Shaping Process Semantics [⋆]
[Extended Abstract]

Christoph M. Kirsch Harald Röck

Department of Computer Sciences
University of Salzburg, Austria
{ck,hroeck}@cs.uni-salzburg.at

Analysis. Composition of virtually all concurrent, distributed, real-time, or embedded software running on an operating system is based on some notion of software processes. Operating systems including many real-time operating systems provide widely used abstractions of which software processes are probably still the most successful. A (software) process is typically a sequential or multi-threaded program with an isolated virtual address space and an execution context, which contains state information such as the processor registers and scheduling status as well as timing and I/O descriptors. Interprocess communication (IPC) provides disciplined and well-understood means to overcome process isolation, in many cases, even across machines. In general, processes compute in isolation and invoke system calls to control timing, perform I/O and IPC, and request memory, as indicated by Figure 1. A modern operating system kernel usually handles such calls with dedicated subsystems for interrupt handling, I/O scheduling, IPC handling, and memory management. The systems community has devoted a lot of attention to their semantics and performance with an arguable preference for performance. However, the behavioral requirements of many software applications on timing, I/O, IPC, and memory are often inadequately addressed by the common focus on highest throughput and lowest latency. Since the observable behavior of processes is effectively determined by the invoked system calls, *serving processes as fast as possible*, e.g., by executing system calls using the fastest I/O schedulers available, may only provide an incomplete solution.

For example, delivering a web page in 100ms rather than 10ms does not make much of a difference to a web surfer. However, if it takes 10s just because someone else is currently downloading a large file from the same sever, expectations are clearly not met. A more-dimensional space of permissible behaviors rather than just maximum speed may therefore characterize the situation more appropriately while the range of what is permissible typically decreases with the requirements. As opposed to a web server, a streaming server usually requires access guarantees to a rather confined frequency band on the network. Similar to decreasing web surfing latency, increasing streaming throughput beyond that band has obviously little benefit. The range of permissible behaviors is even smaller for real-time and embedded applications such as digital controllers where the emphasis is typically on predictable, low latency rather than average, high throughput. Interestingly, the networking community has already addressed such behavioral requirements quite effectively using a variety of queueing techniques

⋆ Supported by Austrian Science Fund grant P18913-N15

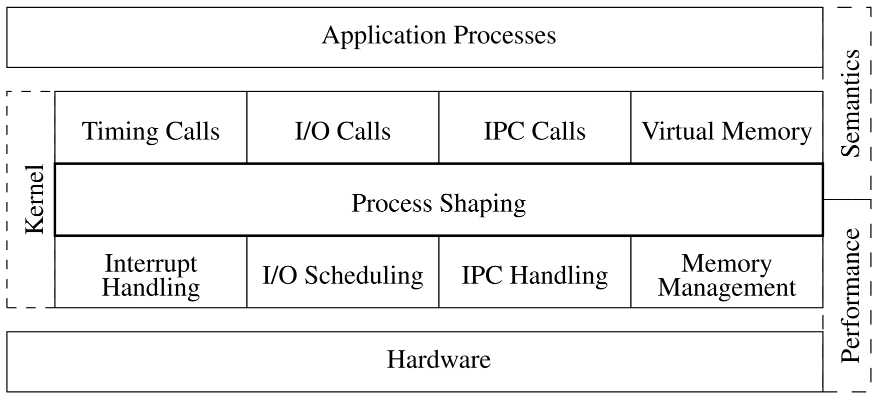


Fig. 1. A process-shaping operating system kernel

and traffic shaping. The idea is to force, i.e., shape traffic into a desired form before handling it. It turns out that such shaped traffic often not only matches application requirements better but can also be handled more efficiently. For example, the success of VoIP applications on nondeterministic but fast networks such as Gigabit Ethernet, which made deterministic networks such as ATM become unexpectedly obsolete, is rooted in the combination of unprecedented transmission speed and adequate prioritization. Higher speed means shorter packet transmission times making prioritization, i.e., traffic shaping, increasingly effective. The result is an inexpensive approximation of quality of service by probabilistic arguments rather than by deterministic but expensive guarantees.

Proposal. We propose the notion of *process shaping* to complement, not replace, the notion of serving processes as fast as possible, in analogy to traffic shaping in networks. Process shaping changes the order, times, and possibly the way in which potentially all side effects of processes, e.g., invoked system calls but also memory page faults, are handled before given to any performance-oriented kernel subsystems such as I/O schedulers and memory management, as indicated in Figure 1. Process shaping promotes more disciplined system composition by providing stronger behavioral process semantics than existing operating systems. Process shaping identifies an unrecognized trade off between serving and shaping, and may even result in improved system-wide performance. As a prototype system, we have designed and implemented a user-space threading library called the TAP library [4, 3], which shapes processes by queueing timing-, network-, and disk-related system calls according to a variety of policies. Some experimental results are discussed below.

With process shaping, we advocate a shift in research attention from performance- to semantics-oriented handling of software processes. Interesting new questions beyond, e.g., traditional scheduling arise such as how to identify dynamically at runtime bandwidth limits and shaping policies for optimal system-wide utilization. Queueing networks [2] may be used in modeling the dynamically changing data flow in operating systems, e.g., from disk to network devices caused by a streaming server. Queueing theory and its real-time extensions [5] may already provide analytical frameworks, in particular, for systems with strict behavioral requirements such as many real-time and embedded systems.

Claims. Our proposal rests on semantics- and performance-related claims that need to be verified. With process shaping, the behavioral semantics of processes may have to be expressed in terms of yet to be developed languages that translate into a process-shaping infrastructure. We claim that typical networking terminology such as bandwidth, flow, burst, collision, and so on, in addition to already universal terminology such as frequency and latency, may also be used to describe many process-based application requirements, in particular, in real-time and embedded systems.

Process shaping is meant to complement, not replace, operating system facilities that operate under the regime of serving processes as fast as possible. We claim that the trends to higher processor speed, also in embedded systems, and to more efficient scheduling facilities and lower kernel latency in operating systems, in analogy to shorter packet transmission times, will make process shaping increasingly effective. The result will be stronger guarantees on process behavior and higher system-wide performance especially in overload scenarios. Note that real-time patches providing lower kernel latency, which used to be rather exotic, unsupported code, increasingly make their way into general-purpose operating systems such as Linux, even with industry support.

