One Server Per City: Using TCP for Very Large SIP Servers

Kumiko Ono
Henning Schulzrinne
{kumiko, hgs}@cs.columbia.edu

Goal

Answer the following question:

How does using TCP affect the scalability and performance of a SIP server?

- Impact on the number of sustainable connections
- Impact of establishing/maintaining connections on data latency
- Impact on request throughput

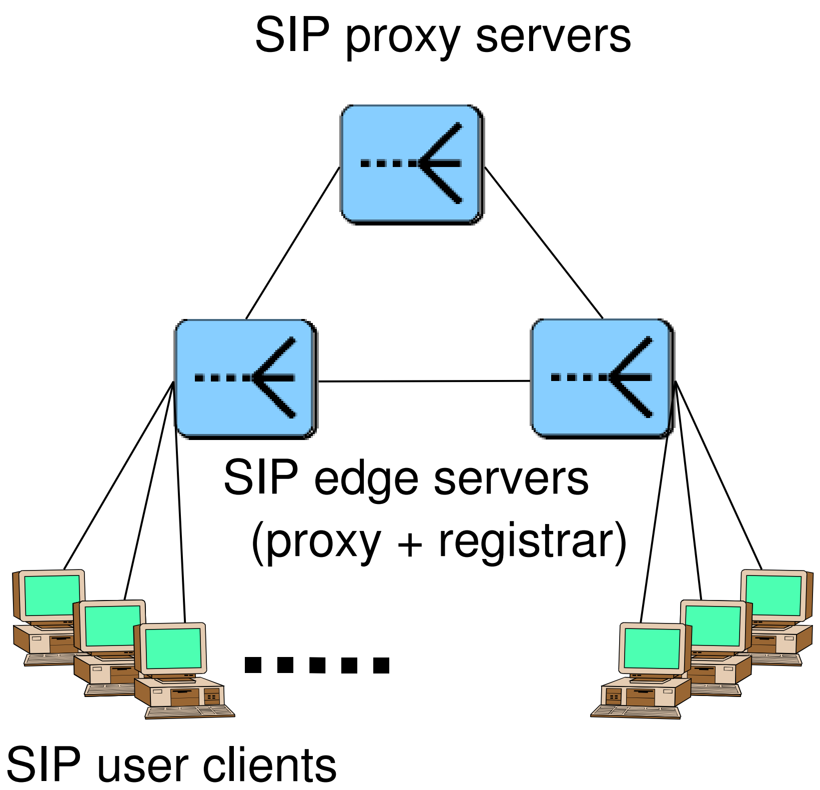
Outline

- Motivation
- Related work
- Measurements on Linux
- Measurement results
  1. Number of sustainable connections
  2. Setup time and transaction time
  3. Sustainable request rate
- Suggestions

Motivation

- A scalable SIP edge server to support 300k users*
  - Handling connections seems costly.
  - Our question:

    How does the choice of TCP affect the scalability of a SIP server?



* Lucent's 5E-XC™, a high capacity 5ESS, supports 250,000 users

SIP server: Proxy and registrar

Comparison with HTTP server

- Signaling (vs. data) bound
  - No File I/O except scripts or logging
  - No caching; DB read and write frequency are comparable
- Transactions and dialogs
  - Stateful waiting for human responses
- Transport protocols
  - UDP, TCP or SCTP

Related work

- A scalable HTTP server
  - I/O system to support 10K clients [1]
    - Use `epoll()` [2] to scale instead of `select()` or `poll()`
    - We built on this work.
  - An architecture for a highly concurrent server
    - Staged Event-Driven Architecture [3]
- A scalable SIP server using UDP
  - Process-pool architecture [4]

[Ref.] Comparison of system calls to wait events

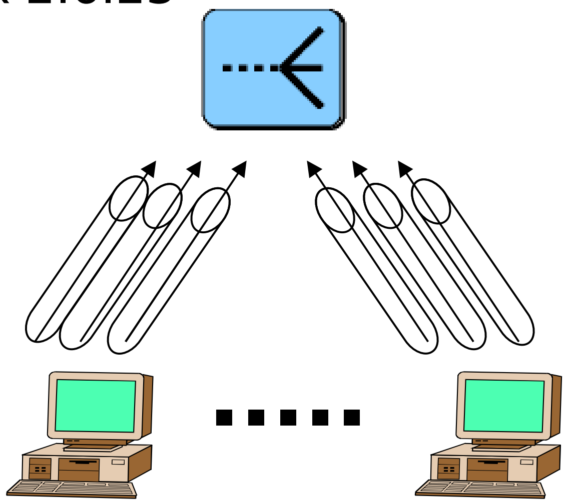
- Upper limit on file descriptor (fd) set size
  - `select()`: 1,024
  - `poll()`, `epoll()`: user can specify
- Polling/retrieving fd set
  - `select()`, `poll()`: the same set both in kernel and user space
    - Events are set corresponding to the prepared fd set.
  - `epoll()`:
    - Different fd set in each by separate I/F
    - Optimal retrieving fd set in user space depending on APL
    - Events are set always from the top of the retrieving fd set

