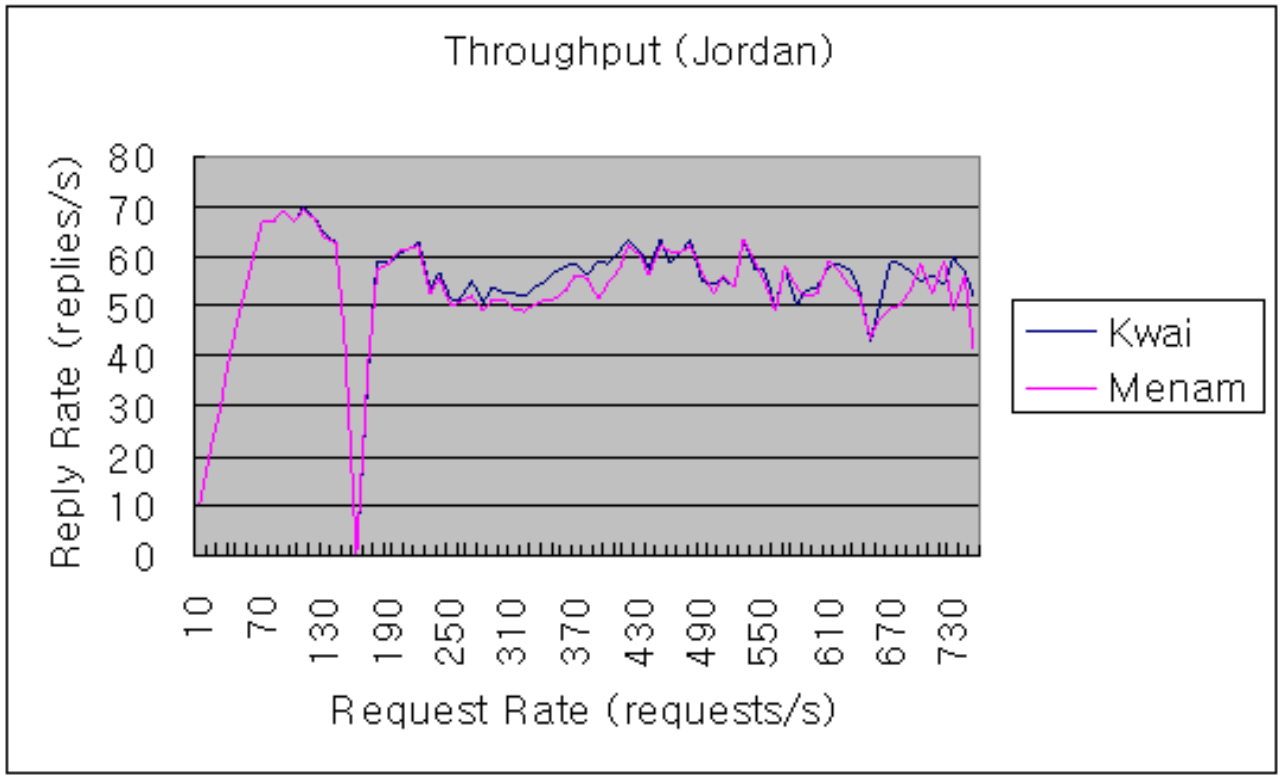


Method of Measurement

- 1 Client (General / Differentiated) Web Server
  - Using shell script
- 2 Clients (General / Differentiated) Web Server
  - Using timing synchronization (rdate –s [system])
  - Using crontab to execute shell script periodically

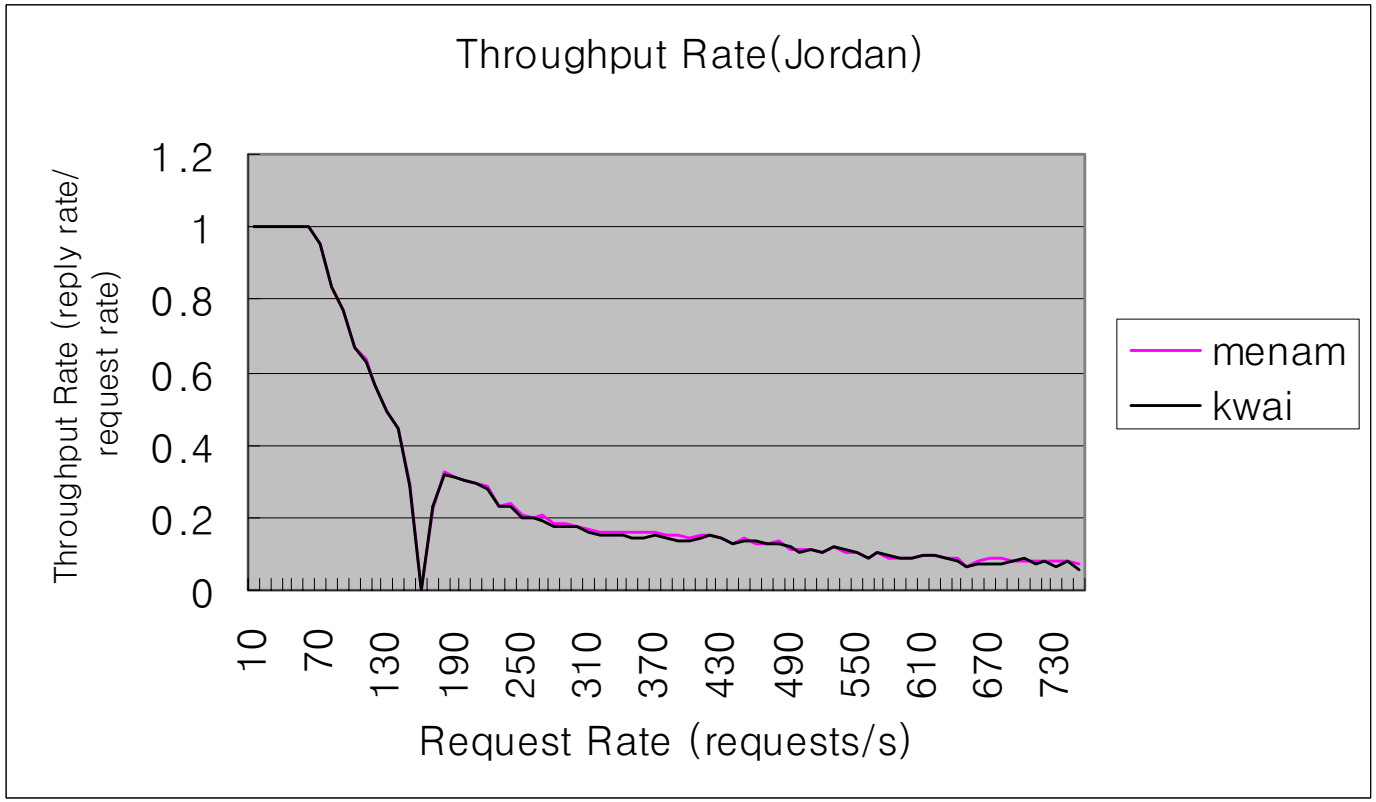
Evaluation Results for General Web Server (Two Clients) (1)