Experiments. We discuss two experiments to substantiate our proposal and claims. Both experiments have been conducted using our latest prototype implementation of the previously mentioned TAP library on an unmodified Linux 2.6.15 kernel. Multi-threaded applications such as many web and streaming servers can usually be linked against the POSIX-compliant TAP library without modifications. The applications' I/O calls, e.g., network and disk reads and writes, are then routed through the library and shaped according to a given bandwidth limit and queueing policy such as the well-known leaky- and token-bucket policies. Here, we have used a token-bucket policy for network and disk calls. A token-bucket policy shapes traffic into a steady stream below a given threshold but also allows infrequent bursts above the threshold. With this policy, traffic can only proceed if there are tokens available in a virtual bucket. Since the bucket is filled with tokens at a given rate but has a finite capacity, traffic can flow freely until the bucket is empty but then flows at a speed proportional to the token rate.

In the first experiment, we ran two multi-threaded web servers on a single machine accepting and serving incoming connections on a gigabit network. Two client machines repeatedly connect to the web servers and request the same and thus cached 380KB file. Figure 2 depicts the net throughput measured at the clients with shaping disabled, and when each web server is shaped to use at most 50% of the available network bandwidth. Note that each web server runs on its own instance of the TAP library, i.e., there is no explicit coordination between the two web servers. The combined throughput of the web servers without shaping shows a distinct peak of 110MB/s at 1300 connections per second, and decreases for higher connection rates to less than 90MB/s. The web servers with shaping do not achieve the same peak throughput but continue to stay at the same

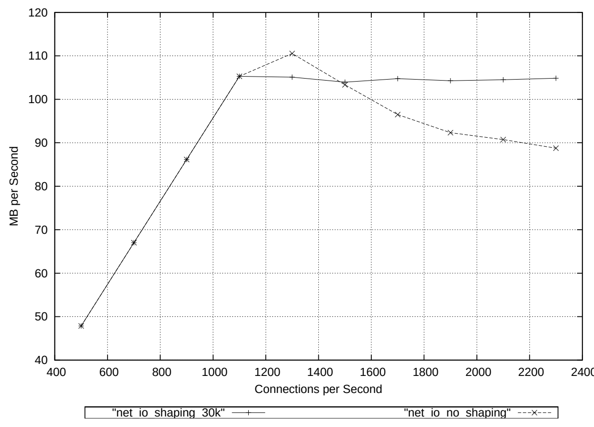


Fig. 2. The combined throughput of two web servers running concurrently on a single machine is more deterministic and scales to higher connection rates with process shaping than without it.

high rate of 105MB/s, and even achieve a higher throughput than without shaping for connection rates higher than 1500 connections per second.

In the second experiment, we ran a streaming server in addition to multiple web servers on the same machine as above. The streaming server reads video and audio data at around 700KB/s to 1MB/s from disk and then streams it over the network to a laptop, which is used as client. The video and audio quality on the client decreases if the server cannot read data from disk on time, e.g., since the disk is accessed by too many other processes, or if data packets cannot be transmitted, e.g., since the network is overloaded by other background traffic. To generate background traffic on the network, we use a web server, which gets hit with a high number of requests for a cached 380KB file. Additionally, to generate load on the disk, we run several other web servers, which process single requests for a 1GB file that is not cached. The optimization problem that arises here is to guarantee the streaming server sufficient access to disk and network while maximizing total disk and network throughput, and have net throughput approach raw throughput on the network as closely as possible. Net throughput refers to network traffic successfully received and processed by remote clients. Figure 3 shows total disk and network throughput (raw and net) for different process shaping configurations, and the estimated disk and network I/O threshold of 19MB/s and 100MB/s, respectively, at which the streaming server can still function properly. Without any shaping, raw network throughput is close to the devices capacity of 120MB/s but net throughput drops below 90MB/s because of thrashing, which also explains the relatively low disk throughput. By turning on network shaping alone, network throughput can be pushed below the 100MB/s network I/O threshold resulting in increased disk throughput, which, however, settles above the 19MB/s disk I/O threshold because of decreased thrashing. Thus the optimum can only be reached when shaping network and disk traffic simultaneously, in our case, using 19k network and 14.4 disk token rates (at the 15.6k disk token rate, video and audio quality begin degrading).

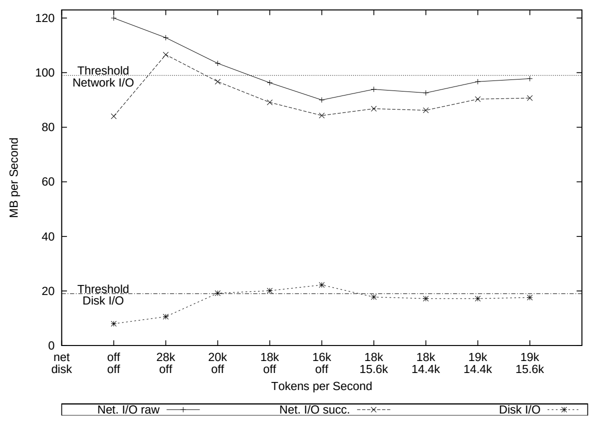


Fig. 3. A streaming server and multiple web servers running on a single machine. The optimal system "shape" with best net throughput and streaming performance as well as maximum non-thrashing disk and network utilization (19MB/s disk and 100MB/s network) is reached when processes are shaped by 14.4k disk tokens per second and 19k network tokens per second using a token-bucket policy.

Future. We feel that our experiments show the potential of process shaping. Our next, short-term steps are to design and implement a kernel-level version of our prototype implementation, and to study ways to identify dynamically at runtime bandwidth limits and shaping policies for optimal system-wide utilization. Our long-term goals are to support even hard real-time applications such as the flight control system of our unmanned quadrotor helicopter called the JAviator [1].

References

1. J. Auerbach, D.F. Bacon, D. Iercan, C.M. Kirsch, H. Röck, and R. Trummer. The JAviator Project. http://javiator.cs.uni-salzburg.at/.
2. R.O. Baldwin, N.J. Davis, S.F. Midkiff, and J.E. Kobza. Queueing network analysis: concepts, terminology, and methods. *Systems and Software*, 66(2):99–117, 2003.
3. C.M. Kirsch and H. Röck. The TAP Project. http://tap.cs.uni-salzburg.at/.
4. C.M. Kirsch and H. Röck. Traffic shaping system calls using threading by appointment. Technical Report T009, Department of Computer Sciences, University of Salzburg, August 2005.
5. J.P. Lehoczky. Real-time queueing theory. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, 1996.