Outline

- Motivation
- Related work
- Measurements on Linux
- Measurement results
  1. Number of sustainable connections
  2. Setup time and transaction time
  3. Sustainable request rate
- Suggestions

Measurement environment

Server:
Pentium IV, 3GHz (dual core),
4GB memory
Linux 2.6.23



55,000 /host

Clients: 8 hosts
Pentium IV, 3GHz,
1GB memory
Redhat Linux 2.6.9

- System configuration
  - Increased the number of file descriptors per shell
    - 1,000,000 at server
    - 60,000 at clients
  - Increased the number of file descriptors per system
    - 1,000,000 at server
  - Expanded the ephemeral port range
    - [10000:65535] at clients

Measurements in two steps

- Using an echo server
  - Number of sustainable connections.
  - Impact of establishing/maintaining connection on the setup and transaction response time
- Using a SIP server
  - Sustainable request rate

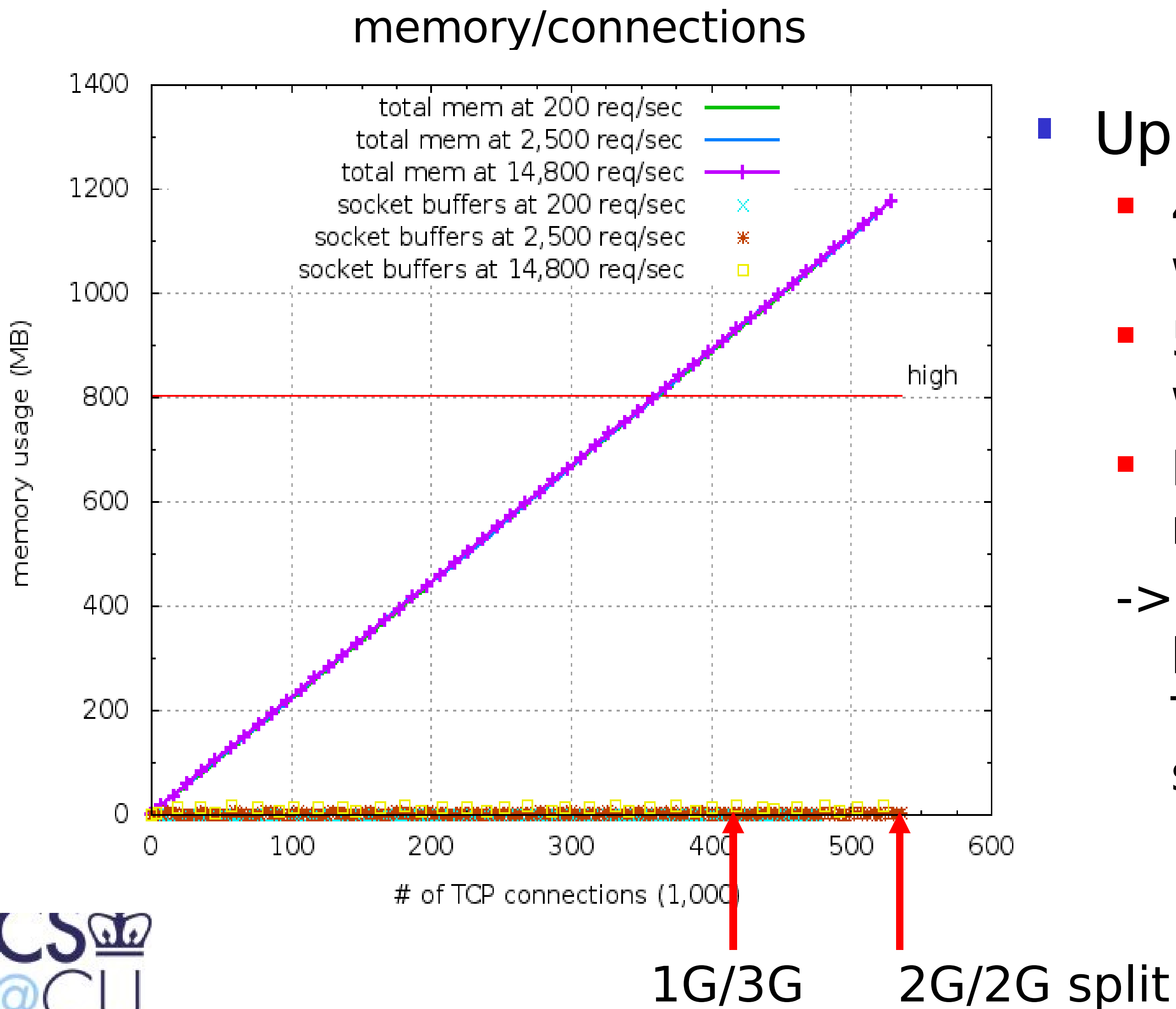
Measurement tools

- Number of sockets/connections
  - /proc/net/sockstat
- Memory usage
  - /proc/meminfo
  - /proc/slabinfo
  - /proc/net/sockstat for TCP socket buffers
  - `free` command for the system
  - `top` command for RSS and VMZ per process
- CPU usage
  - `top` command
- Setup and transaction times
  - timestamps added at the client program
  - `tcpdump` program