- Throughput



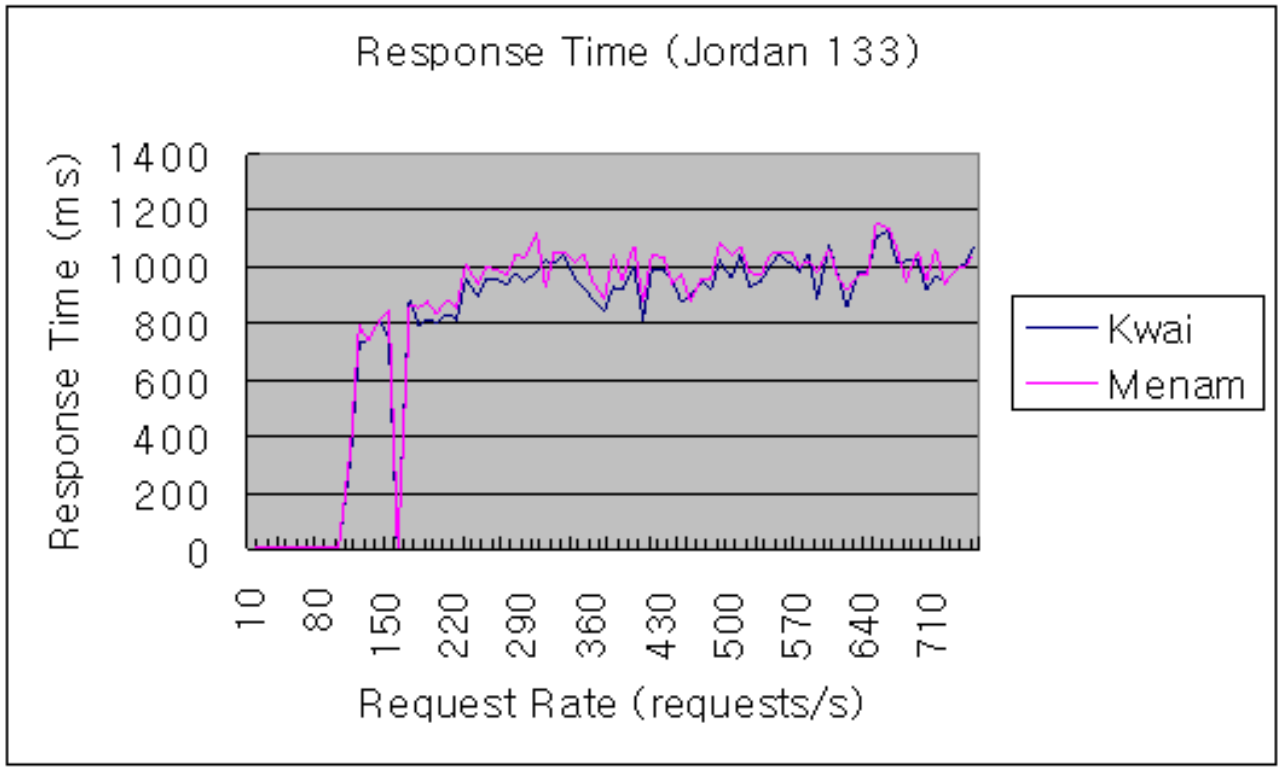
Evaluation Results for General Web Server (Two Clients) (2)