Shaping Process Semantics

Christoph Kirsch
Universität Salzburg

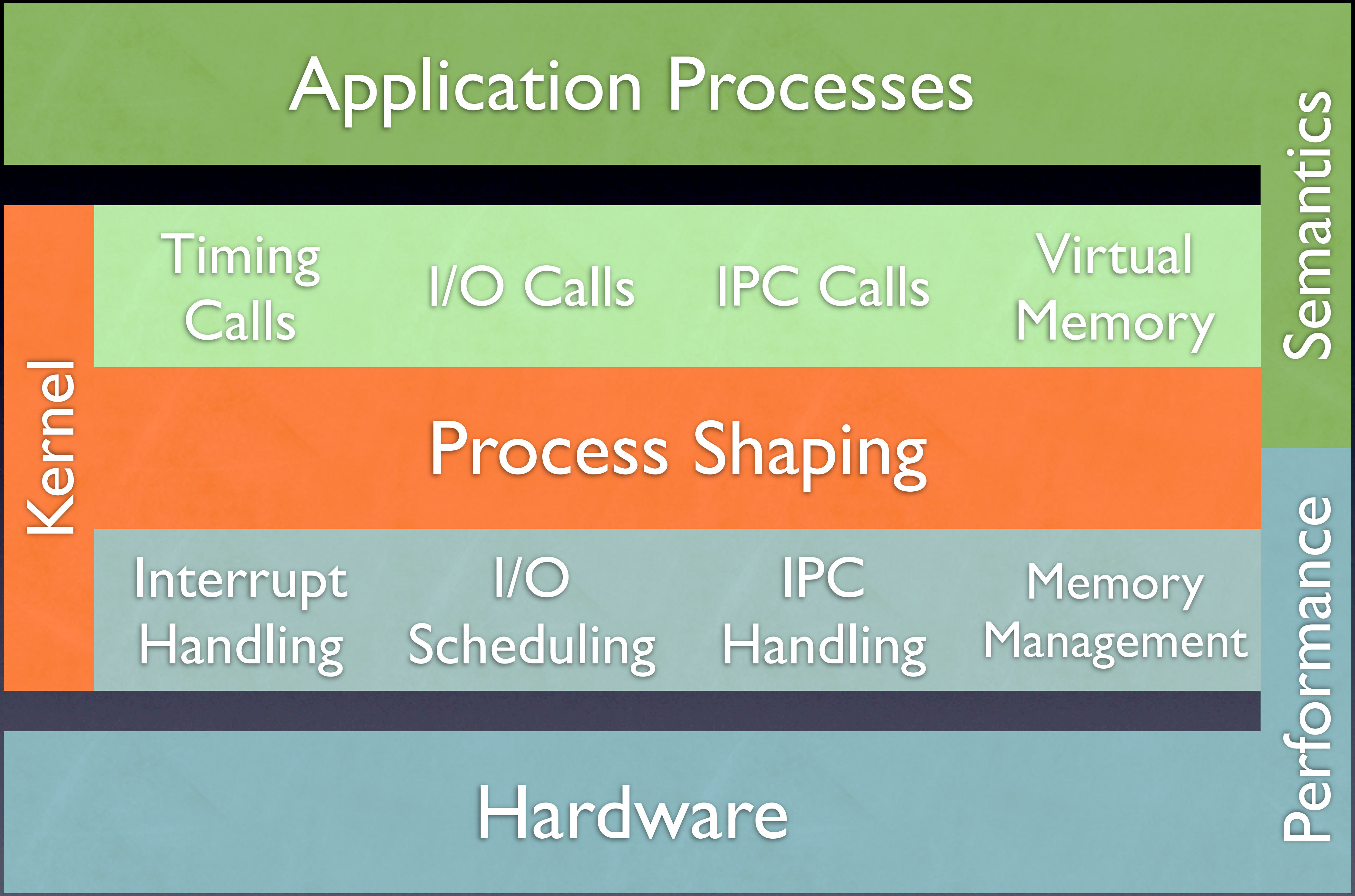
Joint work with Harald Röck
Monterey Workshop, Paris, October 2006

The Idea

- we apply *traffic shaping* technology known in the networking community to *software processes*
- software processes invoke *system calls* to access resources, perform I/O, etc.
- we see system calls as *network packets*

Process Shaping

- process shaping changes the *order* and *times* in which system calls (and potentially other *side effects* of processes) are handled before given to any performance-oriented kernel subsystems
- process shaping promotes more disciplined *system composition*


Application Processes
Kernel
Timing Calls
I/O Calls
IPC Calls
Virtual Memory
Process Shaping
Interrupt Handling
I/O Scheduling
IPC Handling
Memory Management
Hardware
Semantics
Performance

Proposal

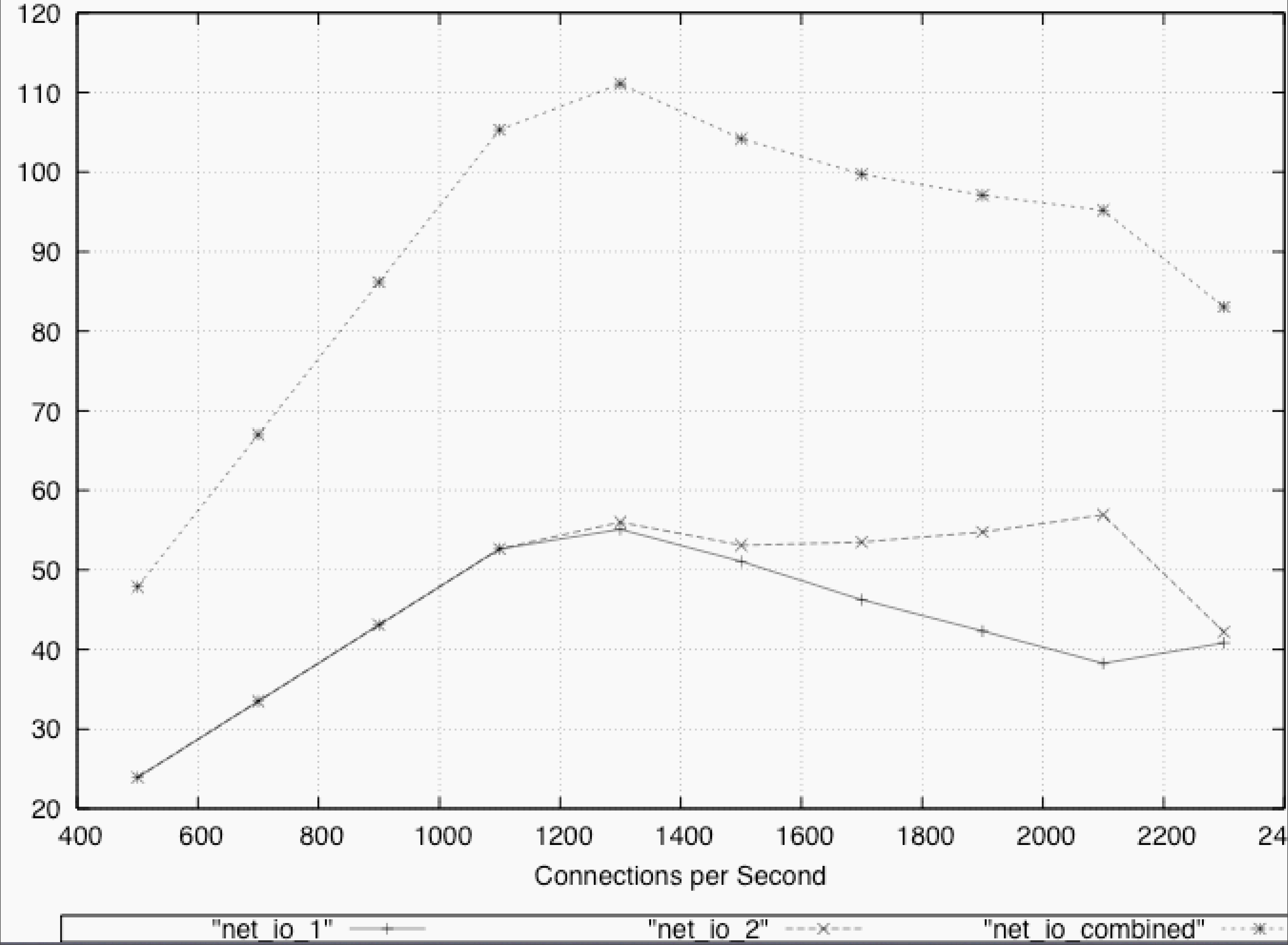
- we propose the notion of *process shaping* to complement, not replace, the notion of *serving* processes *as fast as possible*
- we advocate a shift in research attention from performance- to *semantics-oriented* handling of software processes