Outline

- Motivation
- Related work
- Measurements on Linux
- Measurement results
  - Number of sustainable connections
  - Setup time and transaction time
  - Sustainable request rate
- Suggestions

Echo server measurement: Number of sustainable connections for TCP



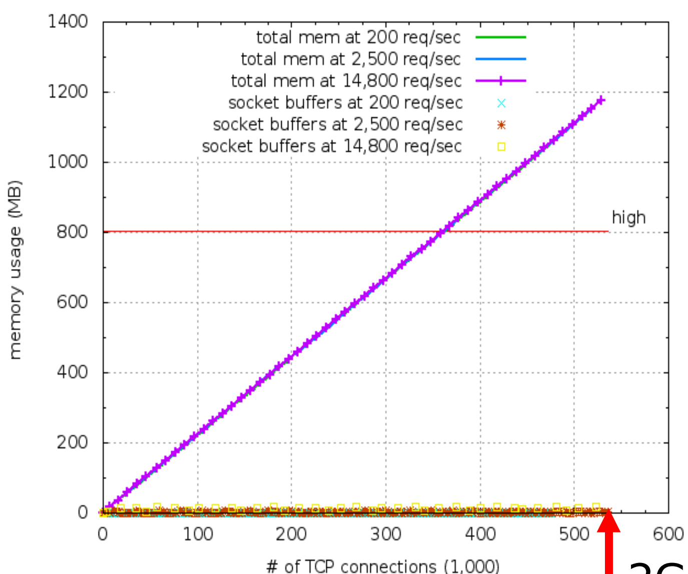
- Upper limit
  - 419,000 connections with 1G/3G split
  - 520,000 connections with 2G/2G split
  - Ends by out-of-memory

-> The bottleneck is kernel memory for TCP sockets, not for socket buffers.

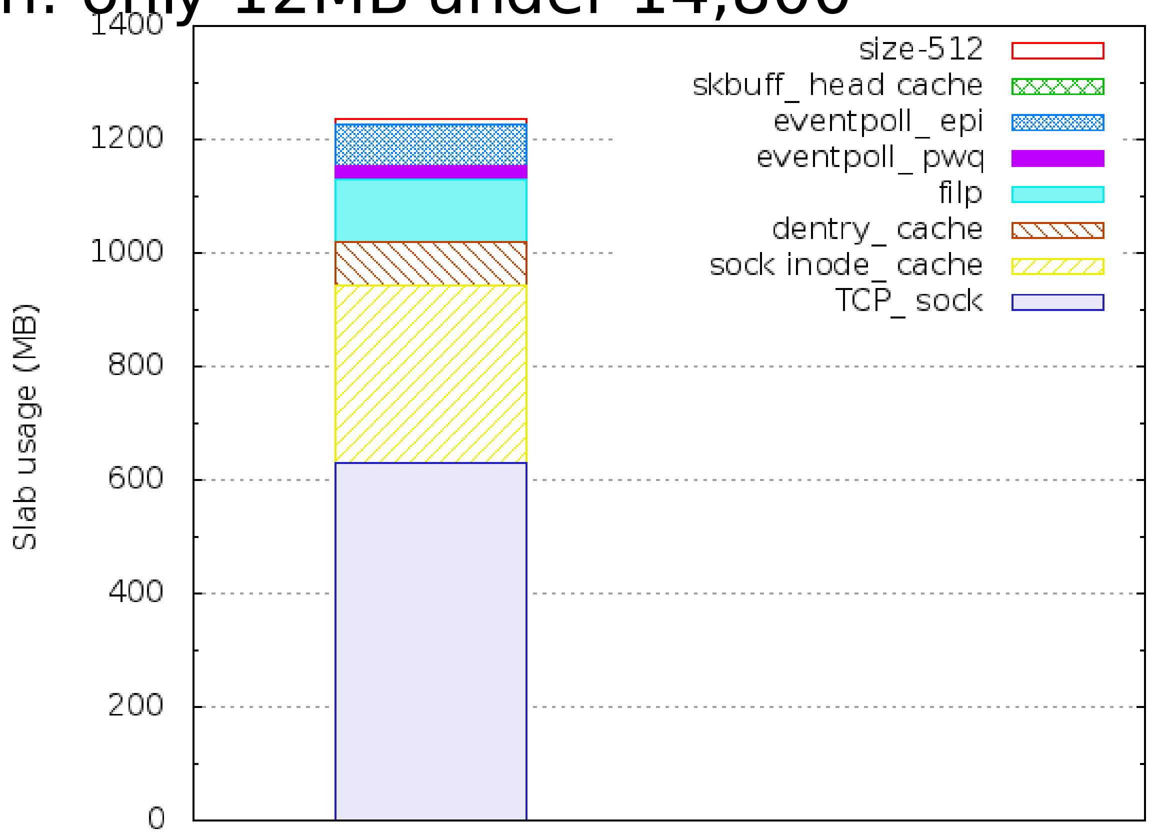
Echo server measurement: Slab cache usage for TCP

- Static allocation: 2.3 KB slab cache per TCP connection
- Dynamic allocation: only 12MB under 14,800 requests/sec. rate

memory/connections



2G/2G split



Slab cache usage for 520k TCP connections

Summary: Number of sustainable connections

- 419,000 connections w/default VM split
- 2.3 KB of kernel memory/connection
- Bottleneck
  - Kernel memory space
  - More physical does not help for a 32-bit kernel. Switch to a 64-bit kernel.

