TRIDIUM

NS2024

POWER OF PARTNERSHIP

Secure IP Driver with NDriver and Java Secure Socket Extension

Brian Todd

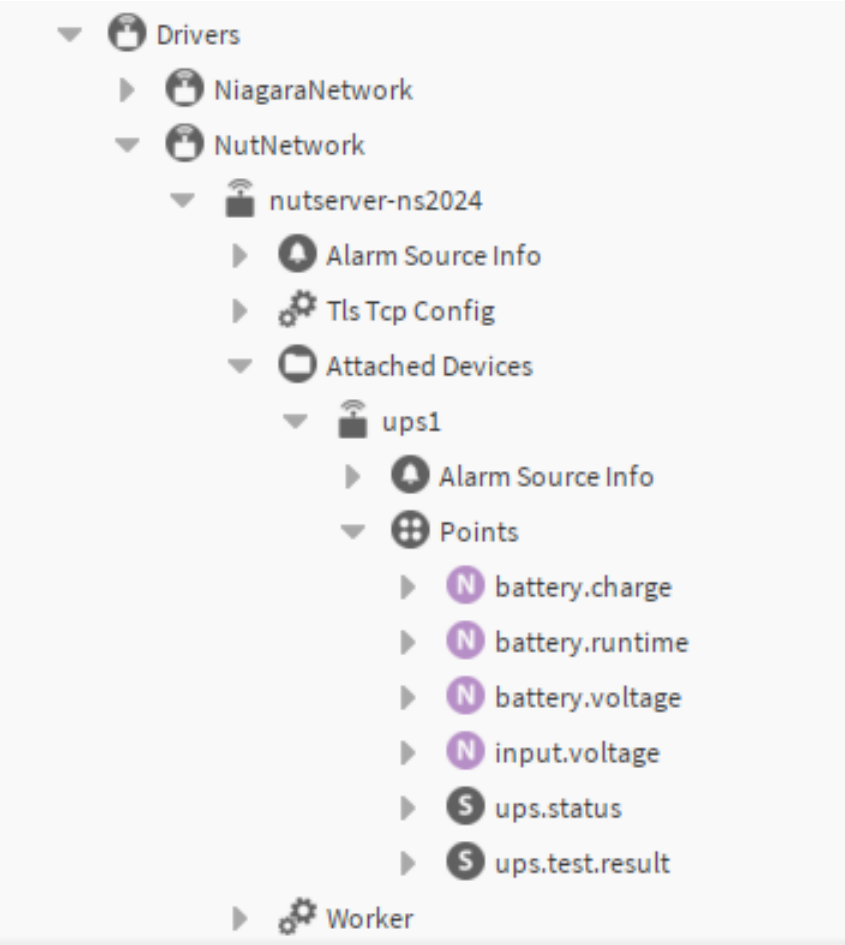
TRIDIUM

Hypothetical NDriver Implementation - NDriver for Network UPS Tools (NUT)