Claim

- we claim that *faster* processors, more efficient scheduling, and lower kernel *latency*, in analogy to *shorter* packet transmission times, will make process shaping increasingly effective
  - see ATM versus Gigabit Ethernet
- note that used-to-be-exotic real-time patches increasingly make their way into general-purpose operating systems

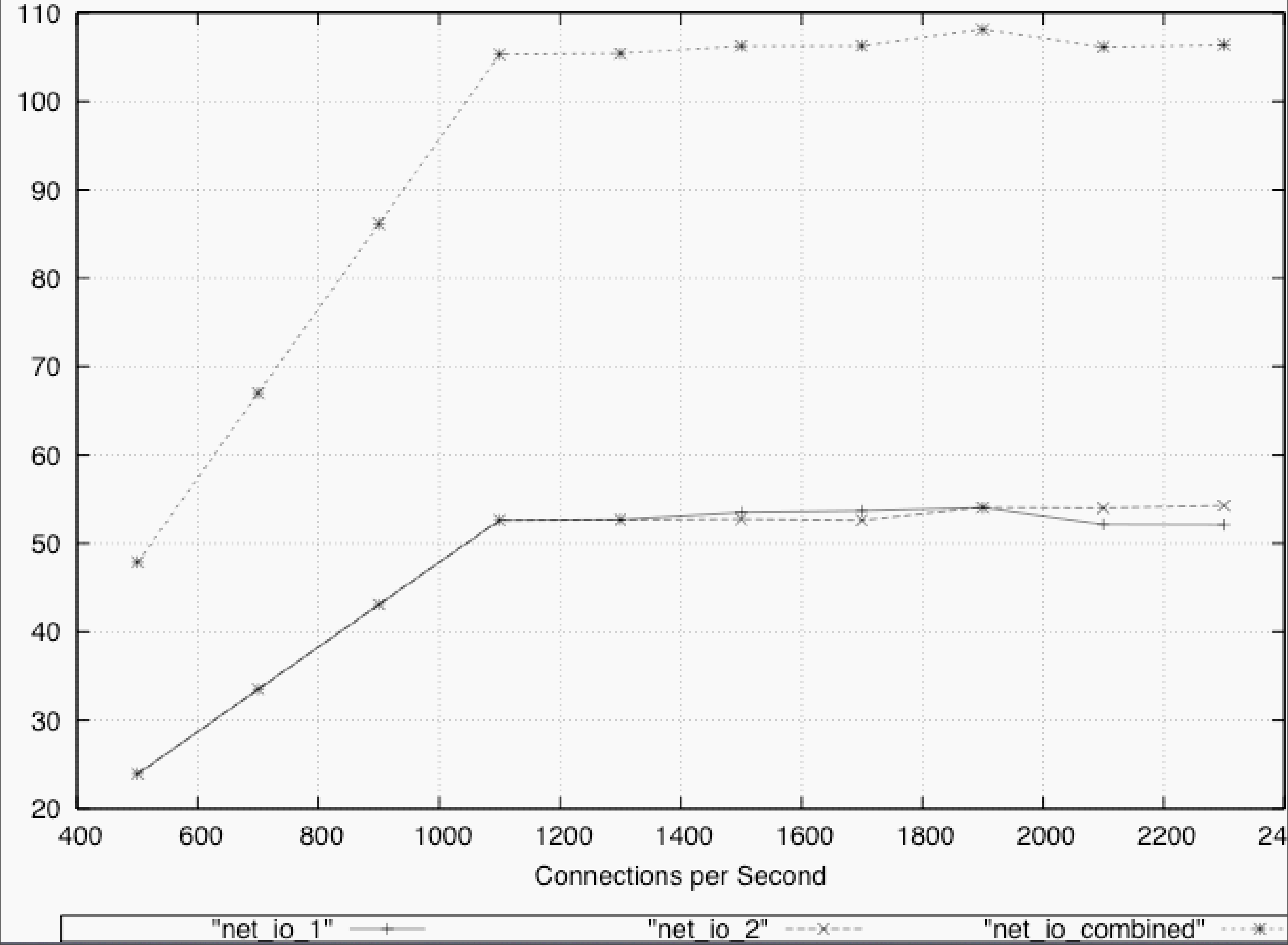
Experiment I

- we run two separate web server processes on an *unmodified* Linux 2.6 server machine with Gigabit Ethernet
- two client machines generate workload by requesting the same and thus *cached* 380KB file


120
110
100
90
80
70
60
50
40
30
20
400
600
800
1000
1200
1400
1600
1800
2000
2200
24
Connections per Second
"net_io_1"
"net_io_2"
"net_io_combined"