Outline

- Motivation
- Related work
- Measurements on Linux
- Measurement results
  - Number of sustainable connections
  - Setup time and transaction time
  - Sustainable request rate
- Suggestions

Echo server measurement: Setup and transaction times

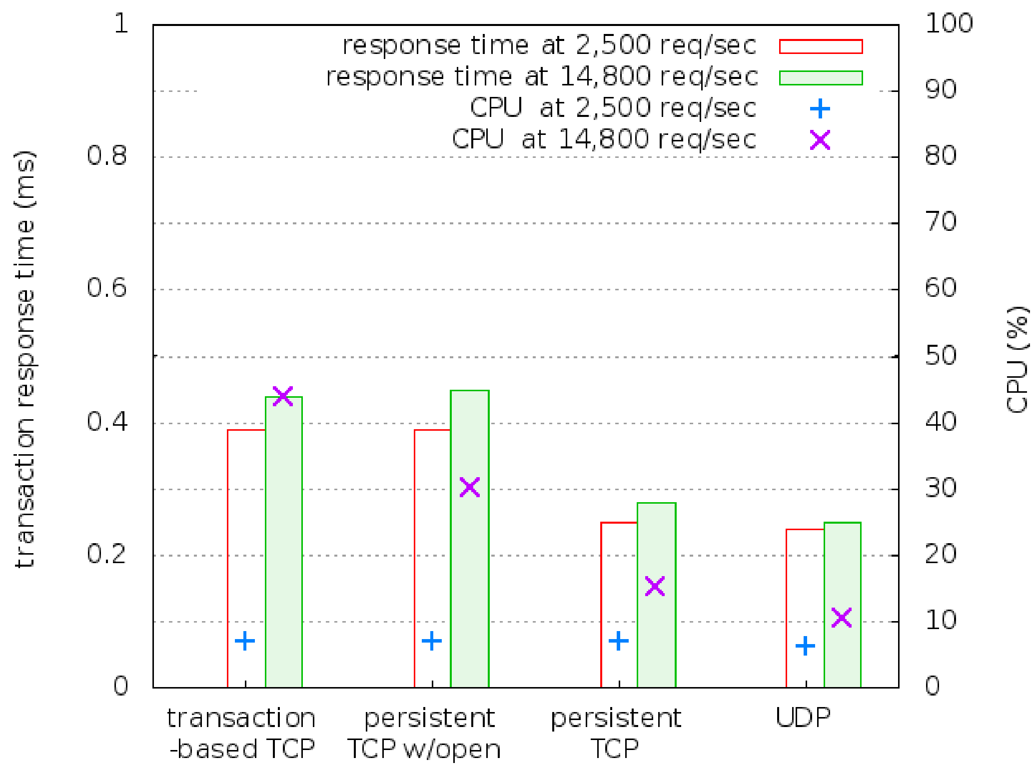
- Objectives:
  - Impact of establishing a connection
    - Setup delay
    - Additional CPU time
  - Impact of maintaining a huge number of connections
    - Memory footprint in kernel space
    - Setup and transaction delay?

Echo server measurement scenarios: Setup and transaction times

- Test sequences
  - Transaction-based
  - Persistent w/ TCP-open
  - Persistent (reuse connection)
- Traffic conditions
  - 512 byte message
  - Sending request rate
    - 2,500 requests/second
    - 14,800 requests/second
- Server configuration
  - No delay option

