Using OpenSSL to boost Tomcat

Jean-Frederic Clere
What I will cover

- Who I am.
- Connectors
  - NIO, NIO2, APR
  - OpenSSL Implementation
  - HTTP/2 and ALPN in Tomcat.
- Performances tests
  - With ab and h2load as client load generator.
- Questions?
Who I am

Jean-Frederic Clere

Red Hat

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)
Tomcat
What is a Connector?

- Tomcat's interface to the world
- Binds to a port
- Understands a protocol and possible upgrades.
- Dispatches requests (example)
Tomcat Connectors

- 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.

The presentation focuses on HTTP and on NIO/NIO2.
What is new in Tomcat 9 / 8.5

- Property sslImplementationName
  - Allows replacement of the SSL code
    - OpenSSLImplementation (use OpenSSL)
    - JSSEImplementation (use JSSE)
- UpgradeProtocol
  - Allows protocol upgrade from HTTP/1.1
    - HTTP/2 (yes)
    - Websocket/Speedy (cool)
Why a new SSL Implementation

- JSSE:
  - Very slow
  - Missing features: like ALPN (JEP 244: TLS Application-Layer Protocol Negotiation)
  - Hardware acceleration very partial (like AES in java8)
- Native connector:
  - Fast but a lot of native code
  - Use OpenSSL for SSL/TLS.
- New OpenSSL implementation:
  - Fast.
  - Uses only a OpenSSL for native code (no native socket, poller etc).
  - Works with NIO and NIO2.
  - Uses OpenSSL for SSL/TLS. (warp, unwarp, handshake etc).
OpenSSL Implementation

- Code originates from netty-tcnative a forked Tomcat Native
- Prototype (last year):
  - Done with the BeFriNe University
  - Tested and ported to tc_trunk last summer
- SSL Configuration compatible with the JSSE connection (*)
- Uses keystores (*)
- Uses SSL BIO to wrap/unwarp, handshake
- Uses java NIO or NIO2 Sockets for the reads and writes
- Automatically enabled when TC native is installed/enabled (*)

5/6/16
How does that work?

SSLContext

JSSESSLContext
createSSLEngine()
SSLEngine

OpenSSLContext
createSSLEngine()
OpenSSLEngine
Overrides
wrap()
unwrap()
getSession()
etc...

etc...
How does wrap works

wrap(plaintext, encrypted)

BIO_new_bio_pair

writePlainTextData
write_ToSSL
SSL_write

internalBIO

networkBIO

SSL_set_bio

readEncryptedData
readFromBIO
BIO_read

5/6/16
How does unwrap works

unwrap(encrypted, plaintext)

BIO_new_bio_pair

readPlaintextData
readFromSSL
SSL_read

internalBIO

writeEncryptedData
writeToBIO
BIO_write

networkBIO

SSL_set_bio
Connector Performance

- Compare connectors throughput against each other
- Only static content was compared, varying file sizes
- Run on “fast” machines, 10 Gbps local network
- Tests:
  - Compare the connectors (trunk) NIO, NIO2 and APR
  - Using JSSE and OpenSSL
  - First without “sendfile”
Connector Throughput (c8)

Concurency 8

File Size

<table>
<thead>
<tr>
<th>File Size</th>
<th>Throughput KB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KiB.bin</td>
<td></td>
</tr>
<tr>
<td>8KiB.bin</td>
<td></td>
</tr>
<tr>
<td>16KiB.bin</td>
<td></td>
</tr>
<tr>
<td>32KiB.bin</td>
<td></td>
</tr>
<tr>
<td>64KiB.bin</td>
<td></td>
</tr>
<tr>
<td>128KiB.bin</td>
<td></td>
</tr>
<tr>
<td>256KiB.bin</td>
<td></td>
</tr>
<tr>
<td>512KiB.bin</td>
<td></td>
</tr>
<tr>
<td>1MiB.bin</td>
<td></td>
</tr>
<tr>
<td>2MiB.bin</td>
<td></td>
</tr>
<tr>
<td>4MiB.bin</td>
<td></td>
</tr>
<tr>
<td>8MiB.bin</td>
<td></td>
</tr>
<tr>
<td>16MiB.bin</td>
<td></td>
</tr>
<tr>
<td>32MiB.bin</td>
<td></td>
</tr>
</tbody>
</table>

Throughput KB/sec vs File Size

- coyote_apr_https
- coyote_nio2_opensslHttps
- coyote_nio_jsseHttps
- coyote_nio_opensslHttps

5/6/16
Connector Throughput (c40)

Concurency 40

File Size

Throughput Kbytes/sec

0 100000 200000 300000 400000 500000 600000 700000

8KiB.bin 16KiB.bin 32KiB.bin 64KiB.bin 128KiB.bin 256KiB.bin 512KiB.bin 1MiB.bin 2MiB.bin 4MiB.bin 8MiB.bin 16MiB.bin 32MiB.bin

- coyote_apr_https
- coyote_nio2_openssl_https
- coyote_nio_jsse_https
- coyote_nio_openssl_https
Connector Throughput (c80)

concurrency 80

Throughput KB/sec

File Size

0 100000 200000 300000 400000 500000 600000 700000

8KiB.bin 16KiB.bin 32KiB.bin 64KiB.bin 128KiB.bin 256KiB.bin 512KiB.bin 1MiB.bin 2MiB.bin 4MiB.bin 8MiB.bin 16MiB.bin 32MiB.bin

5/6/16
Connector CPU Use

Concurency 8

Concurency 40

Concurency 80

File Size

CPU usage

- coyote_apr_https
- coyote_nio2_openssl_https
- coyote_nio_jsse_https
- coyote_nio_openssl_https

5/6/16
Connector Performance

- With sendfile
  - In fact with TLS/SSL sendfile is emulated
Connector Throughput (c8)

File Size

Concurency 8

Throughput in Kbytes/sec

0 100000 200000 300000 400000 500000 600000 700000 800000

4KiB.bin 8KiB.bin 16KiB.bin 32KiB.bin 64KiB.bin 128KiB.bin 256KiB.bin 512KiB.bin 1MiB.bin 2MiB.bin 4MiB.bin 8MiB.bin 16MiB.bin 32MiB.bin

coyote_apr_https
coyote_nio2_openssl_https
coyote_nio_jsse_https
coyote_nio_openssl_https

5/6/16
Connector Throughput (c40)

Concurency 40

Throughput in Kbytes/sec

File Size

- coyote_apr_https
- coyote_nio2_openssl_https
- coyote_nio_jsse_https
- coyote_nio_openssl_https

5/6/16
Connector Throughput (c80)

Concurency 80

File Size

Throught in Kbytes/sec

- coyote_aprHttps
- coyote_nio2_opensslHttps
- coyote_nio_jssseHttps
- coyote_nio_opensslHttps

5/6/16
Connector CPU Use

Concurrent 8

Concurrent 40

Concurrent 80

- coyote_apr_https
- coyote_nio2_openssl_https
- coyote_nio_jsse_https
- coyote_nio_openssl_https

5/6/16
Connector Performance

• Conclusion:
  - OpenSSL performs better than JSSE
  - NIO and NIO(2) give similar results
  - Emulated sendfile doesn't help a lot (bigger files better).
  - APR isn't needed
  - Until Java 9 is released OpenSSL is needed for HTTP/2
Questions?
Thank you!

- jfclere@gmail.com
- users@tomcat.apache.org
- Repo with the scripts for the tests:
  - https://github.com/jfclere/AC2014scripts