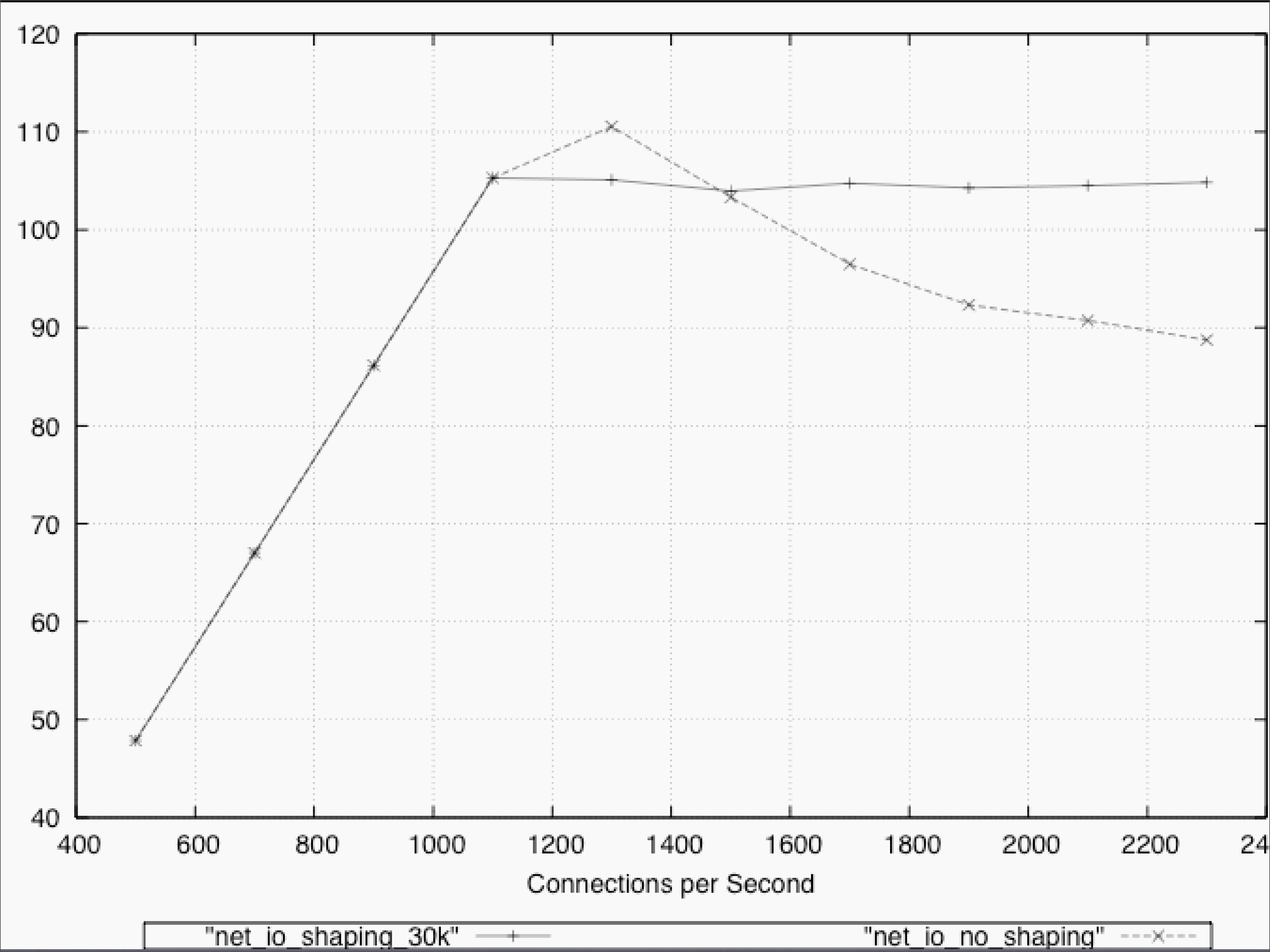
Experiment II

- we run two separate web server processes on a *process-shaping* Linux 2.6 server machine with Gigabit Ethernet
- two client machines generate workload by requesting the same and thus *cached* 380KB file


110
100
90
80
70
60
50
40
30
20
400
600
800
1000
1200
1400
1600
1800
2000
2200
24
Connections per Second
"net_io_1"
"net_io_2"
"net_io_combined"