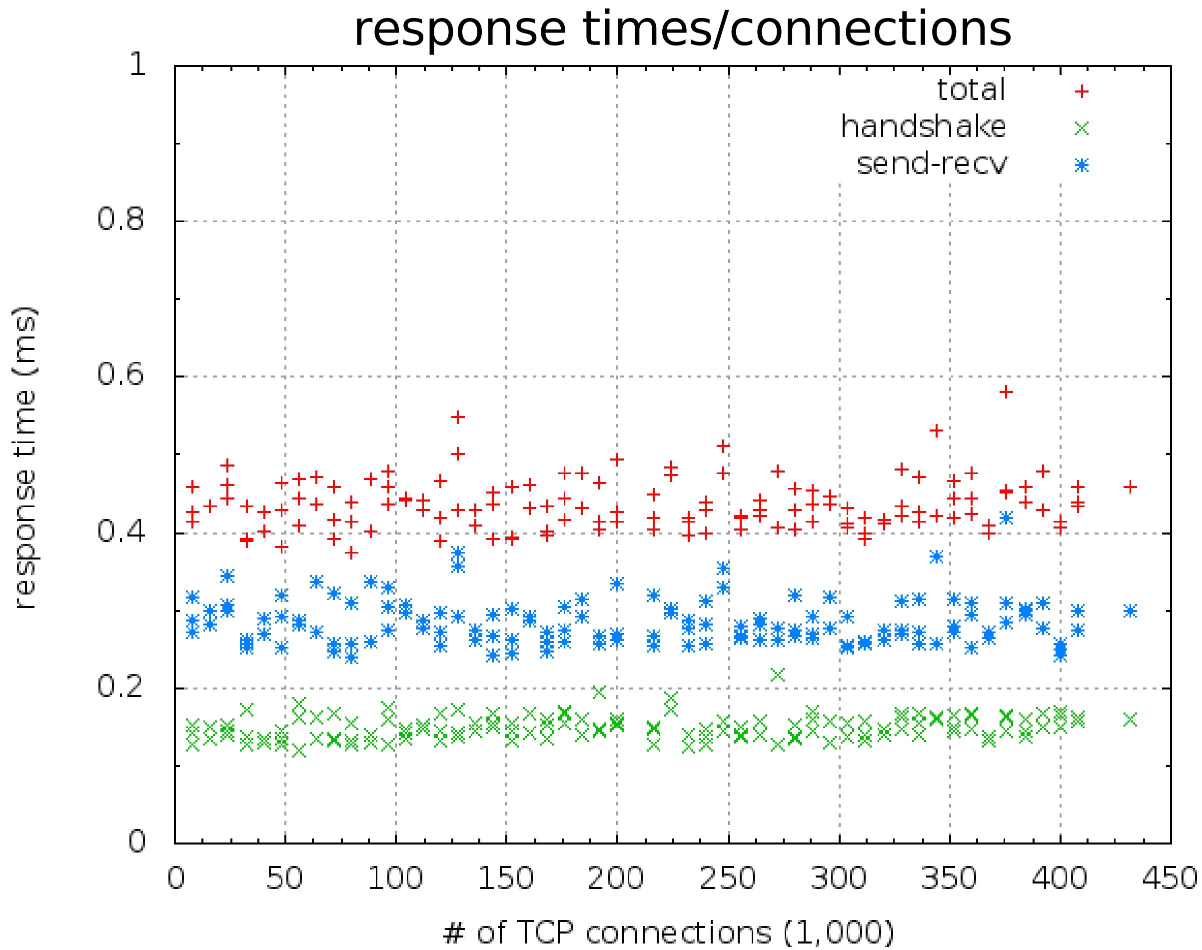
Echo server measurement: Impact of establishing TCP connections

- CPU time:
  - 15% more under high loads, while no difference under mid loads
- Response time
  - Setup delay of 0.2 ms. in our environment
  - Similar time for Persistent TCP to that for UDP



Echo server measurement: Impact of maintaining TCP connections

- Remains constant independently of the number of connections



Summary: Impact on setup and transaction times

- Impact of establishing a connection
  - Setup delay
    - 0.2 ms in our measurement
  - Additional CPU time
    - No cost at low request rate
    - 15% at high request rate
- Impact of maintaining a huge number of connections
  - Memory footprint in kernel space
  - Setup and transaction delay
    - No significant impact for TCP
    - Persistent TCP has a similar response time to UDP.

Outline

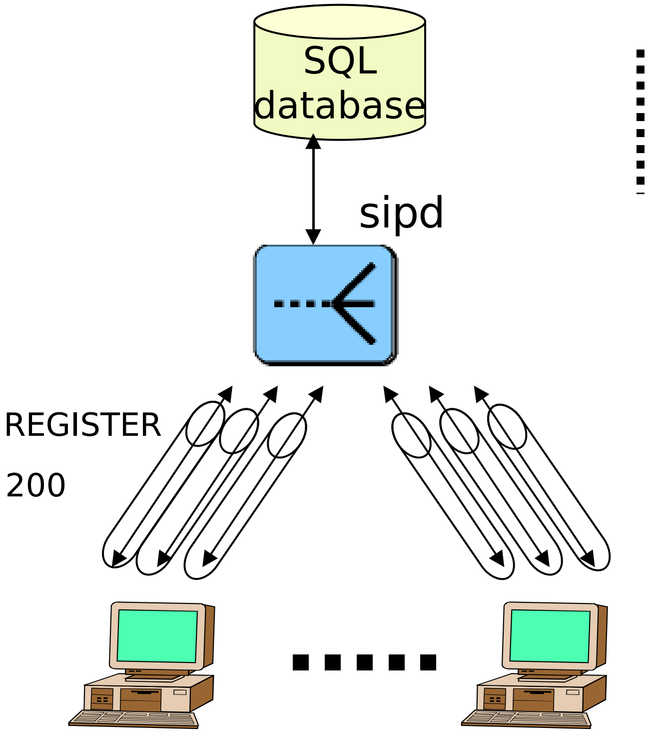
- Motivation
- Related work
- Measurements on Linux
- Measurement results
  - Number of sustainable connections/associations
  - Setup time and transaction time
  - Sustainable request rate
- Suggestions