- Throughput Rate



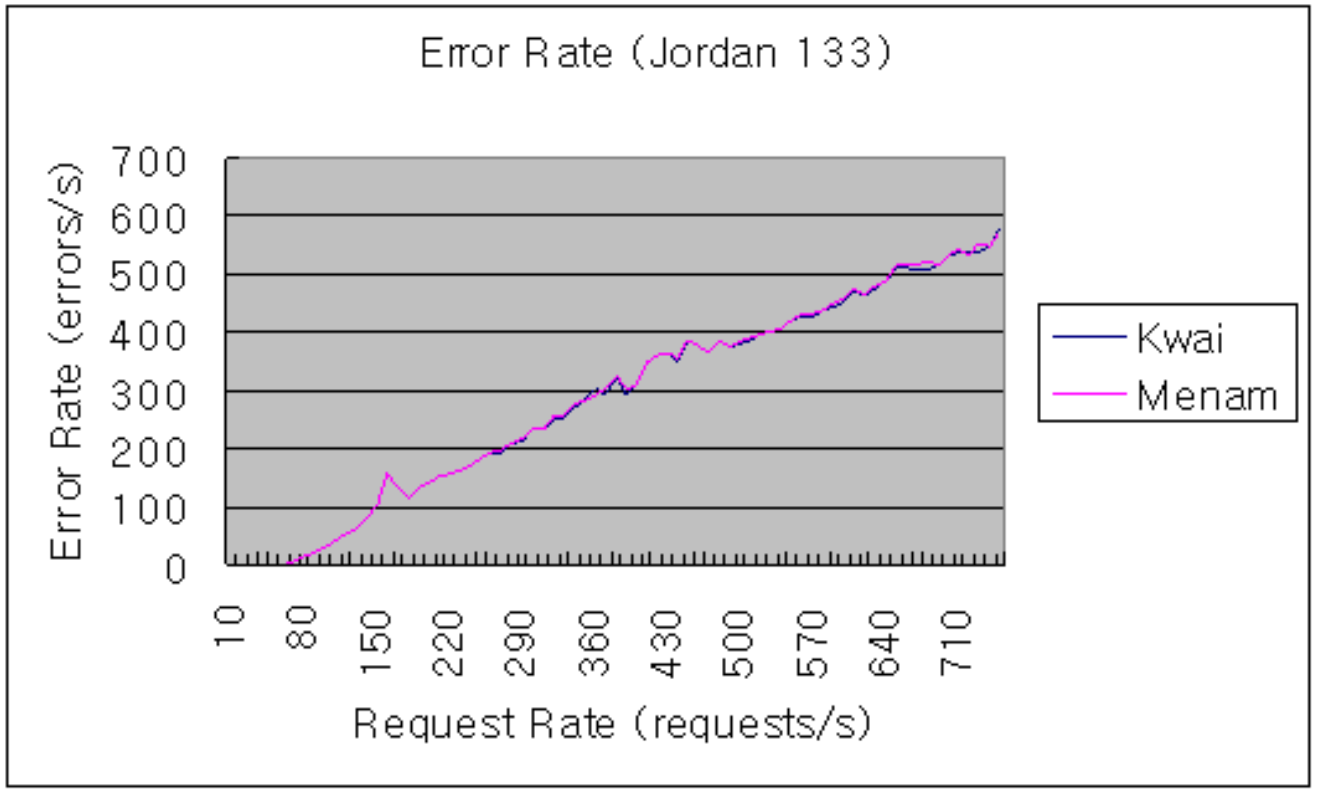
Evaluation Results for General Web Server (Two Clients) (3)

- Response Time



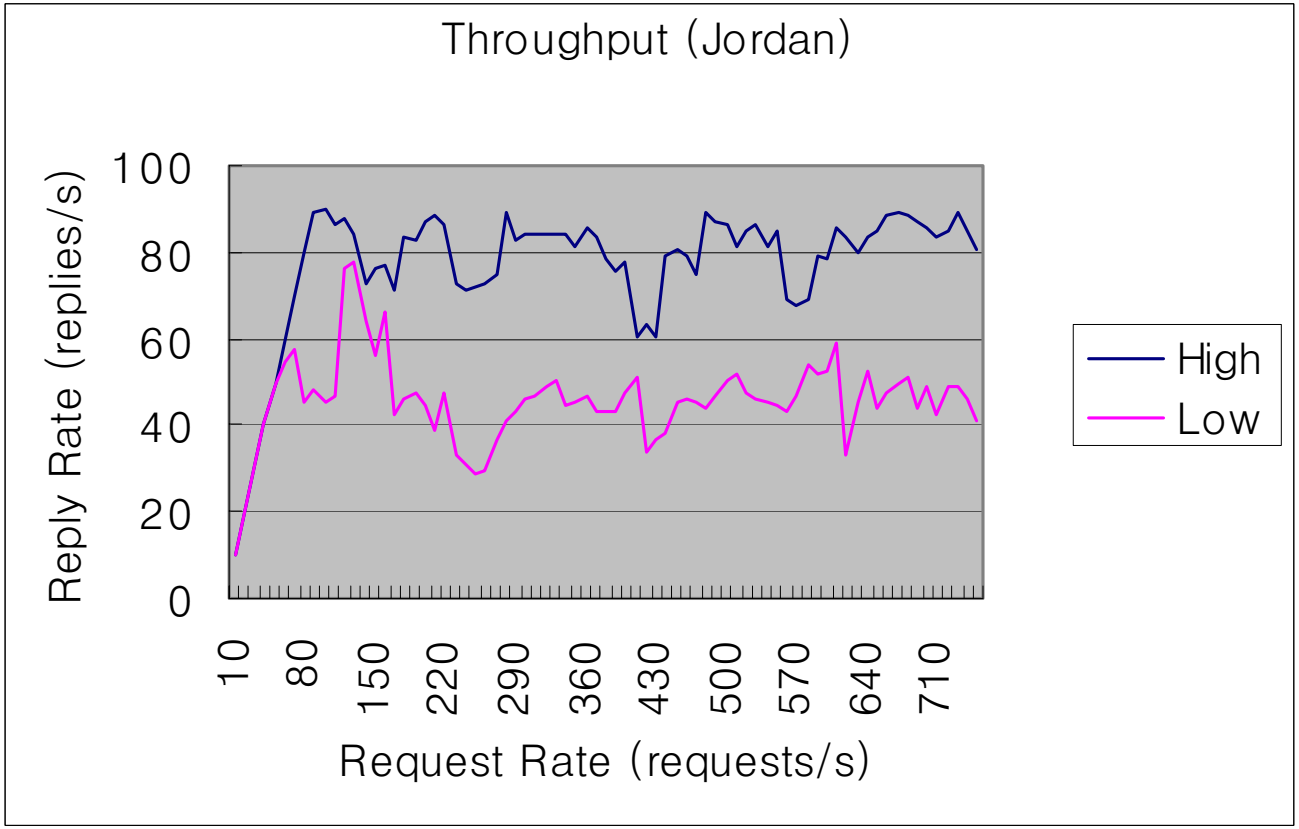
Evaluation Results for General Web Server (Two Clients) (4)