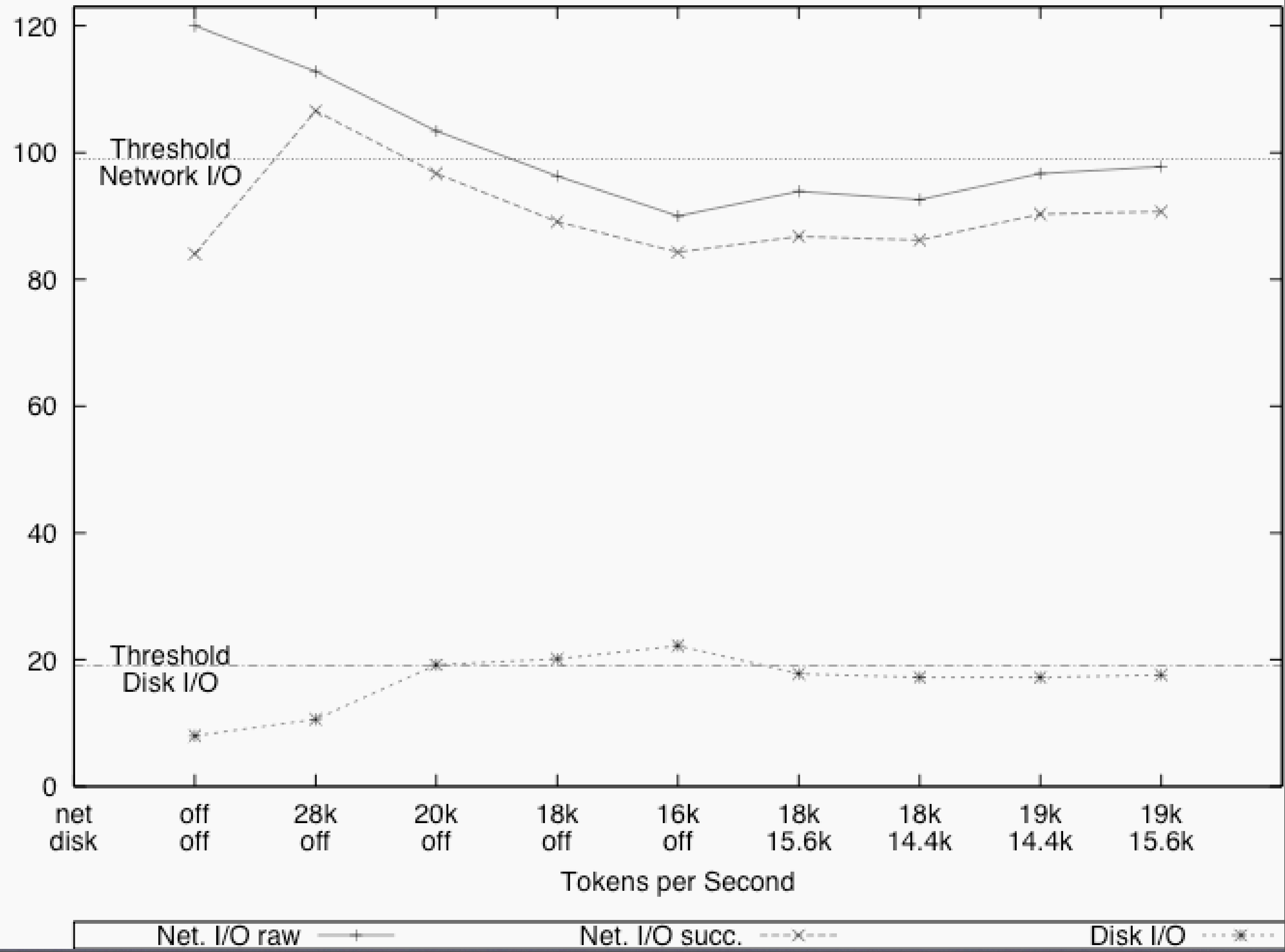
Experiment I+II

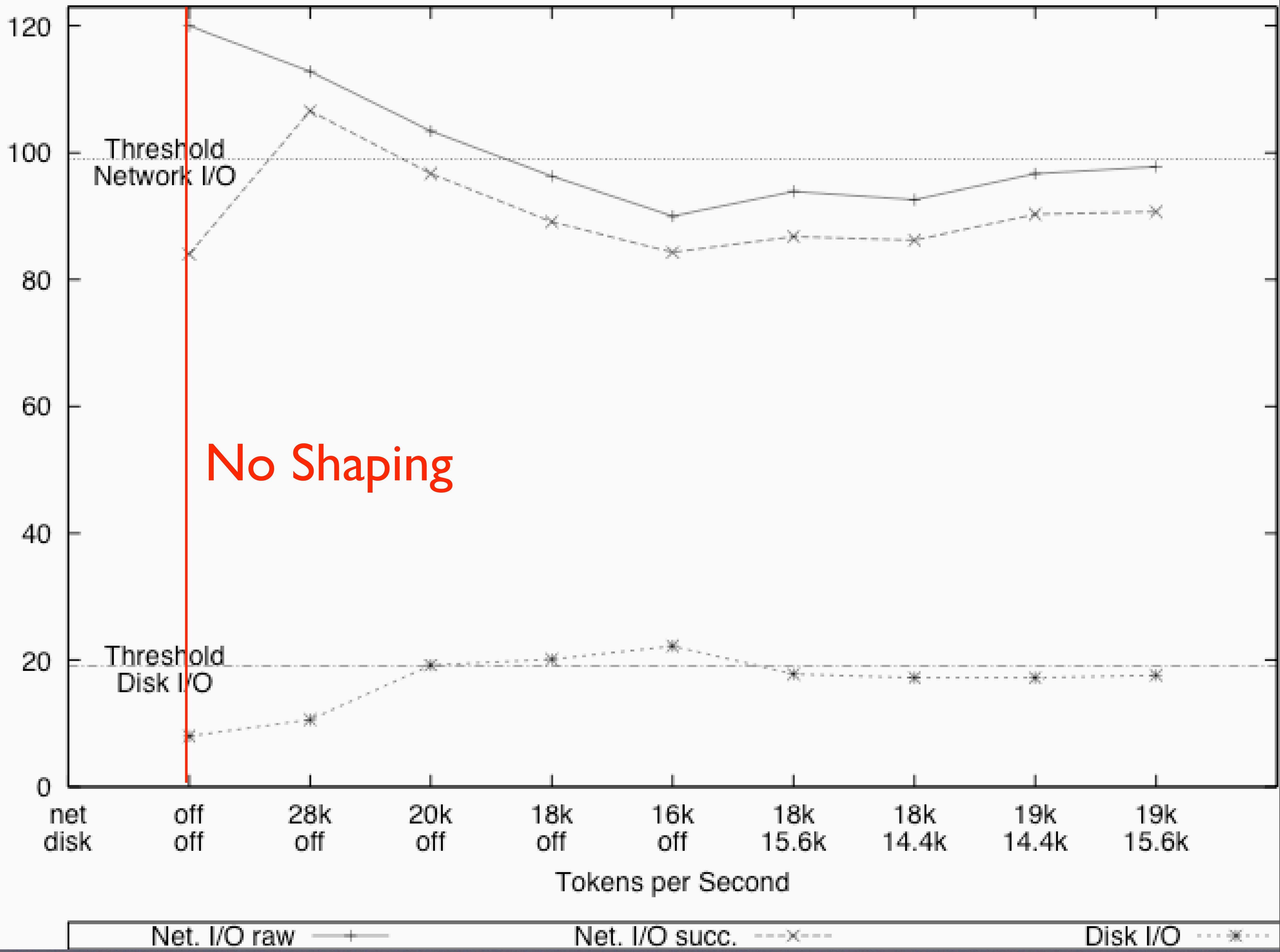
- higher total peak performance without process shaping
- but total peak performance more robust with process shaping


120
110
100
90
80
70
60
50
40
400
600
800
1000
1200
1400
1600
1800
2000
2200
24
Connections per Second
"net_io_shaping_30k"
"net_io_no_shaping"