Measurements in two steps

- Echo server for simplicity
  - Number of sustainable connections
  - Impact of establishing/maintaining connection on the setup and transaction response time
- SIP server
  - Sustainable request rate
  - (Impact of establishing/maintaining connection on the setup and transaction response time)

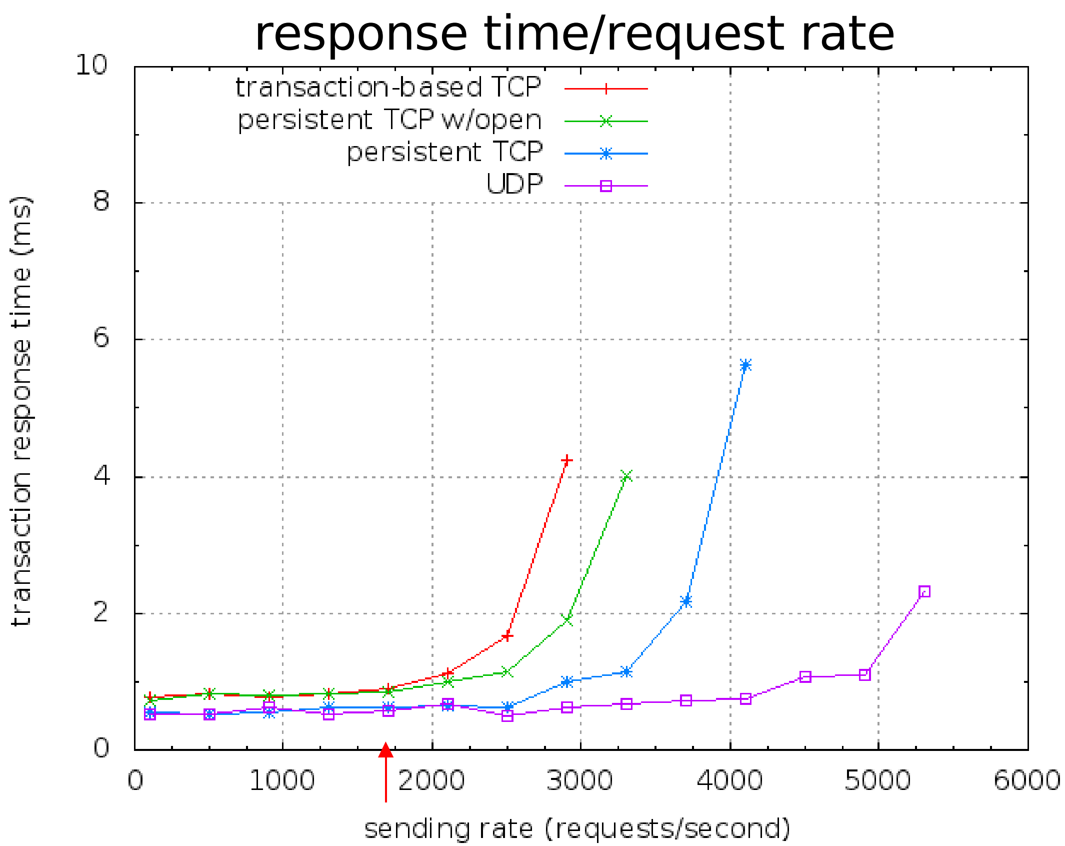
SIP server measurement: The environment

- SUT
  - SIP server: sipd
    - registrar and proxy
    - Transaction stateful
    - Thread-pool model
  - the same host as the echo server
- Clients
  - sipstone
  - Registration:
    - TCP connection lifetime
      - Transaction
      - Persistent w/open
      - Persistent
  - 8 hosts of the echo clients