- Error Rate



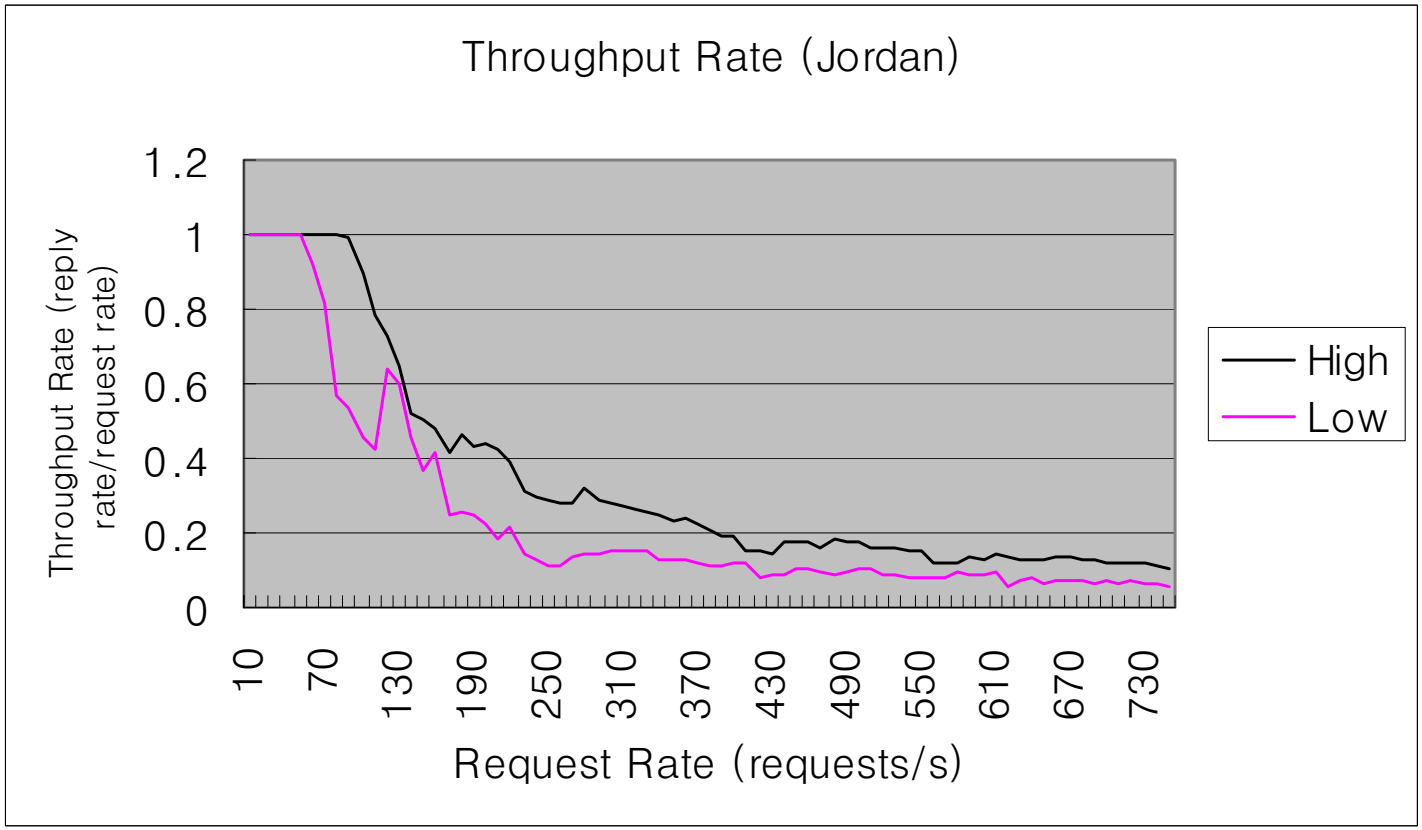
Evaluation Results for Differentiated Web Server (Two Clients) (1) – Kernel-Level Approach