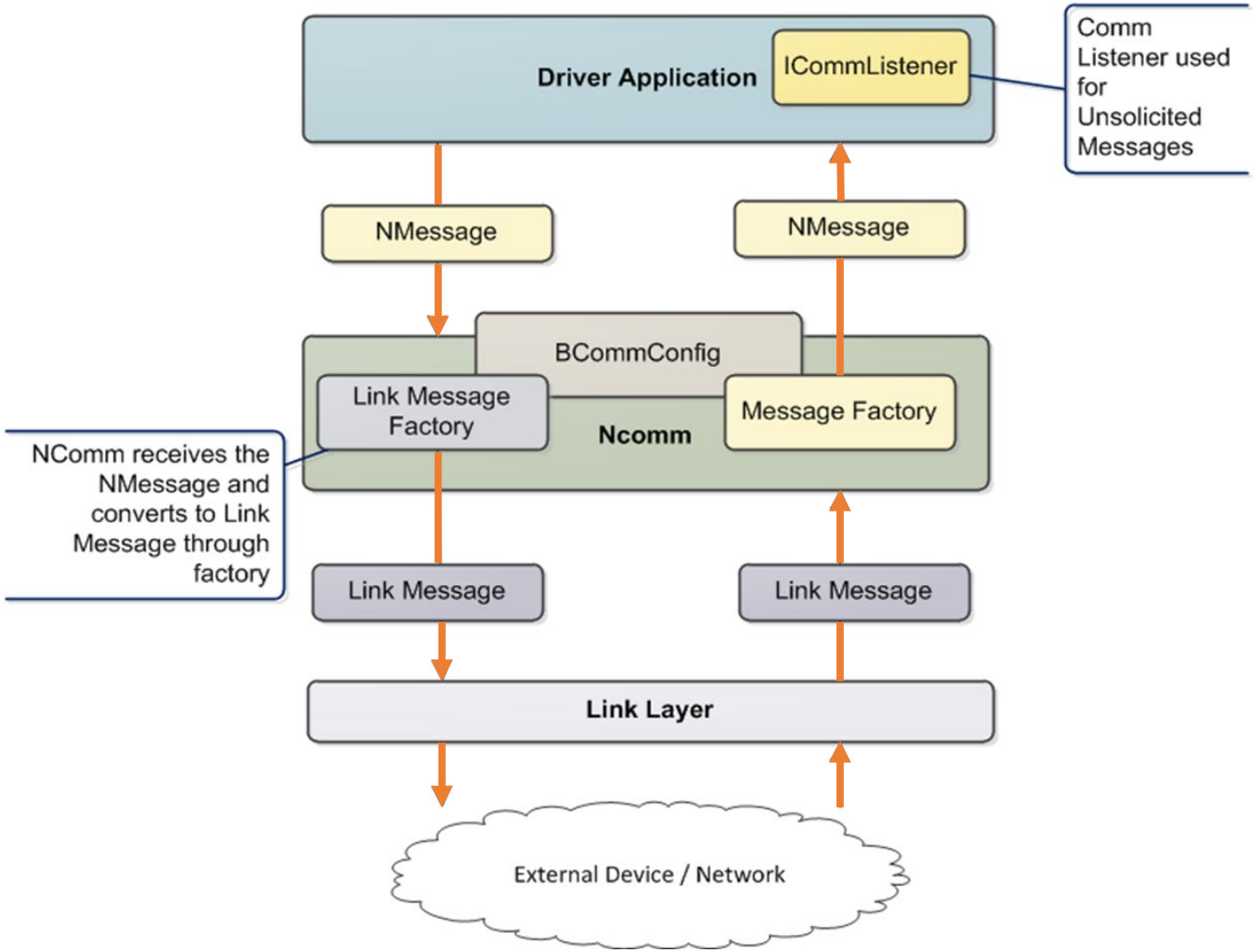
Hypothetical: Existing NUT TCP/IP NDriver implementation using *TcpLinkLayer* needs to be to modified to allow encrypted communication.

Implementation Plan: Create an implementation of ILinkLayer capable of sending/receiving encrypted messages using Transport Layer Security (TLS) that supports:

