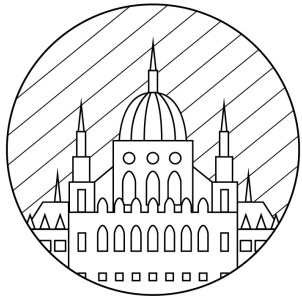


# Choosing Tomcat Connectors

Jean-Frederic Clere



APACHECON  
EUROPE

CORINTHIA HOTEL  
BUDAPEST, HUNGARY

— NOVEMBER 17-21, 2014 —



# What I will cover

- Who I am.
- Connectors
  - JIO, NIO, NIO2, APR
  - AJP/HTTP
  - Proxy AJP/HTTP
- Performances tests
  - With ab or customized client load generator.
- Questions?



# Who I am

Jean-Frederic Clere

Red Hat

Responsible of JWS product.

Years writing JAVA code and server software

Tomcat committer since 2001

Doing OpenSource since 1999

Cyclist/Runner etc

Lived 15 years in Spain (Barcelona)

Now in Neuchâtel (CH)



# Remote location







# Red Hat Office Neuchâtel





# Protocol basic

- HTTP/1.1 request
- Responses:
  - Normal
  - Chunked
  - Upgrade (to websocket for example)
  - Proxy AJP/HTTP



# Request HTTP/1.1

POST /comet/CometServletTest1 HTTP/1.1\n

User-Agent: testclient\n

Host: localhost\n

Transfer-Encoding: chunked\n



# Response for example

## Chunked:

HTTP/1.1 200 OK\n

Server: Apache-Coyote/1.1\n

Set-Cookie: JSESSIONID=obcoR30qlz7DMJfZmsVTt+Uv; Path=/comet\n

Transfer-Encoding: chunked\n

Date: Mon, 07 Nov 2011 22:09:33 GMT\n

## Upgrade:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMIYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat





# What is a Connector?

- Tomcat's interface to the world
- Binds to a port
- Understands a protocol
- Dispatches requests
  - `protocol="org.apache.coyote.http11.Http11Protocol"`
  - `protocol="org.apache.coyote.http11.Http11AprProtocol"`
  - `protocol="org.apache.coyote.http11.Http11NioProtocol"`
  - `protocol="org.apache.coyote.http11.Http11Nio2Protocol"`



# Tomcat Connectors

- Java Blocking I/O (BIO or sometimes JIO)
- Java Non-blocking I/O (NIO)
- Native / Apache Portable Runtime (APR)
- Java NIO.2

Technically, there are combinations of all of the above with HTTP and AJP protocols.

We'll discuss those a bit later.



# Types of I/O

- Polling
  - Straightforward API: `peek()`
  - CPU-inefficient
  - Thread loops while waiting for data
- Blocking
  - Straightforward API: `read()`
  - CPU-efficient (blocks)
  - Thread stalls while waiting for data



# Types of I/O

- Non-blocking
  - Complicated API (registration, event callbacks)
    - Channel
    - Buffer
    - Selector
  - CPU-efficient
  - Thread not required to: execution continues
  - When data is ready, the selector notifies observers



# Common Connector Features

- Support for all protocols
  - HTTP, AJP, WebSocket
- Support for all dispatch methods
- Standard, Comet, Servlet 3.0 async
  - Support for HTTPS (SSL/TLS)
- Acceptor thread(s) call `accept()` and hand-off
- Request processor thread pool





# Blocking I/O Connector (1)

- All I/O operations are blocking in processor thread
  - SSL handshake
  - Read request line (e.g. GET, POST, etc.)
  - Read request body
  - Write response
  - Read next request (HTTP keep-alive)
- Simple, stable, mature



# Blocking I/O Connector (2)

- Single thread handles request straight-through, after accept
- Uses Java Secure Sockets Extension (JSSE) for SSL/TLS



# Blocking I/O Connector (3)

- Request throughput limited by thread count
- Clients can waste threads
  - Slow request line (mobile)
  - Aborted keep-alive stalls thread (default=20sec!)
- Unfair: accepted connections get priority for keep-alive requests



# NON-Blocking I/O Connector (1)

- Single thread handles request after request-line
- Poller thread(s) manage non-blocking Selector
  - Read SSL handshake
  - Read request line
  - Wait for next keep-alive request



# NON-Blocking I/O Connector (2)

- Block poller simulates blocking
  - Request header/body reads
  - Response writes
  - Processor thread sleeps during sim-blocking
- Uses JSSE for SSL/TLS
- Supports sendFile





# NON-Blocking I/O Connector (3)

- Allows huge number of parallel requests
  - Not limited by request-processor threads
- Slow clients do not stall threads
- Aborted keep-alives die in the poller queue
- Simulated blocking adds overhead



# Native Connector (APR) (1)

- Single thread handles request after accept()
- Poller thread(s) handle certain I/O reads
  - Wait for next keep-alive request
- Some I/O operations block processor thread
  - SSL handshake
  - Read request line
  - Read request body
  - Write response



# Native Connector (APR) (2)

- Uses OpenSSL for SSL/TLS
- Supports sendFile



# Native Connector (APR) (3)

- Request throughput limited by thread count
- Slow clients can stall threads
- Aborted keep-alives die in the poller queue
- OpenSSL offers performance advantage
- Native code risks JVM instability



# “Non-blocking” I/O Connector NIO.2 (1)

- Single thread handles request after request-line
- The thread are handled via an `AsynchronousChannelGroup` and completion call backs
  - Read SSL handshake
  - Read request line and headers
  - Wait for next keep-alive request





# “Non-blocking” I/O Connector NIO.2 (2)

- NIO2 implementation takes care of blocking using Future objects waiting for IO
  - Request body reads
  - Response writes
  - Processor thread sleeps during blocking
- Uses JSSE for SSL/TLS
- It emulates sendFile (NIO1 transferTo doesn't work with NIO2)



# “Non-blocking” I/O Connector NIO.2 (3)

- Allows huge number of parallel requests
  - Not limited by request-processor threads
- Slow clients do not stall threads
- High level of abstraction and blocking over async adds overhead
- NIO 2 provides blocking capabilities over its async IO. The tomcat code is simpler (good) but an overhead still exists.



# Technical constraints

- Don't try bother using non-blocking protocols with blocking connectors (BIO+Websocket = bad)
- AJP can be thought of as 100% keep-alive
- AJP doesn't support HTTP upgrade
- Use of sendFile is highly recommended for any static-content (APR or NIO.1)