- Throughput



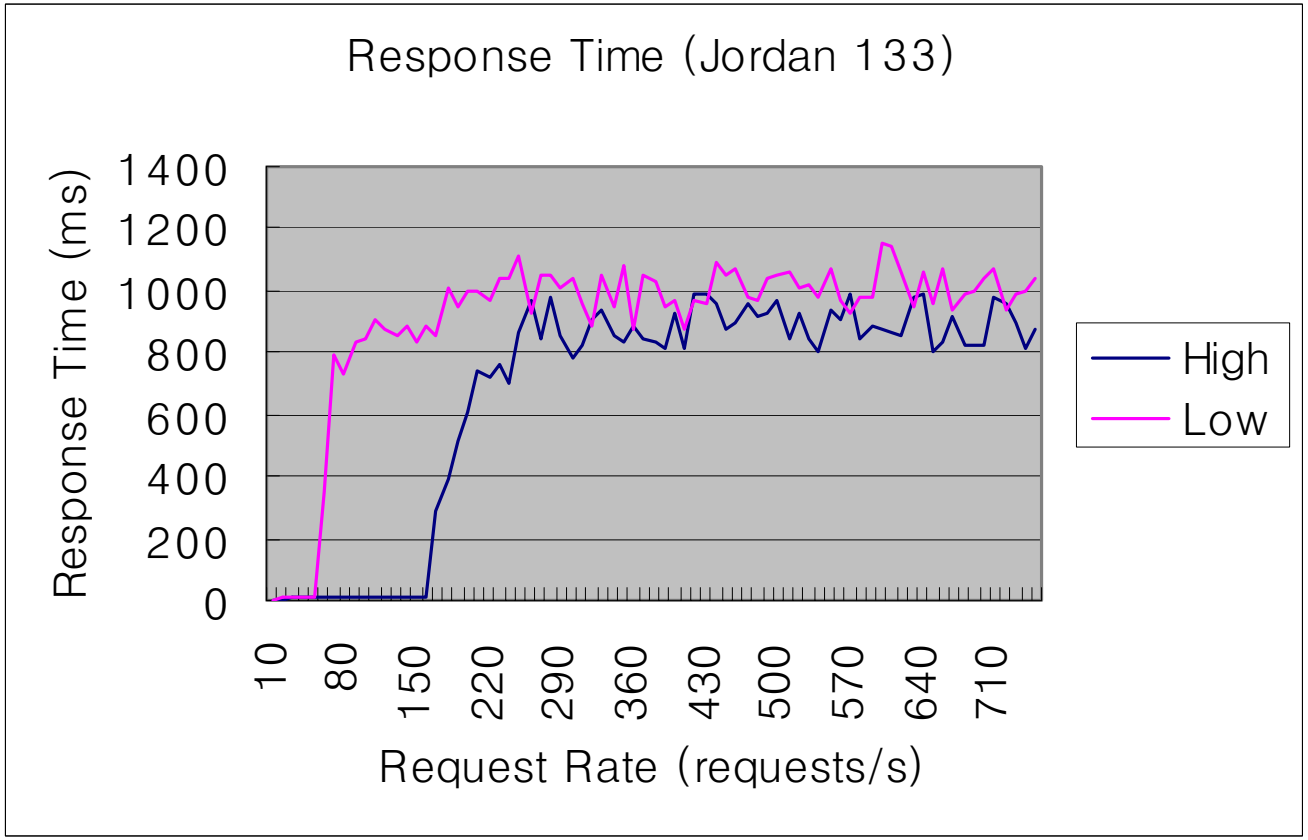
Evaluation Results for Differentiated Web Server (Two Clients) (2) – Kernel-Level Approach

- Throughput Rate



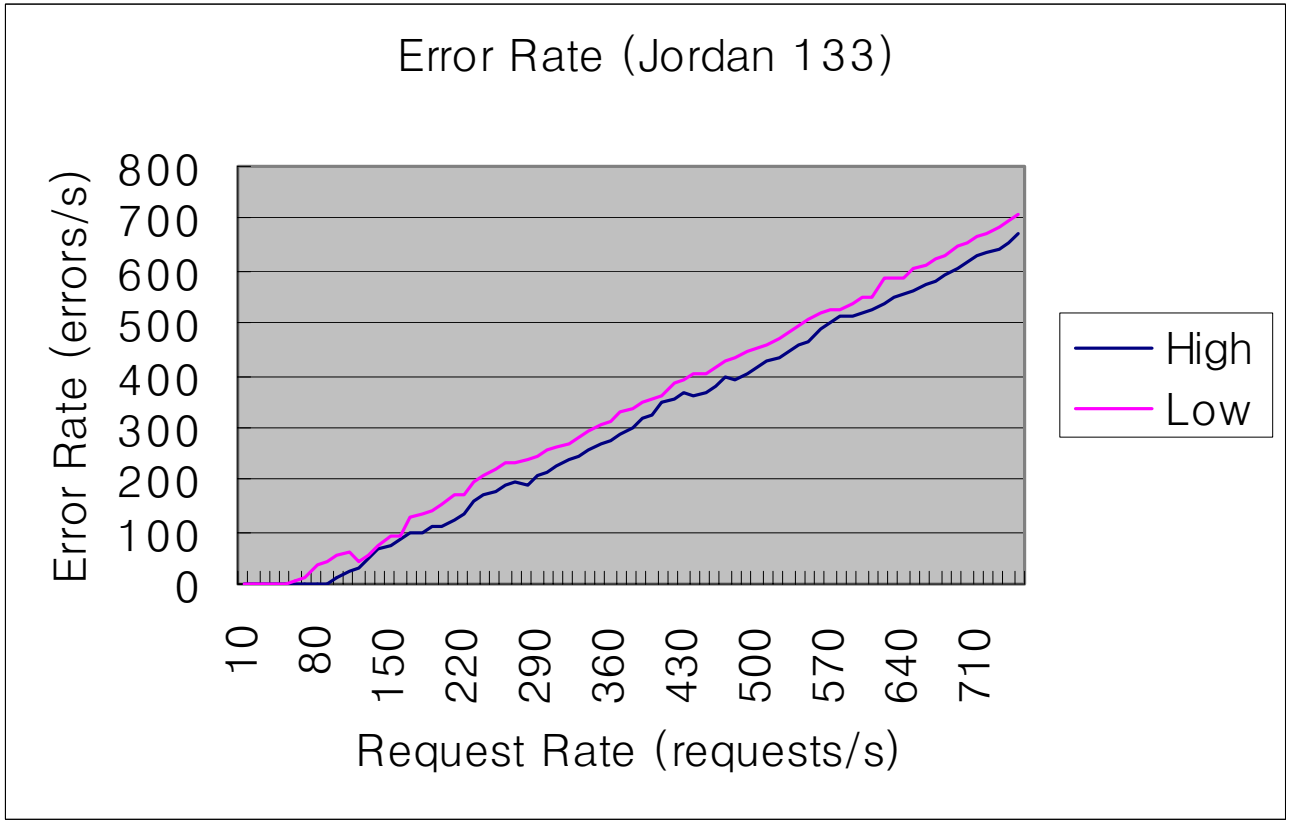
Evaluation Results for Differentiated Web Server (Two Clients) (3) – Kernel-Level Approach