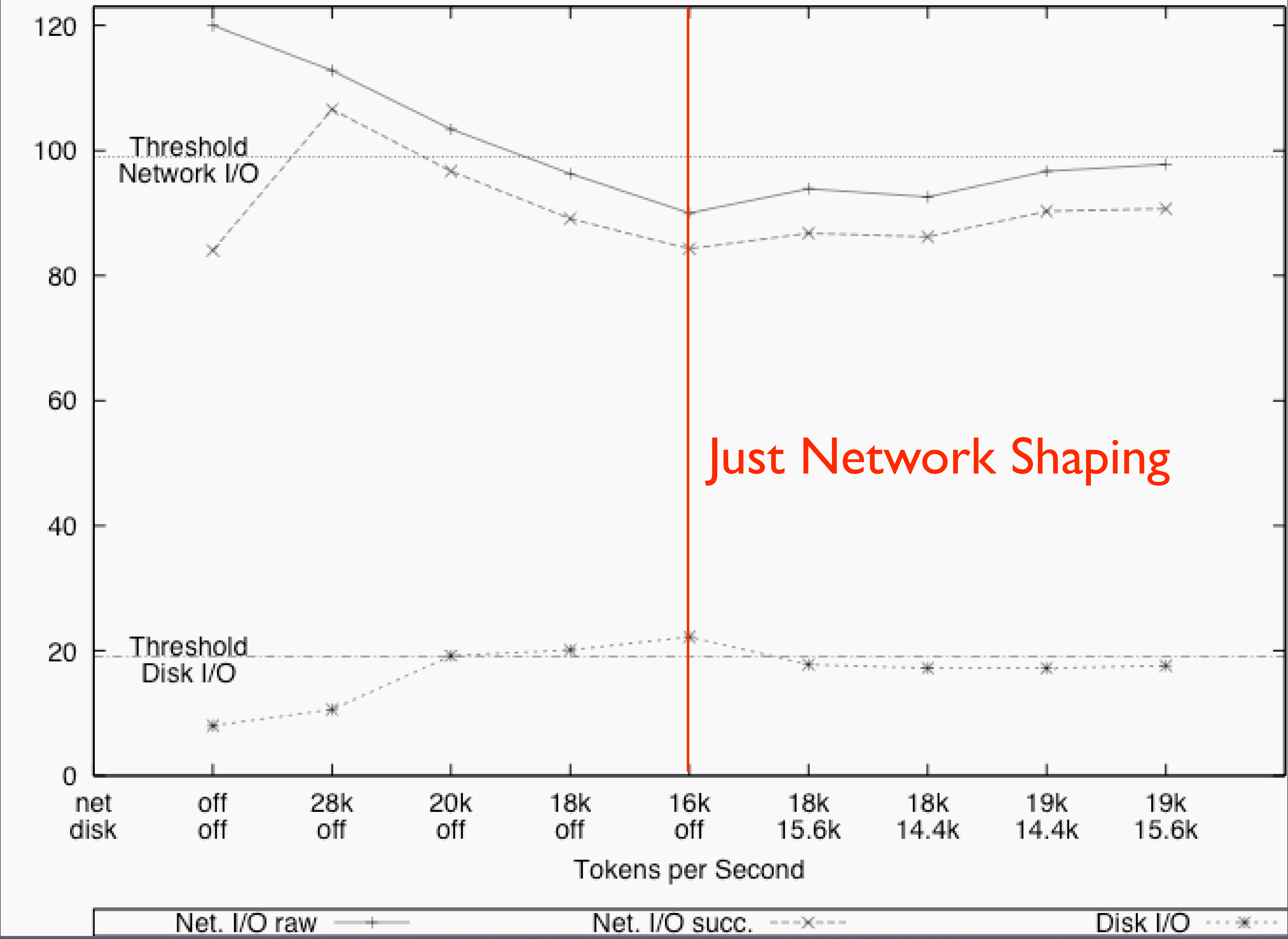
Experiment III

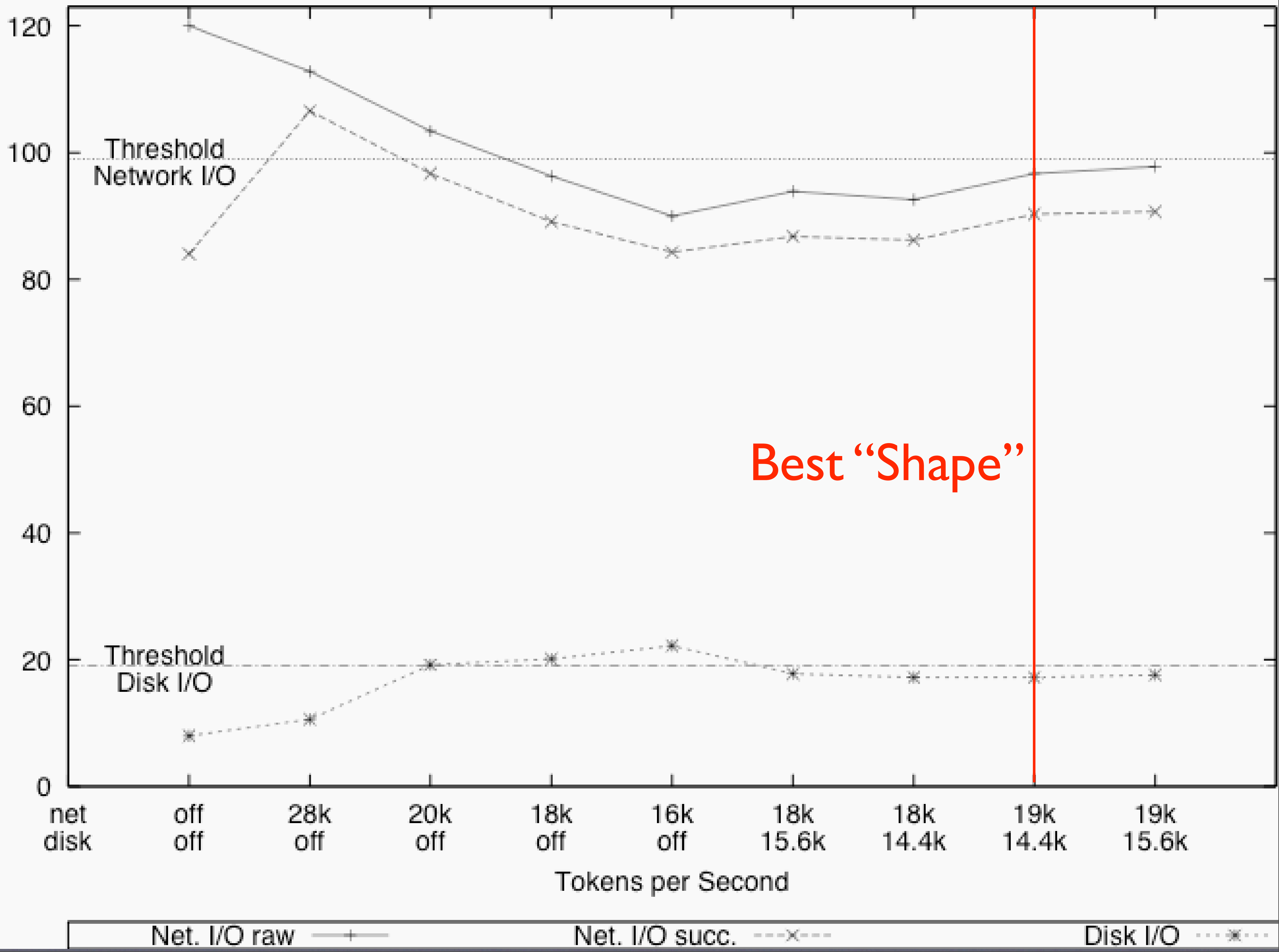
- we run a *video-streaming* server on a *process-shaping* Linux 2.6 server machine with Gigabit Ethernet
- to generate background *network traffic*, we also run one of the web servers of experiment II on the same machine
- to generate background *disk traffic*, we also run several web servers processing requests for a non-cached 1GB file