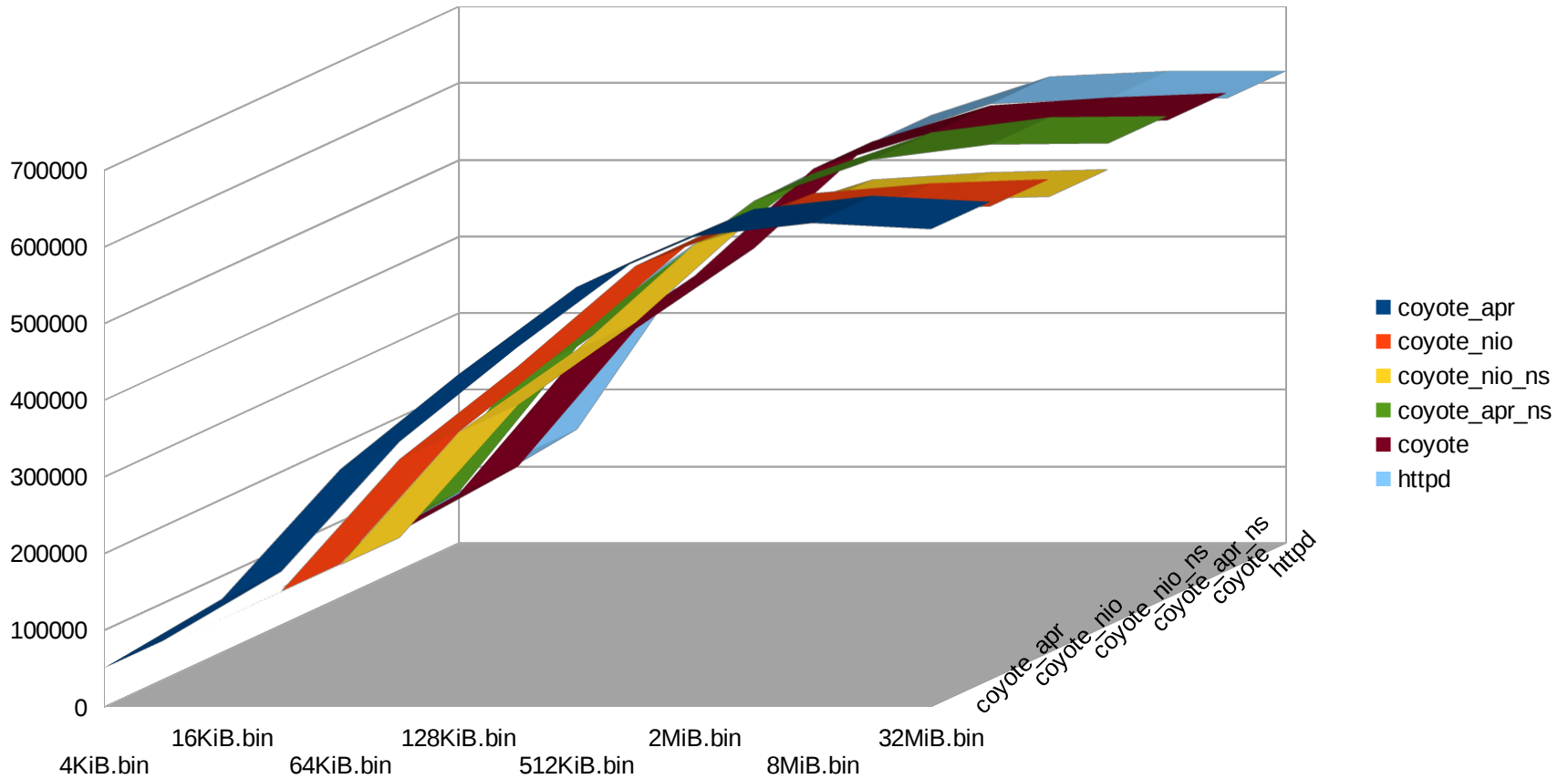


# Connector Performance

- Compare connector throughput against each other
- Only static content was compared, varying file sizes
- Run on fast machines, 10 Gbps local network
- Tests:
  - Compare the connectors (tc8.0.14) with httpd (2.2.22) no SSL.
  - Same with SSL
  - What about using a proxy: compare proxies.



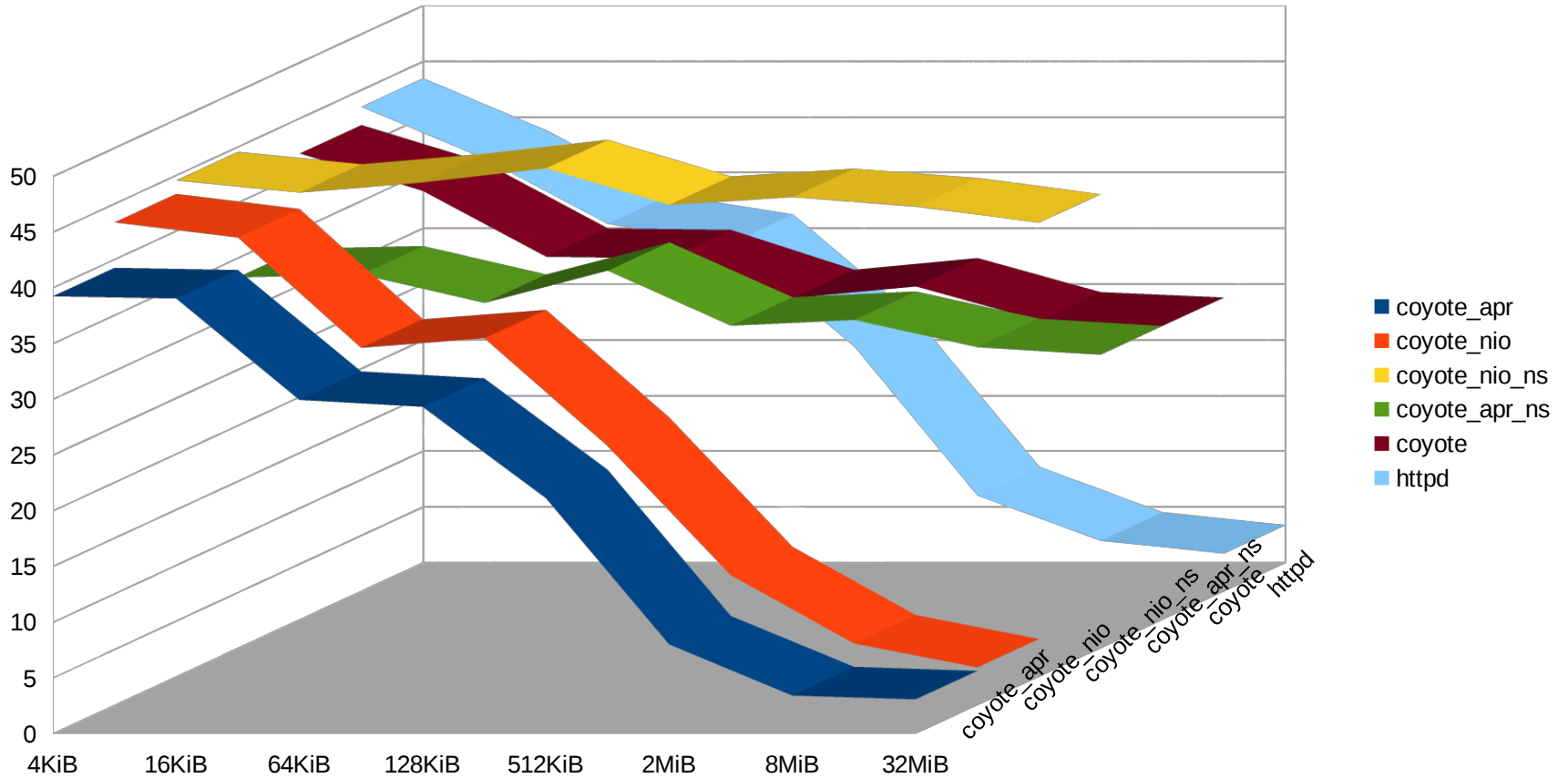
# Connector Throughput (c4)