- Response Time



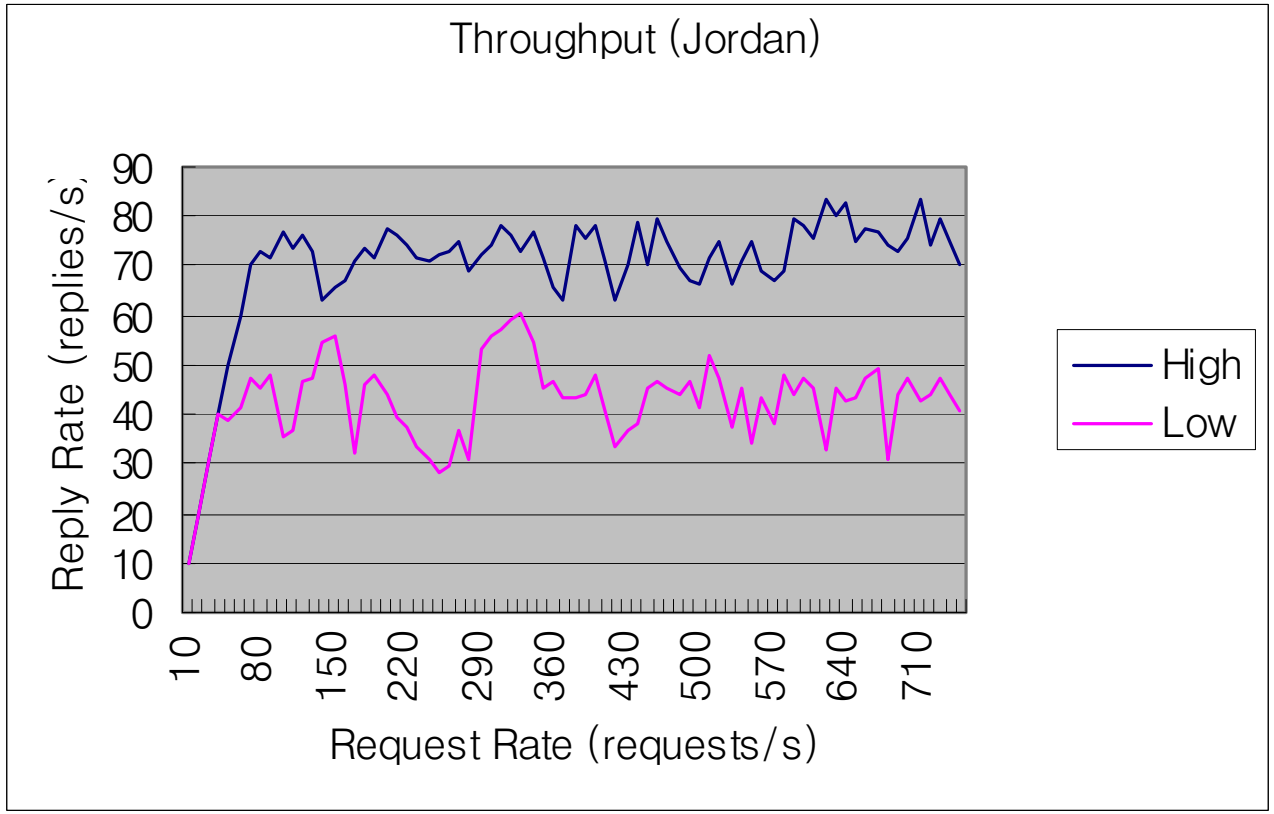
Evaluation Results for Differentiated Web Server (Two Clients) (4) – Kernel-Level Approach