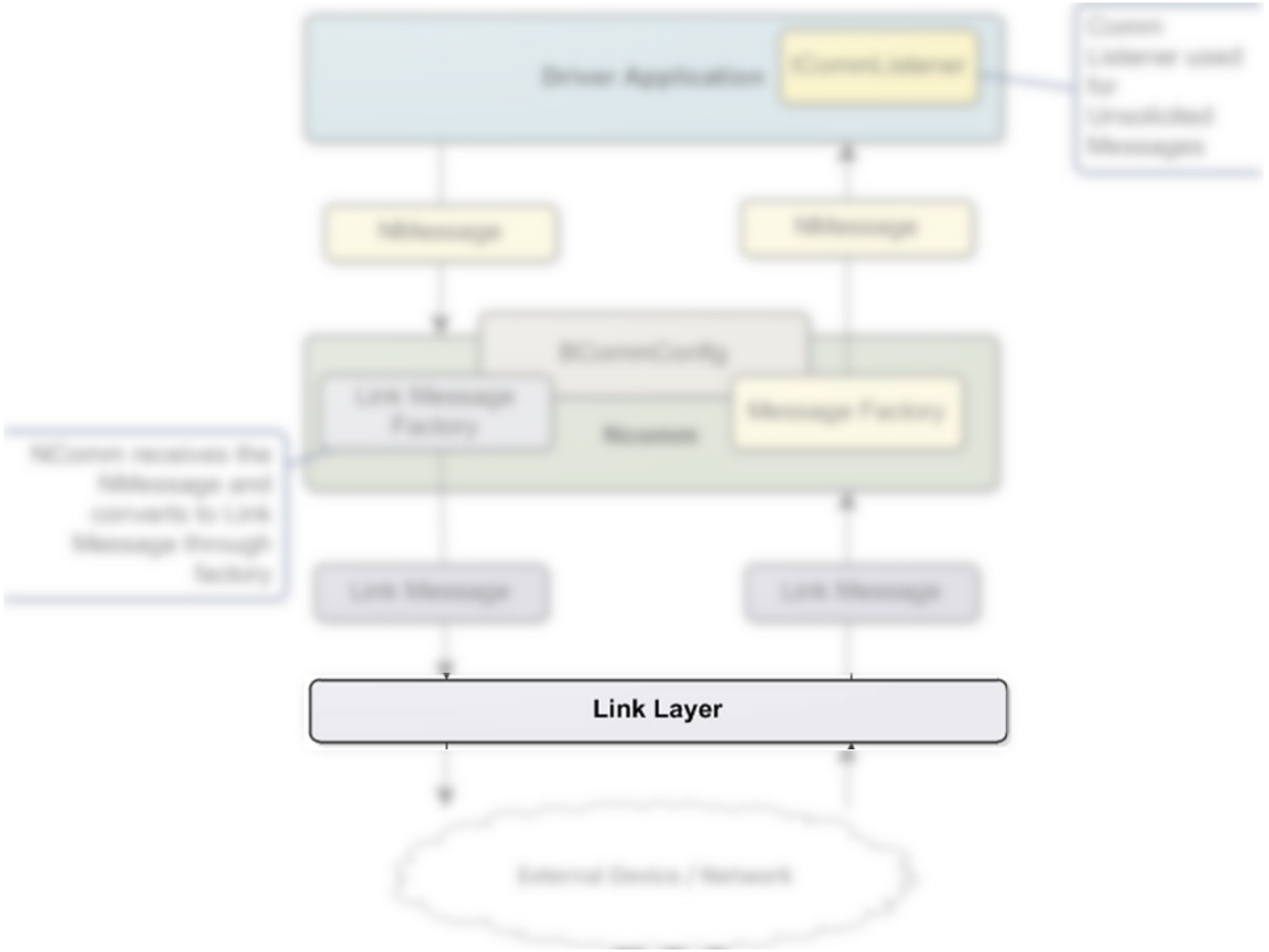
- Specifying a minimum TLS protocol
- Use of mutual TLS Authentication (mTLS)
- Optional Opportunistic TLS (STARTTLS)
  - Ability to reject connection if encrypted connection can not be established



NDriver Architecture Primer



NDriver Link Layer – A Closer Look



Transport Layer Security (TLS)

- The most widely used protocol for implementing cryptography on the web
- Secure enhancement to standard TCP/IP Socket Protocol
- Effective when used correctly -- though, incorrect usage may result in a compromise of confidentiality
- Positioned between Transport (usually TCP/IP) and Application Layers
- Largely transparent to Application Protocol

Source: TLS protocol overview - IBM Documentation

Java Secure Socket Extension (JSSE) Primer

Security component of the Java SE platform that provides SSL/TLS

javax.net.SSLSocketFactory — Factory for creating pre-configured secure sockets

javax.net.SSLSocket — Subclass of *java.net.Socket* that adds secure socket-specific functionality

Usually created by *SSLSocketFactory* using the *createSocket()* methods

javax.net.ssl.TrustManager — Determines whether the remote authentication credentials (and thus the connection) should be trusted