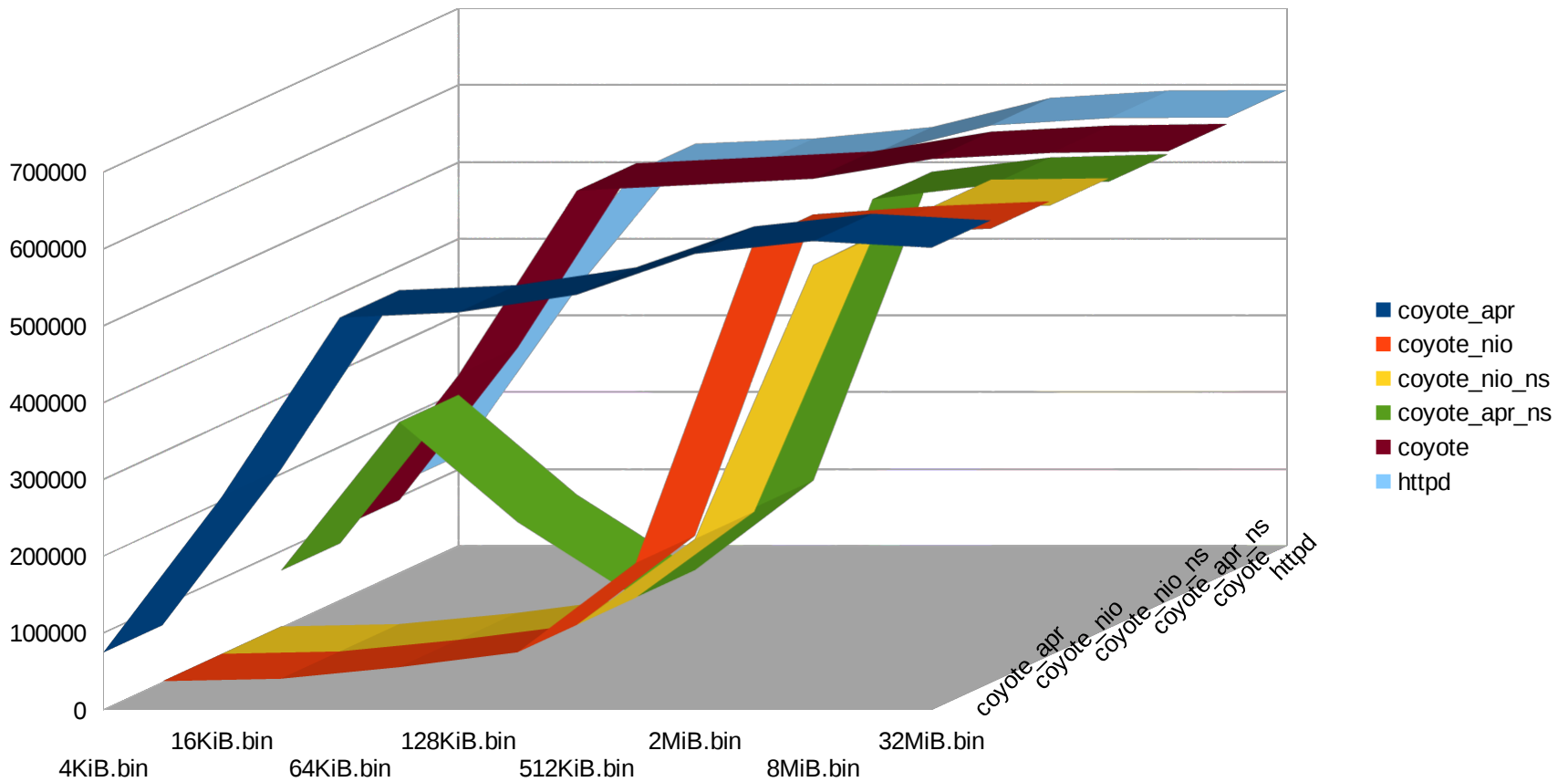


# Connector CPU Use (c4)



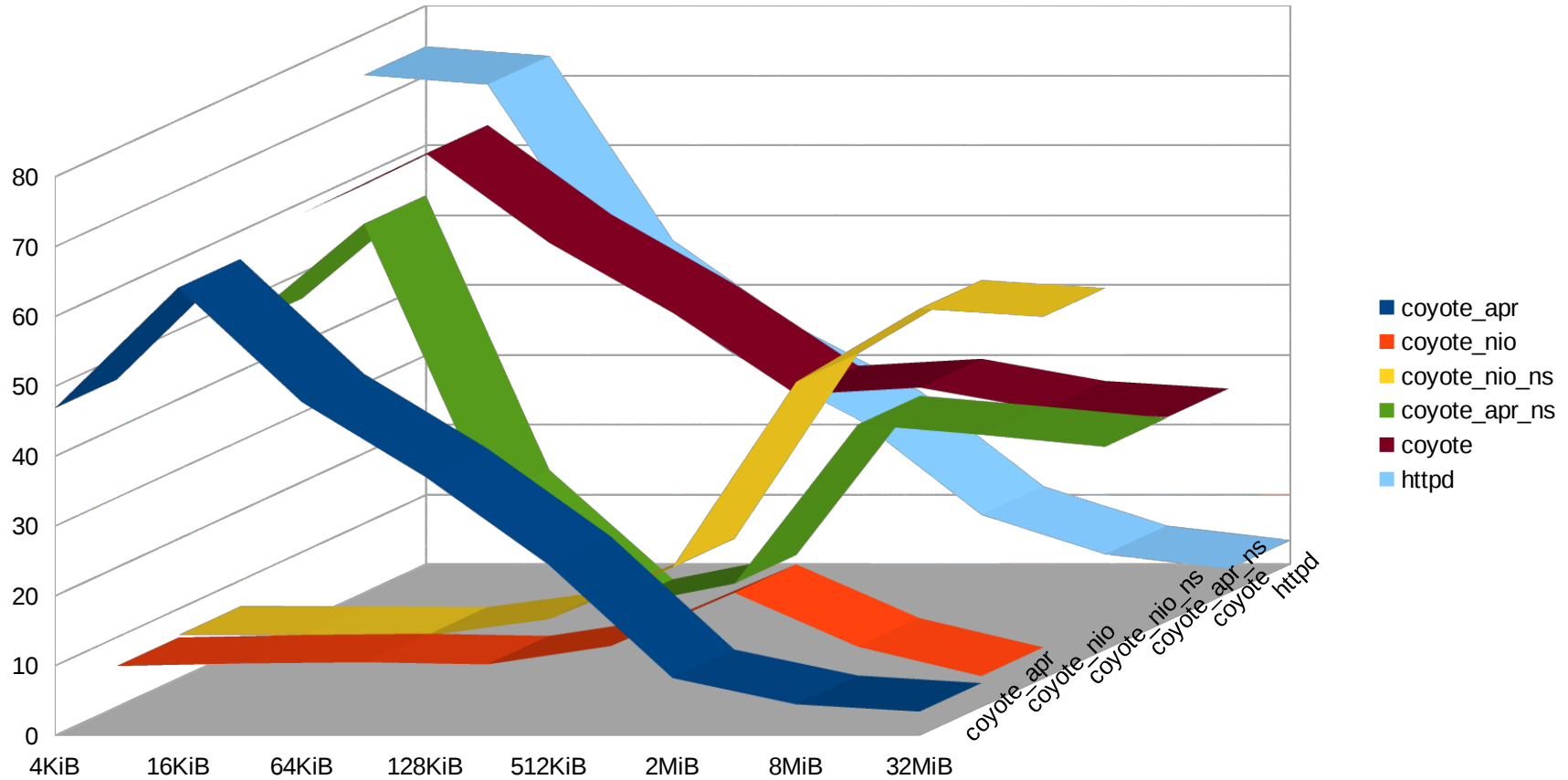


# Connector Throughput (c40)



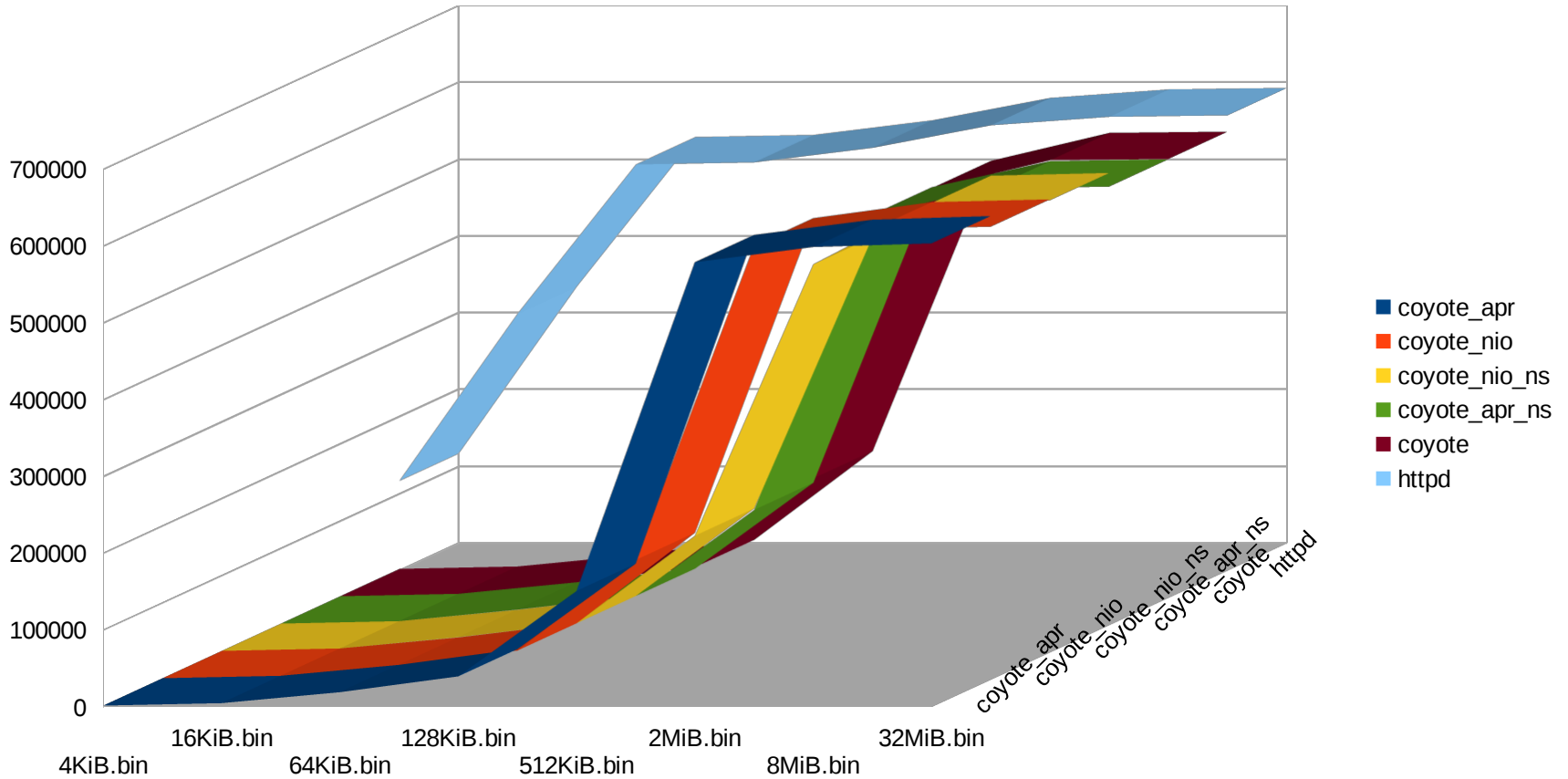


# Connector CPU Use (c40)



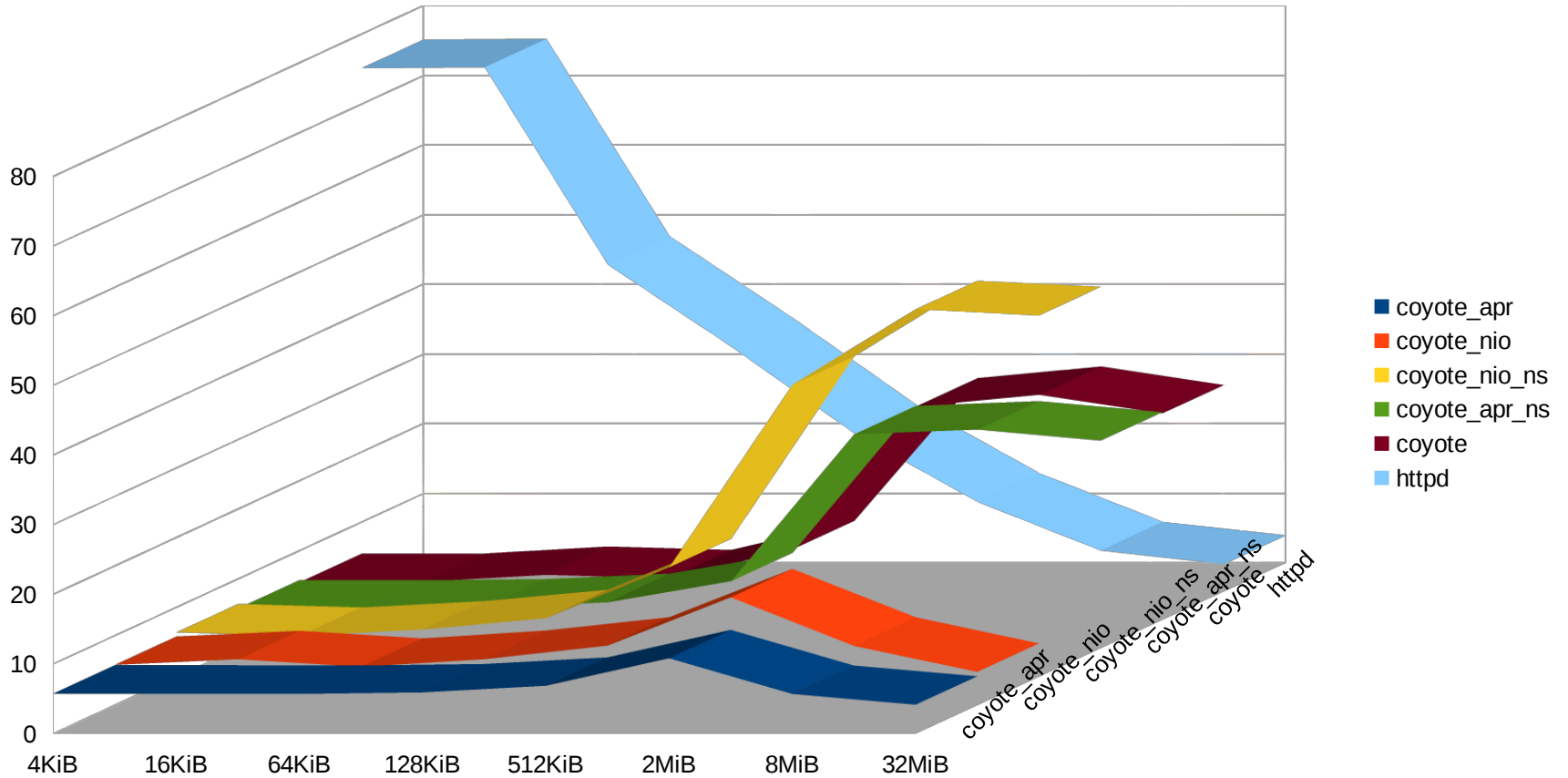