SIP server measurement: Sustainable req. rate for registration

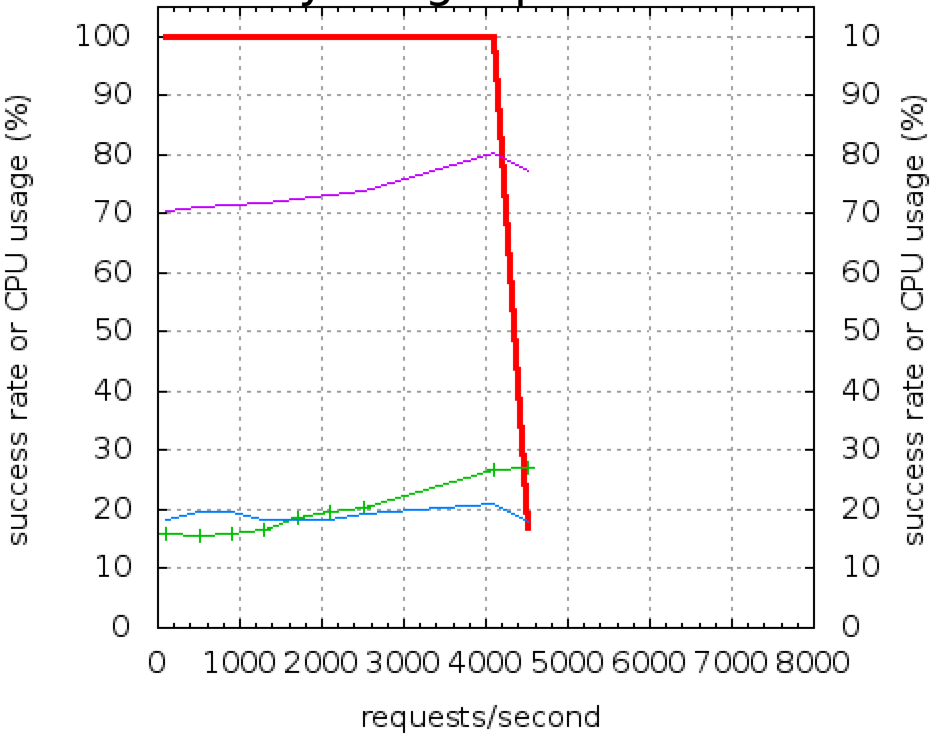
- The less number of messages delivered to application, the more sustainable request rate.
  - Better for UDP, although persistent TCP has the same number of messages with UDP



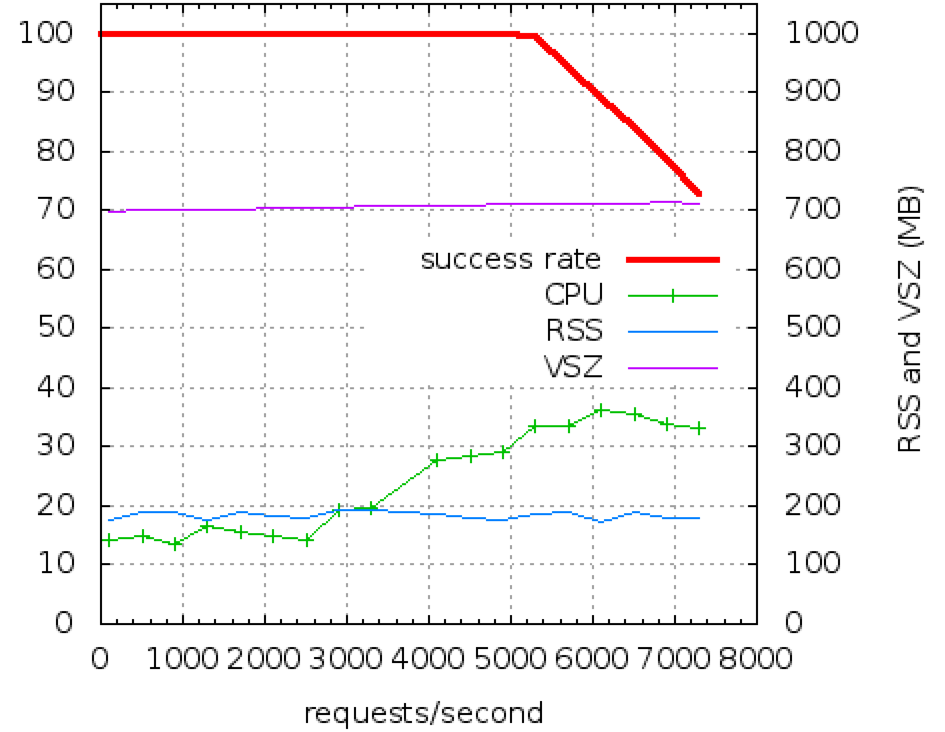
What is the bottleneck of sustainable request rate ?

- No bottleneck in CPU time and memory usage
- Graceful failure by the overload control for UDP, not for TCP