120
100
80
60
40
20
0
Threshold
Network I/O
Threshold
Disk I/O
net
disk
off
off
28k
off
20k
off
18k
off
16k
off
18k
15.6k
18k
14.4k
19k
14.4k
19k
15.6k
Tokens per Second
Net. I/O raw
Net. I/O succ.
Disk I/O



120
100
80
60
40
20
0
Threshold Network I/O
Threshold Disk I/O
No Shaping
net disk
off off
28k off
20k off
18k off
16k off
18k 15.6k
18k 14.4k
19k 14.4k
19k 15.6k
Tokens per Second
Net. I/O raw
Net. I/O succ.
Disk I/O


120
100
80
60
40
20
0
Threshold Network I/O
Threshold Disk I/O
Just Network Shaping
net disk
off off
28k off
20k off
18k off
16k off
18k 15.6k
18k 14.4k
19k 14.4k
19k 15.6k
Tokens per Second
Net. I/O raw
Net. I/O succ.
Disk I/O


120
100
80
60
40
20
0
Threshold Network I/O
Threshold Disk I/O
Best "Shape"
net disk
off off
28k off
20k off
18k off
16k off
18k 15.6k
18k 14.4k
19k 14.4k
19k 15.6k
Tokens per Second
Net. I/O raw
Net. I/O succ.
Disk I/O

Future

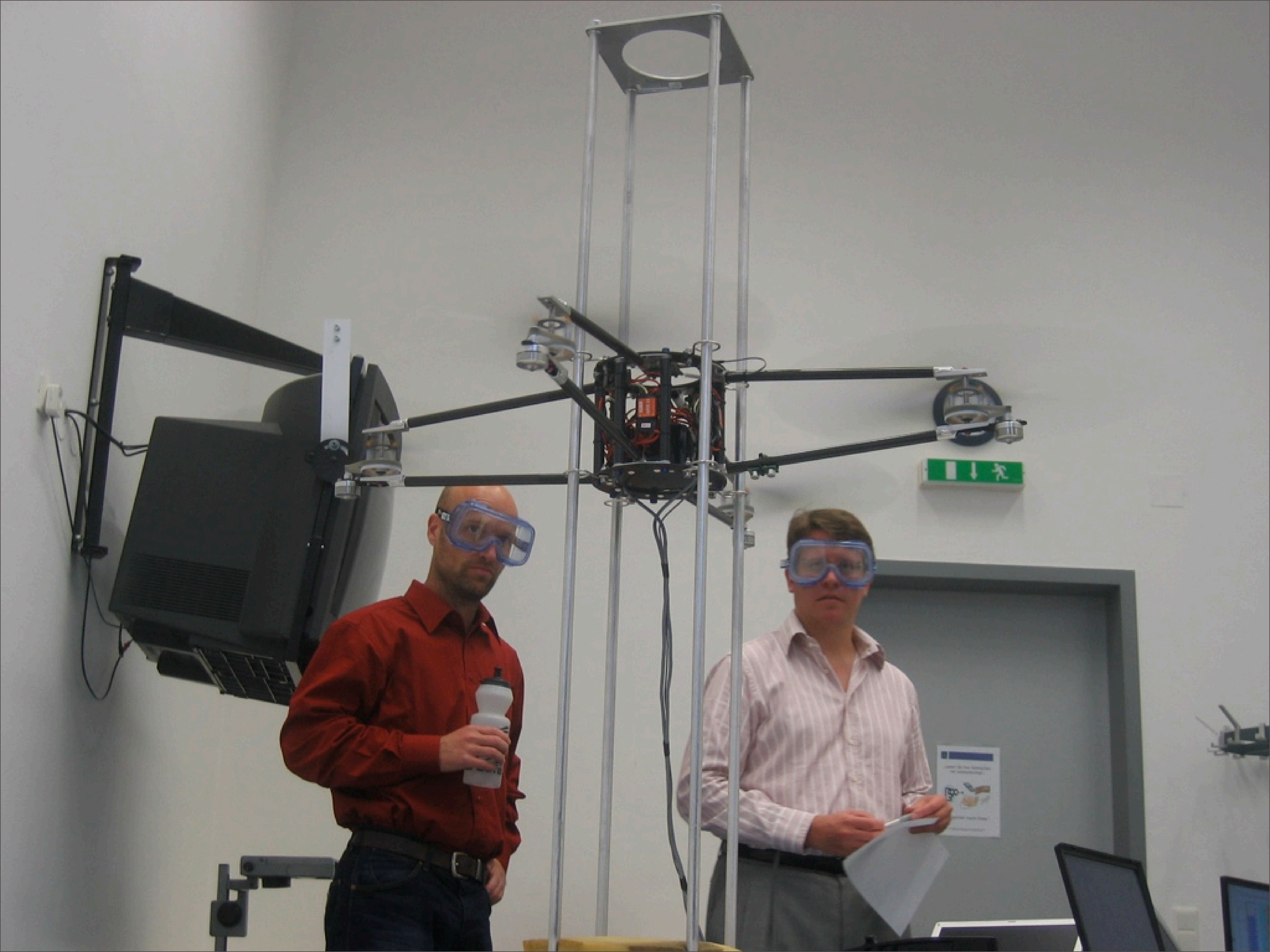
- Multimedia (*Soft* Real Time): Can kernel-level process shaping automatically find the best "shape"?
- Control (*Hard* Real Time): Can kernel-level process shaping provide sufficient real-time guarantees?

Experiment IV

- we run helicopter flight control software written in *Java* on IBM's commercial J9 JVM with the real-time garbage collector *Metronome* on top of a Linux 2.6 machine with real-time patches applied to the kernel
- joint work with J. Auerbach, D. Bacon, H. Röck, and R. Trummer

The JAviator Project

javiator.cs.uni-salzburg.at



First All Java Flight

Thank you