# Connector Throughput (c80)





# Connector CPU Use (c80)



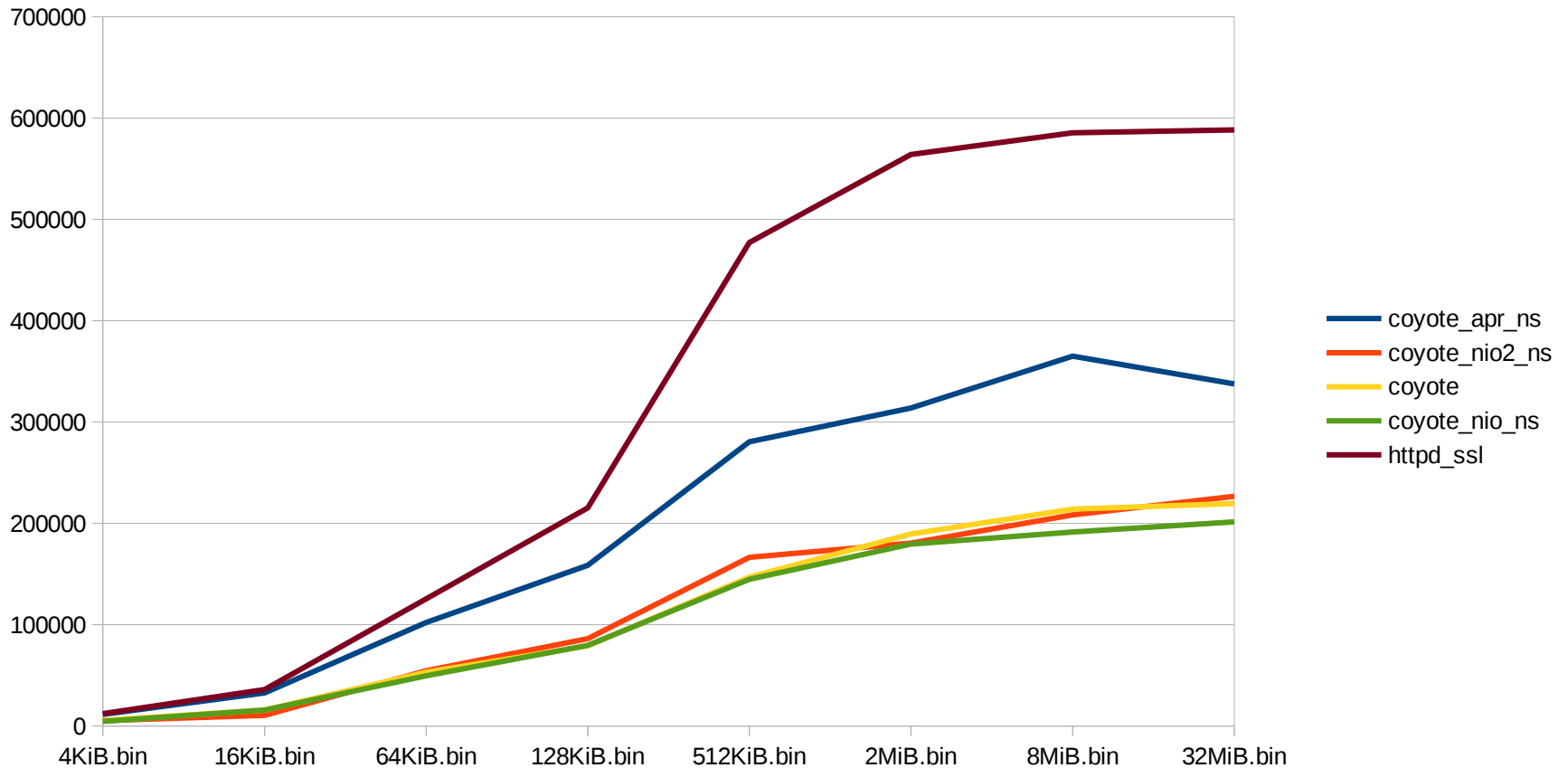


# Connector Performance

- Intermediate conclusion:
  - Using sendfile helps a little. (but just emulated in NIO2!)
  - So using JIO/BIO is probably an “old” idea.