- Error Rate



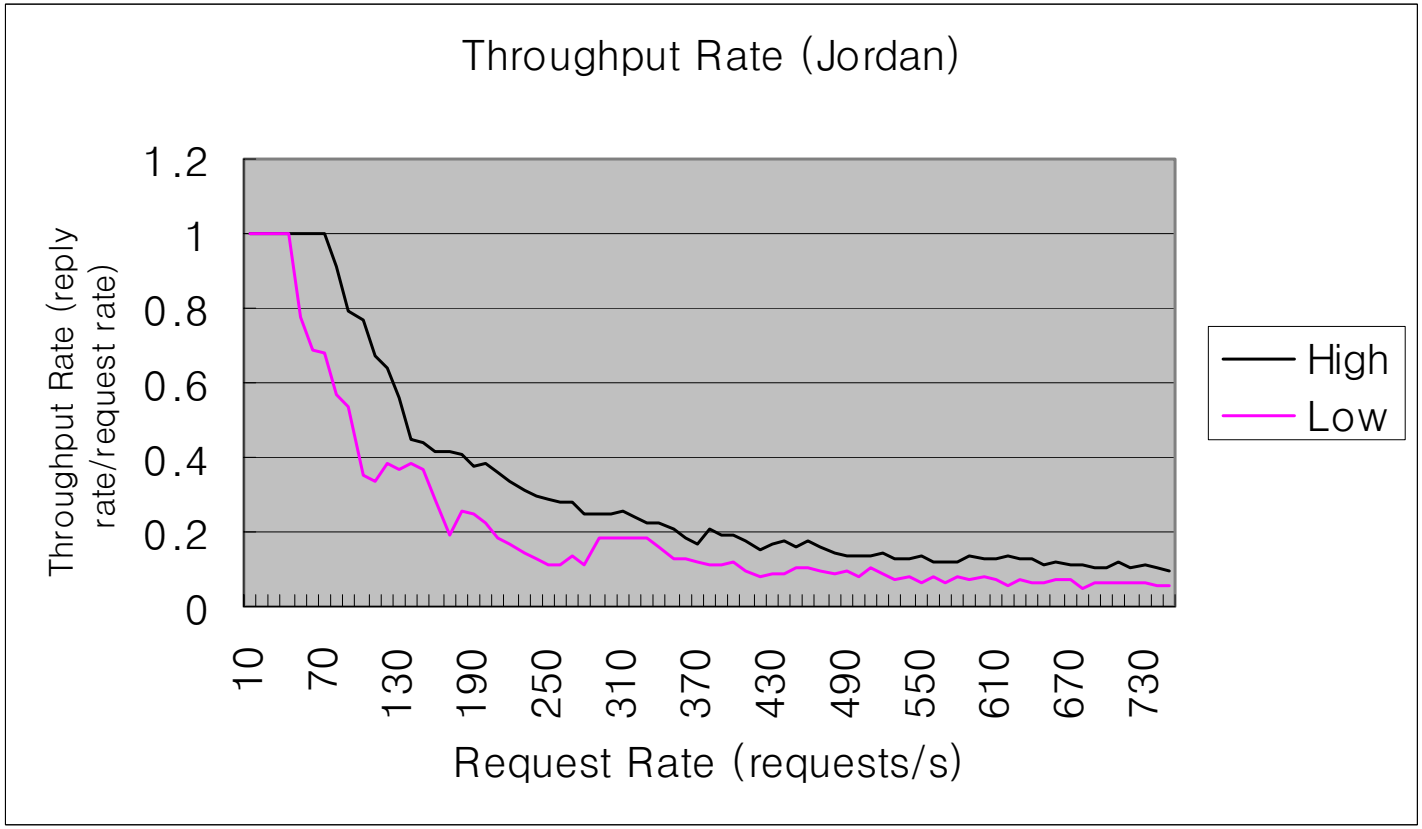
Evaluation Results for Differentiated Web Server (Two Clients) (1) – User-Level Approach

- Throughput



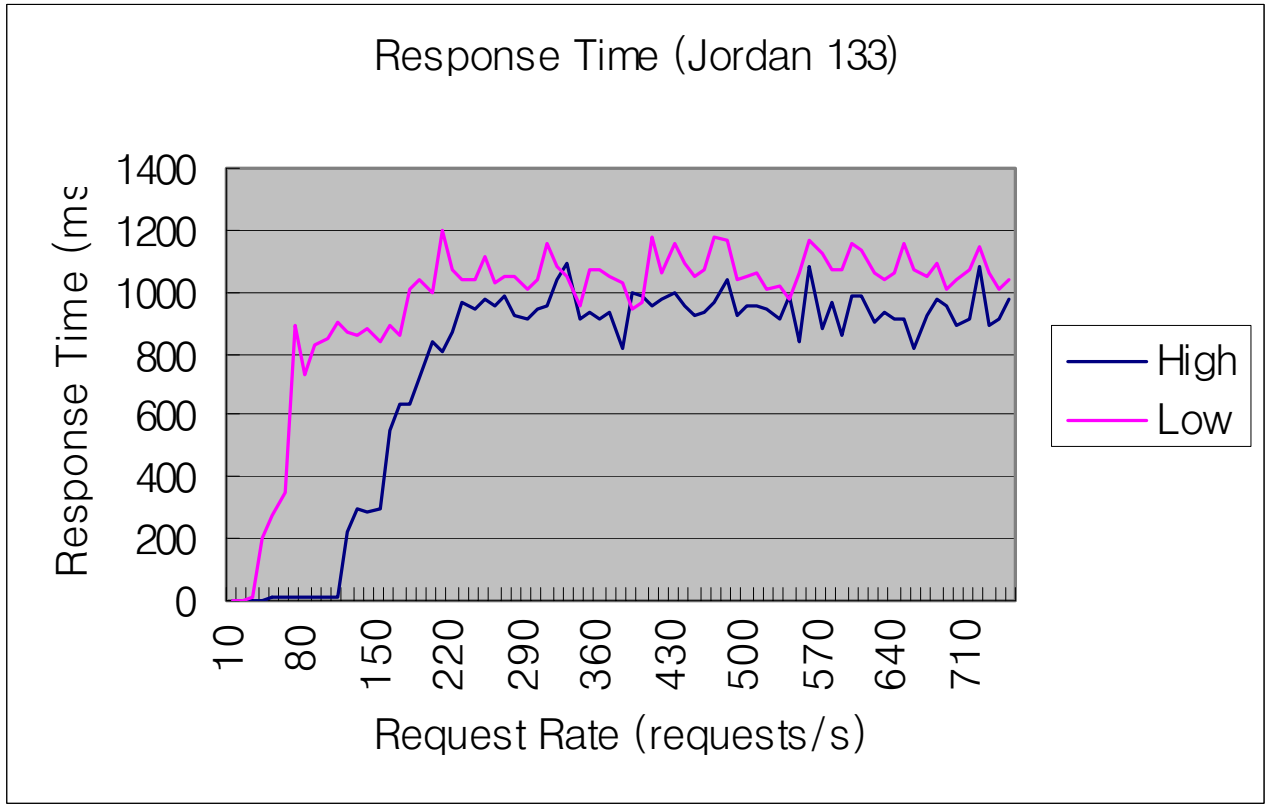
Evaluation Results for Differentiated Web Server (Two Clients) (2) – User-Level Approach