Success rate, CPU time and memory usage: persistent TCP

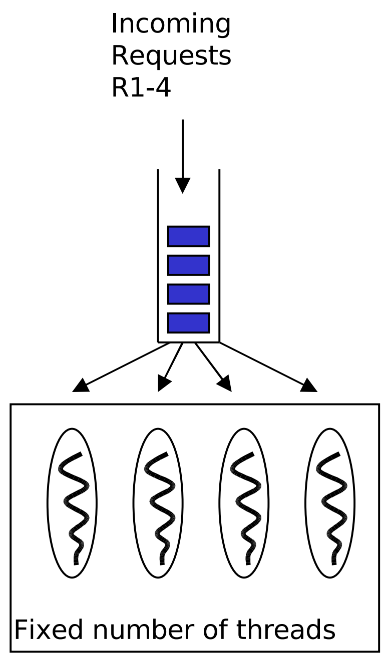


Success rate, CPU time and memory usage: UDP



Software architecture of sipd: Overload control in thread-pool model

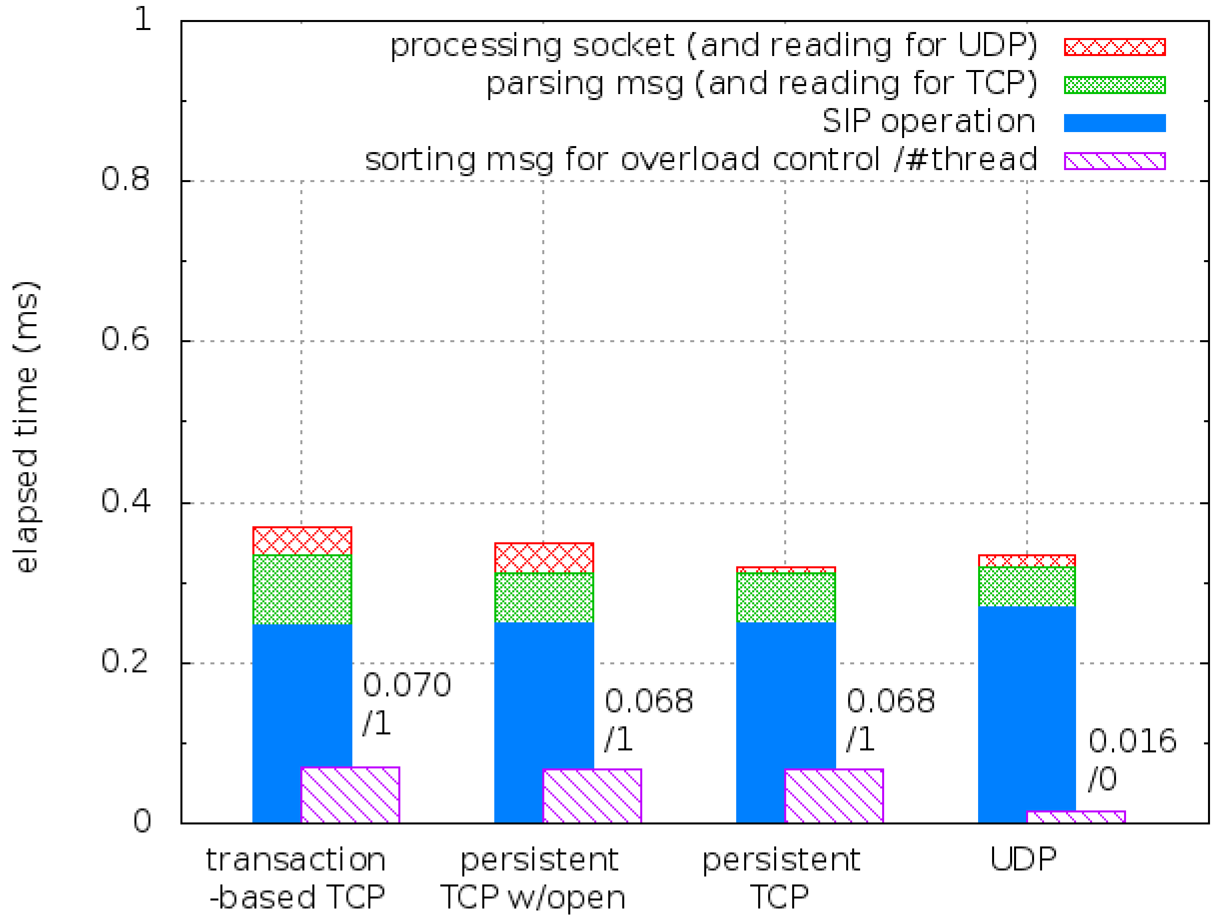
- Overload detection by the number of waiting tasks for thread allocation
- Sorting and favoring specific messages
  - Response over requests
  - BYE requests



- Sorting messages is easier for UDP than TCP
  - Message-oriented protocol enables to parse only the first line.
  - Byte-stream protocol requires to parse Content-Length header to find the first line.

Component test: Message processing test

- Longer elapsed time for reading and parsing REGISTER message using TCP than that for UDP



Suggestions

- Accelerate parsing message for sorting
  - By reading the first-line of buffered message without determining the exact message boundary
    - Not 100% accurate, but works mostly at edge server
- Perform overload control at the base thread in thread-pool model
  - No need to wait for another thread
- Use persistent connections as HTTP/1.1

Conclusions

- Impact of using TCP on a SIP server
  - Scalable well
  - Memory footprint
    - 2.3 KB/connection in kernel memory
  - Setup delay
    - Better to use persistent connections
  - Parsing messages
    - Need to accelerate for overload control

References

[1] D. Kegel. The C10K problem. http://www.kegel.com/c10k.html.

[2] D. Libenzi. Improving (network) I/O performance. http://www.xmailserver.org/linux-patches/nio-improve.html.

[3] M.Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, October 2001.

[4] K. Singh and H. Schulzrinne. Failover and Load Sharing in SIP Telephony. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2005.

Thank you!

Any questions?

mailto: kumiko@cs.columbia.edu