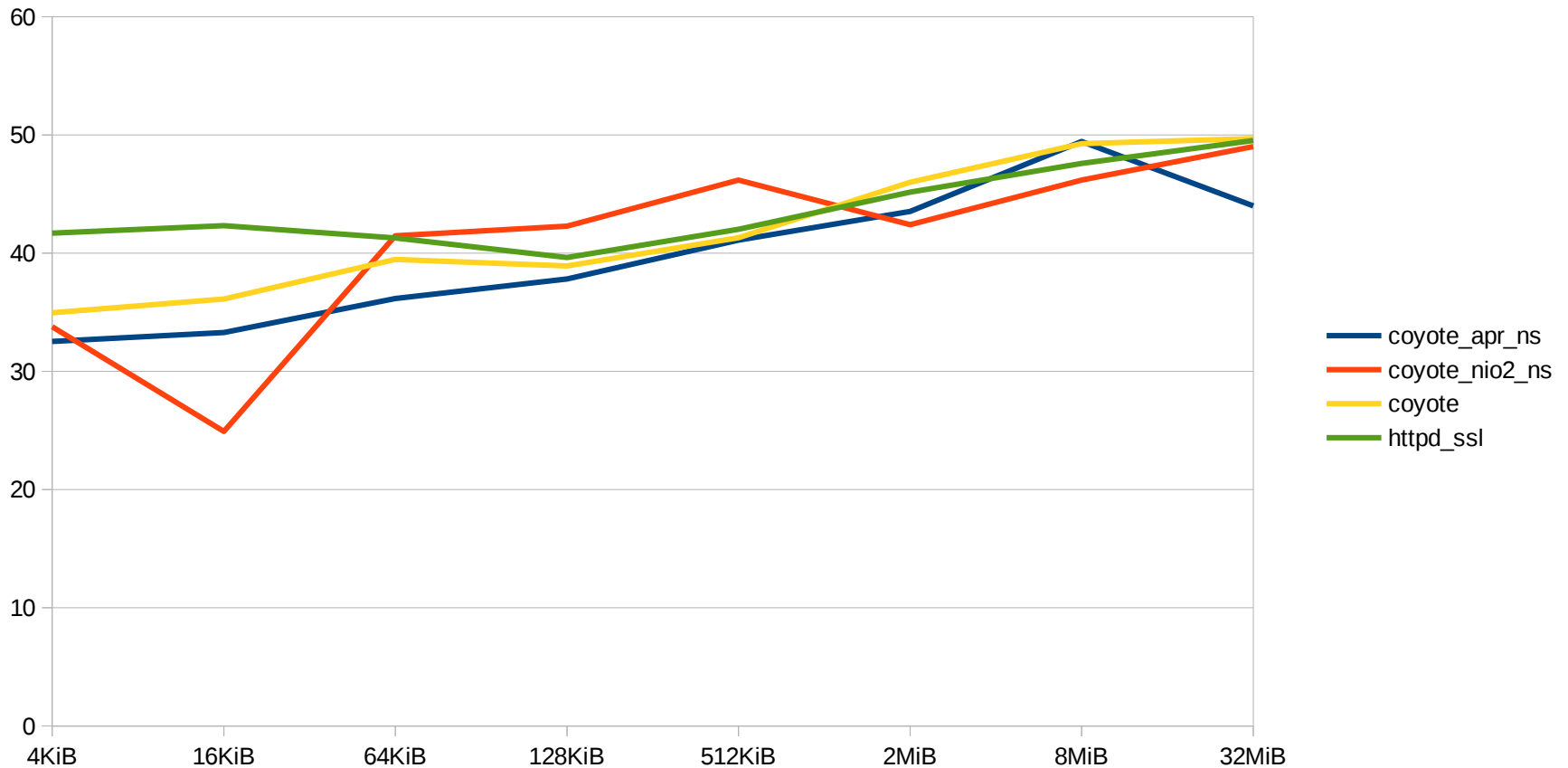


# SSL Connector Throughput (c4)





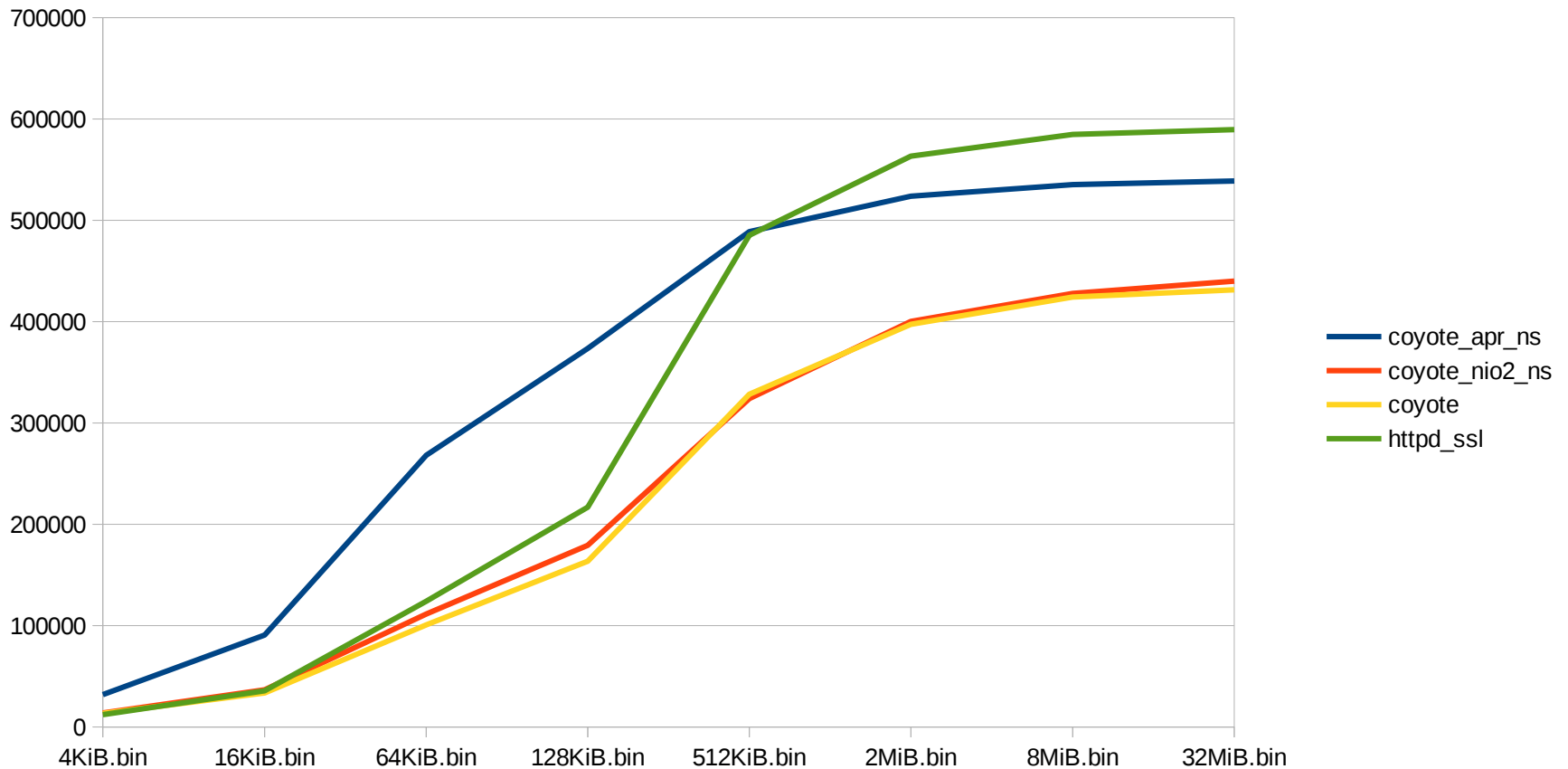
# SSL Connector CPU Use (c4)