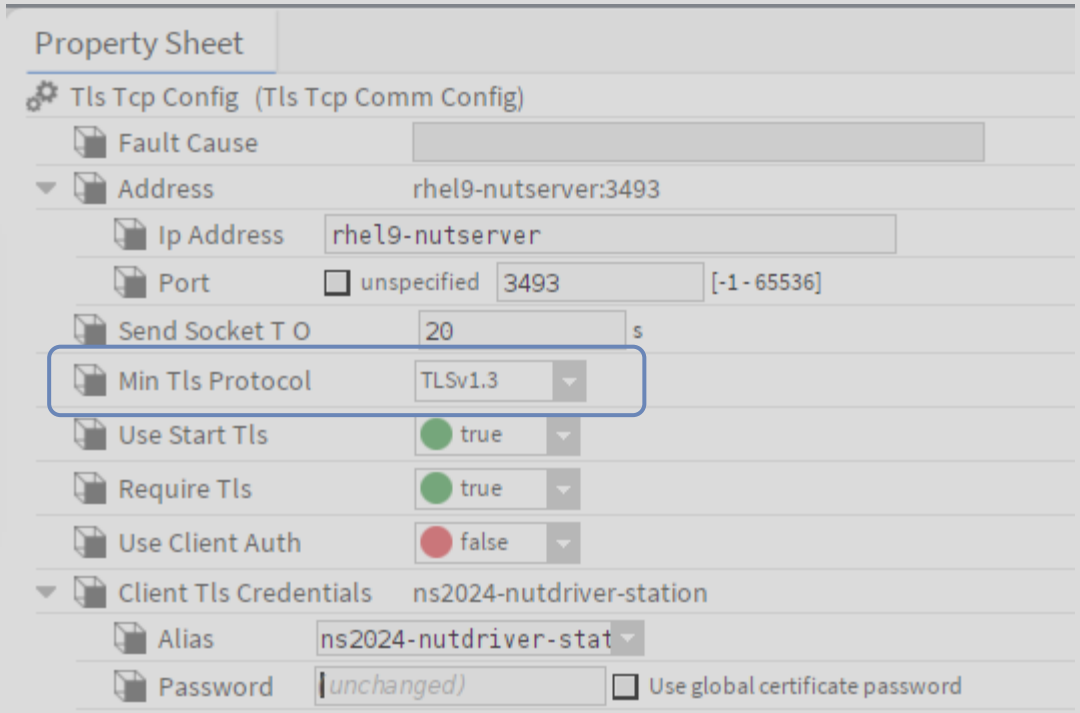
javax.net.ssl.KeyManager — Determines which authentication credentials to send to the remote host

TLS TCP Comm Config Implementation

```
@NiagaraType
{ @NiagaraProperty... }
/*
 Specify minimum TLS protocol to be used.
```

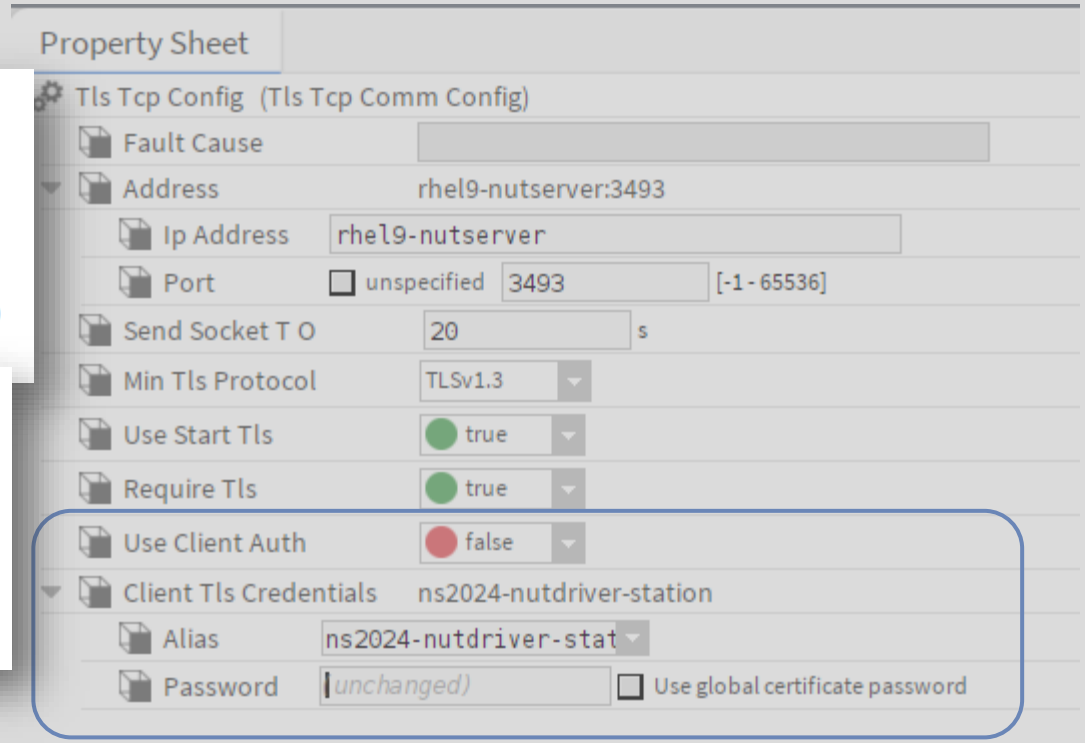
```
/*
 Specify minimum TLS protocol to be used.
 */
@NiagaraProperty(
  name = "minTlsProtocol",
  type = "BSslTlsEnum",
  defaultValue = "BSslTlsEnum.tlsv1_2",
  facets = @Facet("BFacets.make(BFacets.SECURITY, BBoolean.TRUE)")
)
```

```
 extends BTcpCommConfig
{
  ...
}
```