- Throughput Rate



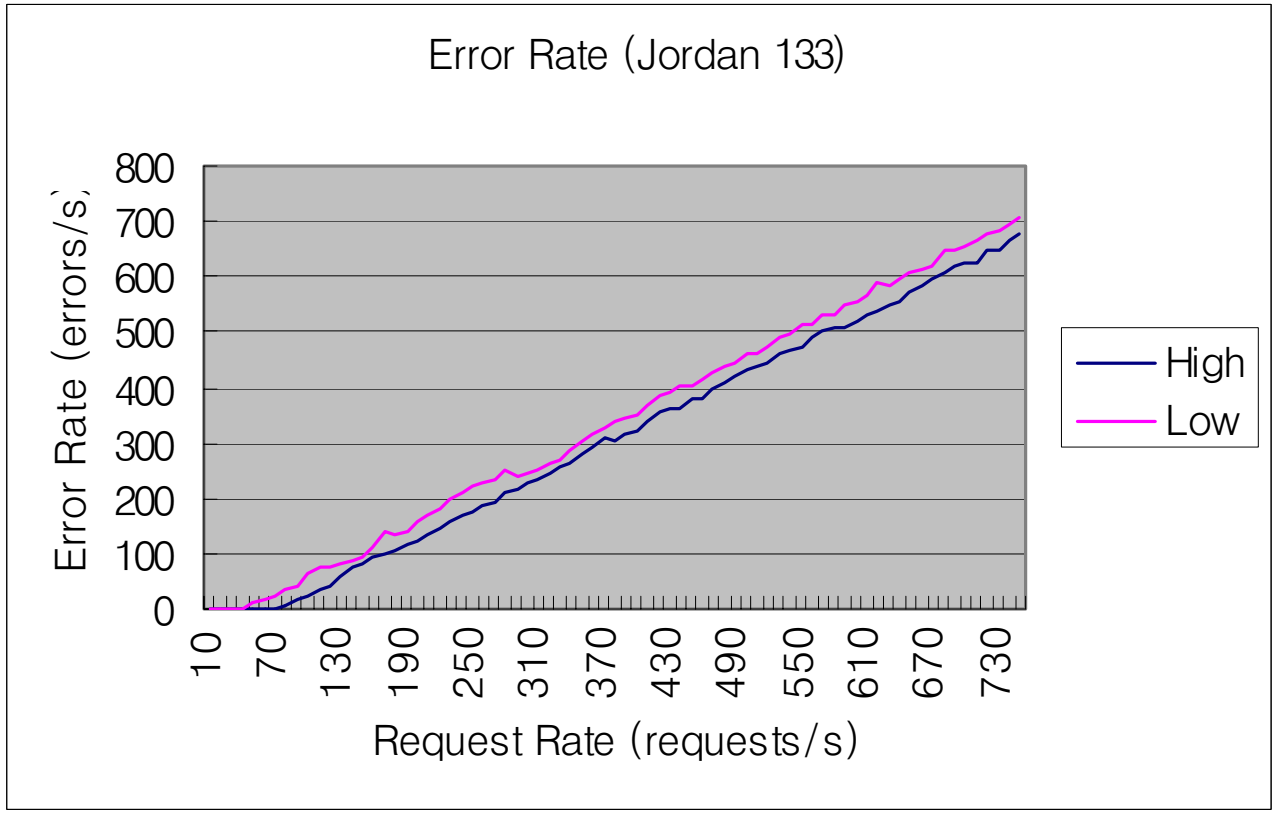
Evaluation Results for Differentiated Web Server (Two Clients) (3) – User-Level Approach

- Response Time



Evaluation Results for Differentiated Web Server (Two Clients) (4) – User-Level Approach

- Error Rate



Conclusion

- Users demand for differentiated quality of Web service, but current Web services cannot satisfy this
- We developed support to satisfy the users needs
- Our scheme categorizes HTTP requests into classes based on classification methods, with the requests of each class handled differently → differentiated QoS is possible

Future Work

- Applying the current work in real Web Services
- A study for Web Server Framework supporting Server QoS
  - Consideration of admission control
  - Consideration of resource management
  - Consideration of disk scheduling
- A study for integrating between Server QoS and Network QoS
  - Unification of QoS parameter
  - Consideration of ToS(Type of Service) field
- Extending current work on Web Server to other Internet Servers (e.g, FTP server, VOD server, RealAudio server, etc.)