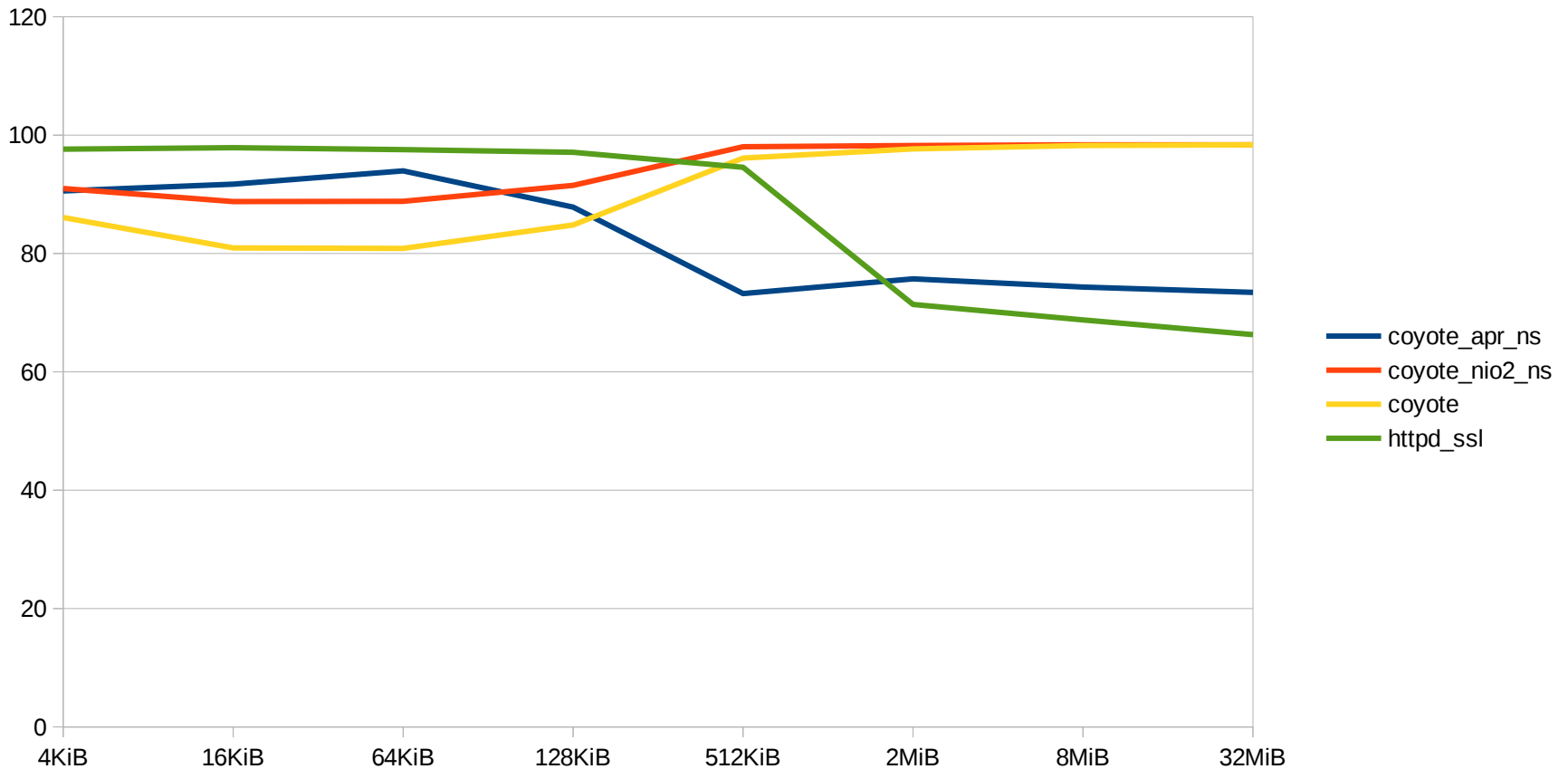


# SSL Connector Throughput (c40)





# SSL Connector CPU Use (c40)



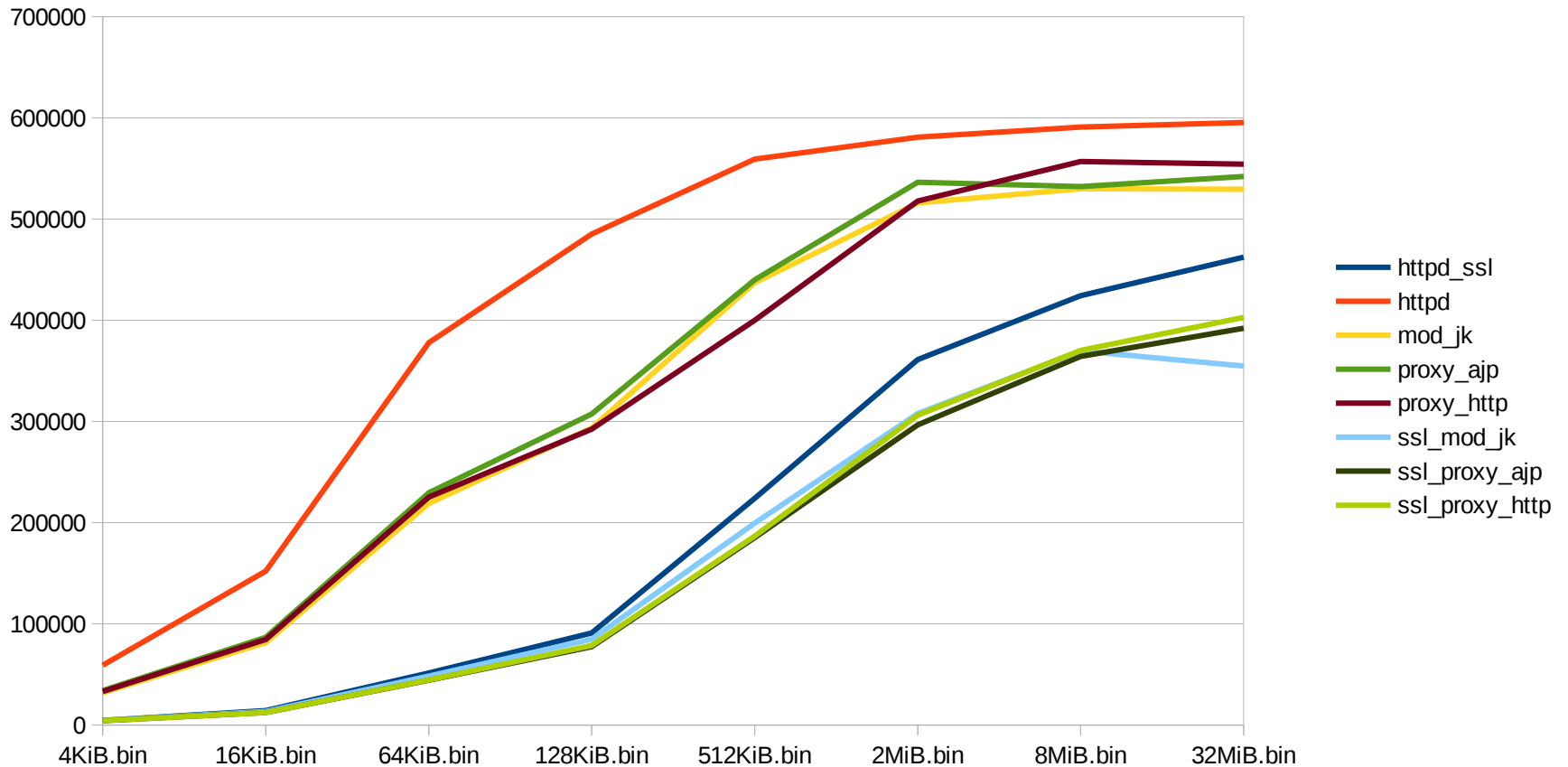


# Connector Performance

- Intermediate conclusion:
  - OpenSSL performs better than JSSE
  - JIO/BIO and NIO(2) give similar results.

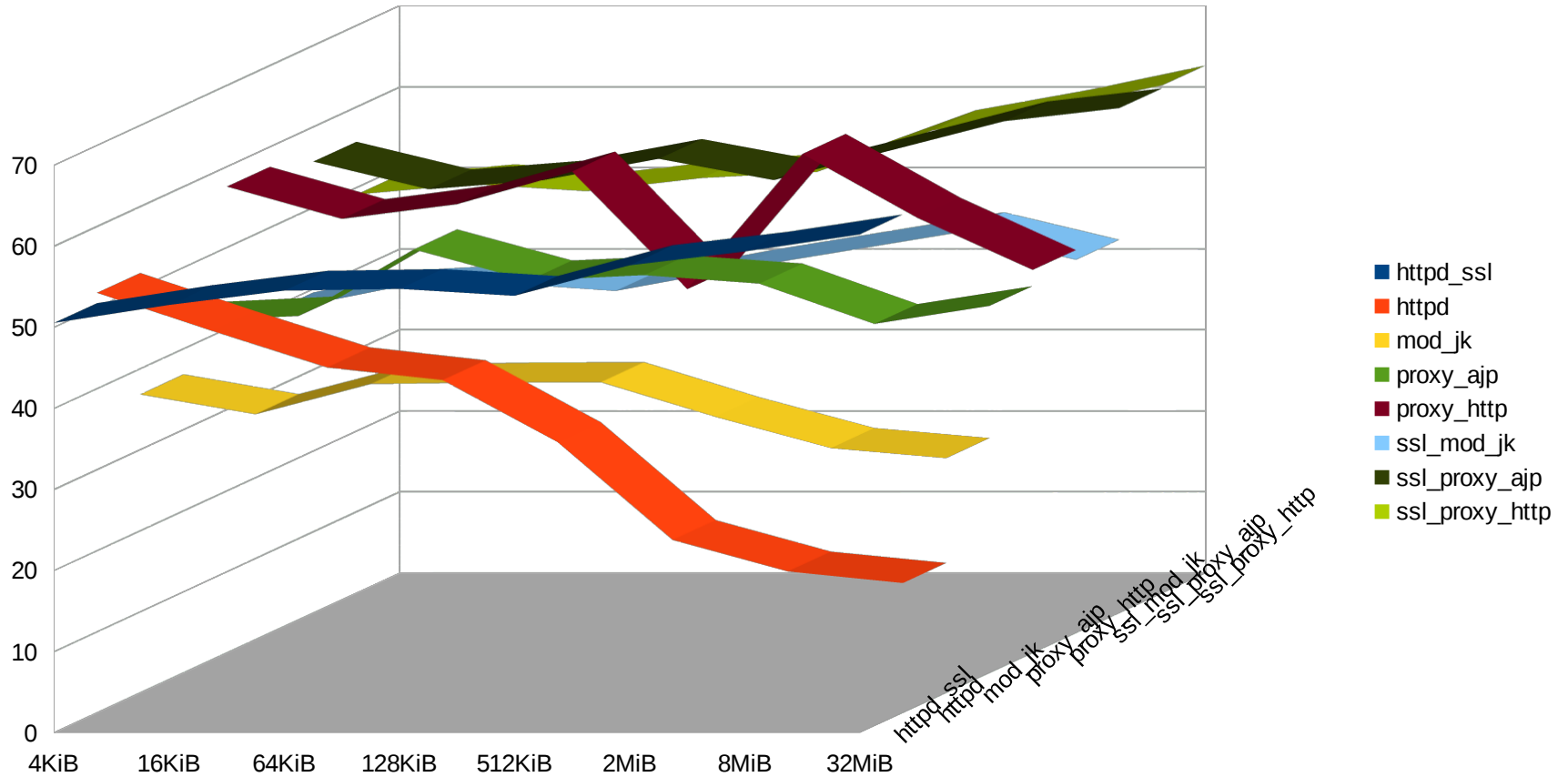


# Proxy Throughput (c4)



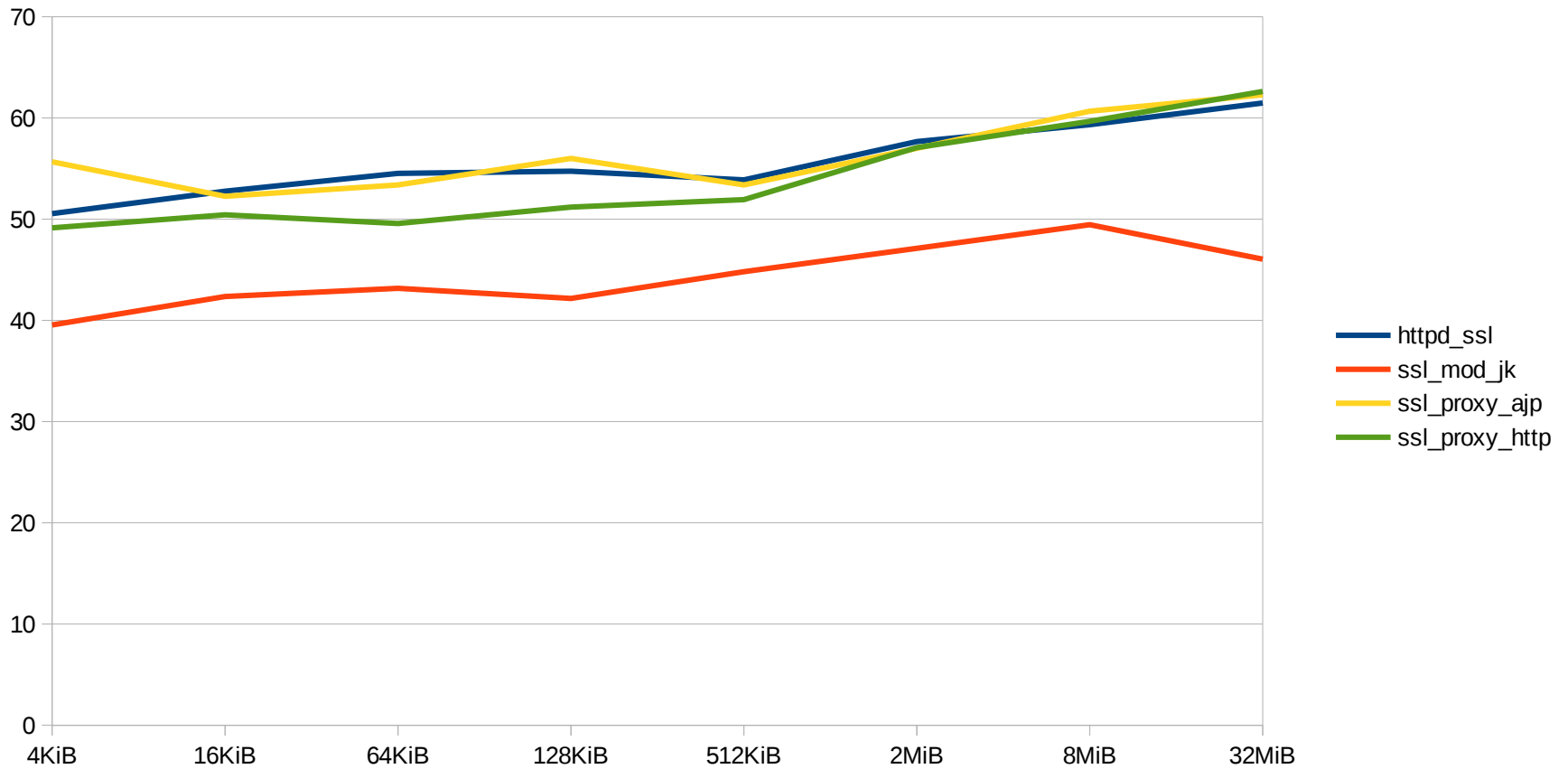


# HTTPD CPU Use (c4)



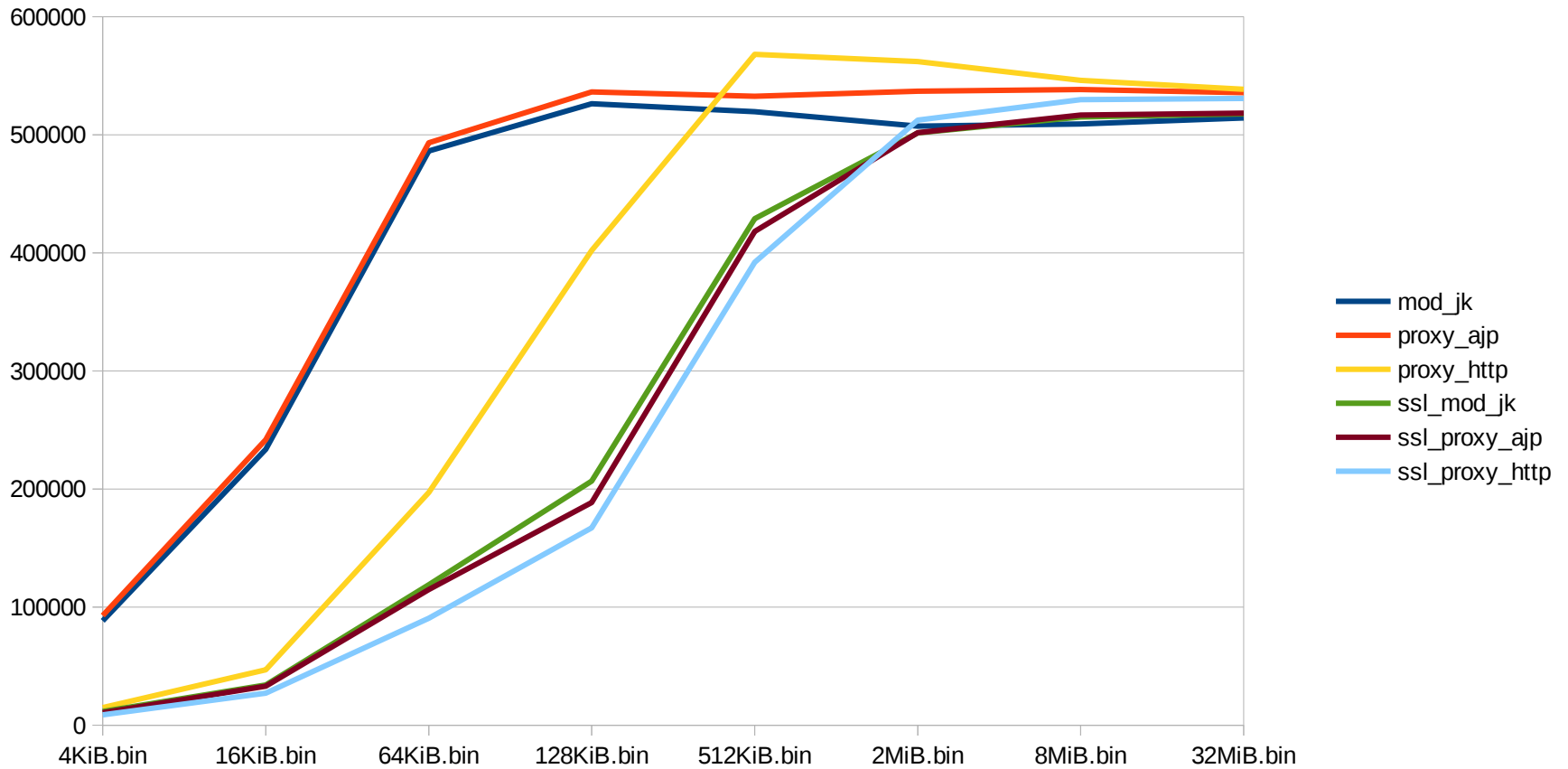


# SSL Proxy CPU Use (c4)



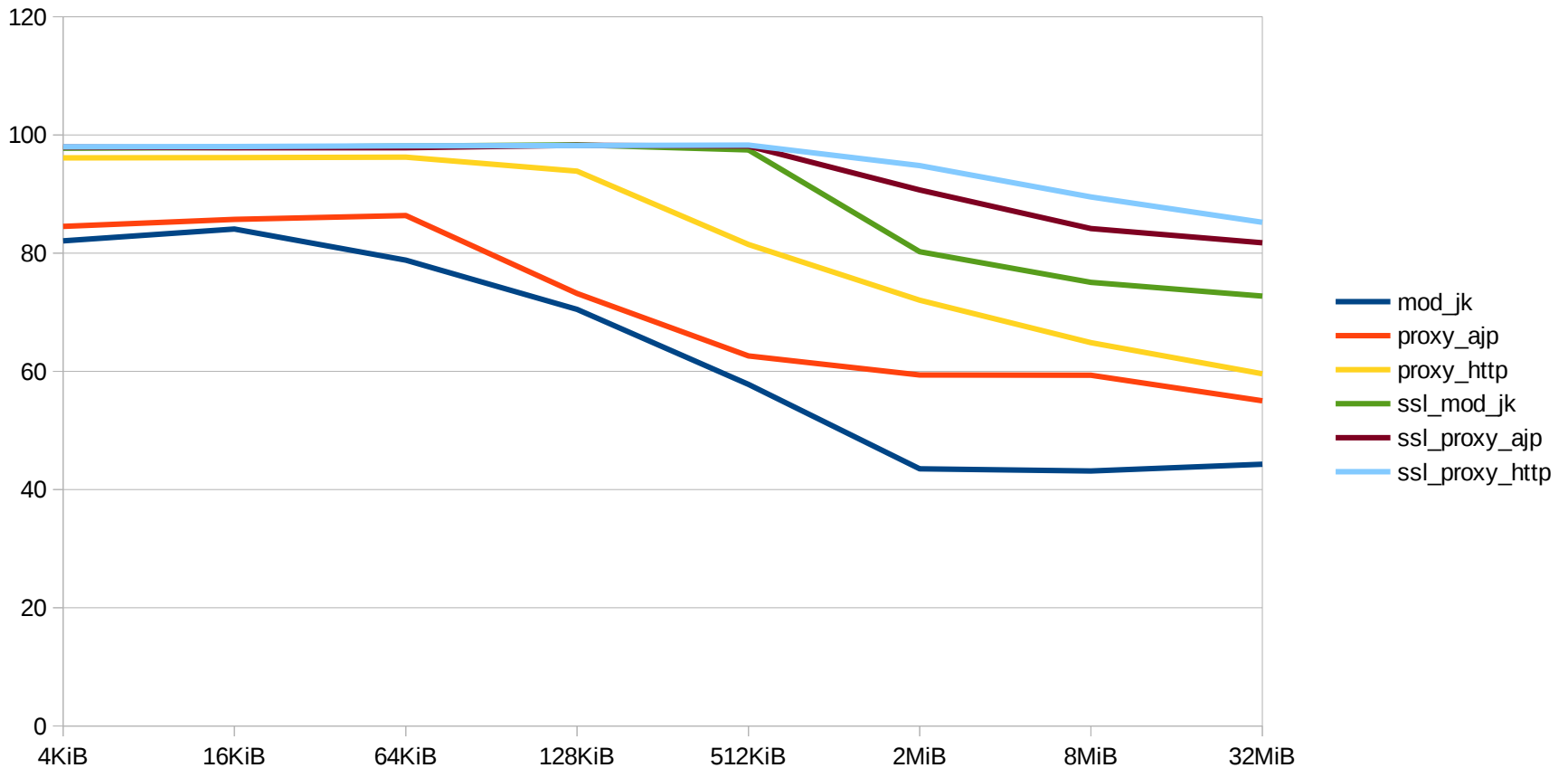


# Proxy Throughput (c40)





# Proxy CPU Use (c40)







# Connector/Proxy

## What use?

- Conclusion:
  - If you need SSL better use a proxy
    - Basically any “httpd proxy” will do the work.
    - Use mod\_jk if you need a Swiss knife configuration.
    - Use http otherwise
  - WebSocket
    - Use httpd-2.4.x for mod\_proxy\_wstunnel.



# Questions? Thank you!

- [jfclere@gmail.com](mailto:jfclere@gmail.com)
- [users@tomcat.apache.org](mailto:users@tomcat.apache.org)
- Repo with the scripts for the tests:
  - <https://github.com/jfclere/AC2014scripts>



# Choosing tomcat connectors