TLS TCP Comm Config Implementation

```
@NiagaraType
{ @NiagaraProperty... }
/*
  Controls whether the driver will use TLS Client Authentication
 */
@NiagaraProperty(
  name = "useClientAuth",
  type = "boolean",
  defaultValue = "false",
  facets = @Facet("BFacets.make(BFacets.SECURITY, BBoolean.TRUE)")
)
/*
  Credentials to use for TLS Client Authentication
 */
@NiagaraProperty(
  name = "clientTlsCredentials",
  type = "BCertificateAliasAndPassword",
  defaultValue = "BCertificateAliasAndPassword.DEFAULT",
  facets = @Facet("BFacets.make(BFacets.SECURITY, BBoolean.TRUE)")
)
public class BTlsTcpCommConfig
  extends BTcpCommConfig
{
  ...
}
```



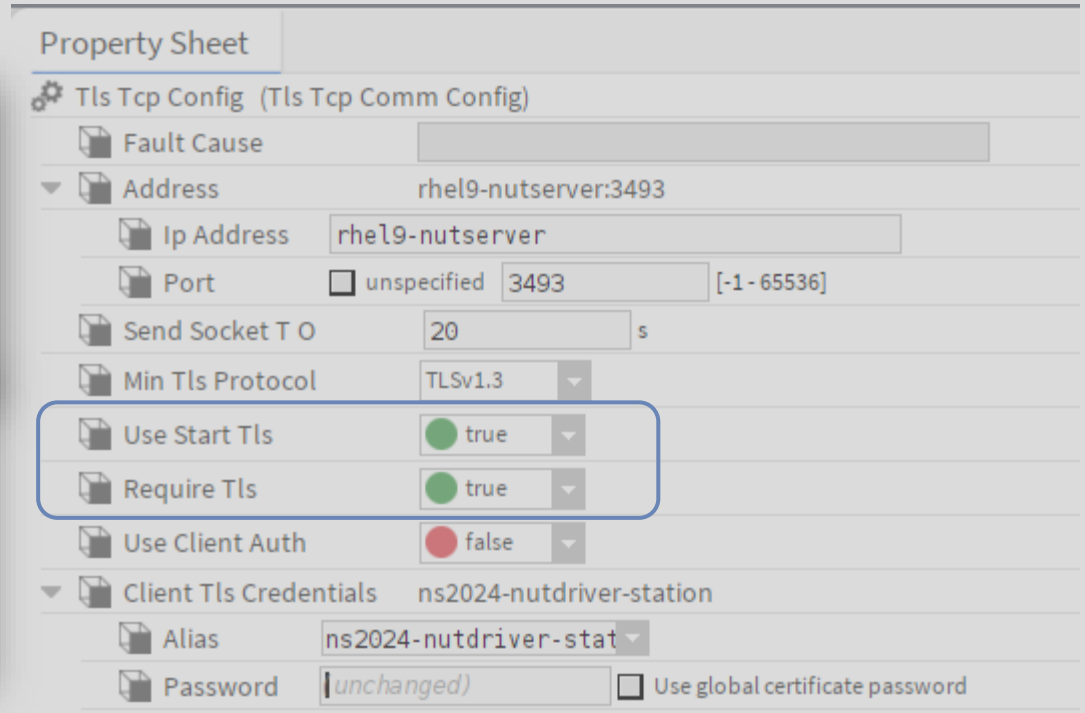
TLS TCP Comm Config Implementation

```
@NiagaraType
{ @NiagaraProperty... }

/*
 Use Opportunistic TLS (i.e., 'STARTTLS') if server/application
  does not support use of direct SSL/TLS.
 */
@NiagaraProperty(
  name = "useStartTls",
  type = "boolean",
  defaultValue = "false",
  facets = @Facet("BFacets.make(BFacets.SECURITY, BBoolean.TRUE)")
)

/*
  Controls whether connections to remote devices should
   be allowed if STARTTLS fails.
 */
@NiagaraProperty(
  name = "requireTls",
  type = "boolean",
  defaultValue = "true",
  facets = @Facet("BFacets.make(BFacets.SECURITY, BBoolean.TRUE)")
)

{ @NiagaraProperty(... }
public class BTlsTcpCommConfig
  extends BTcpCommConfig
{
  ...
}
```

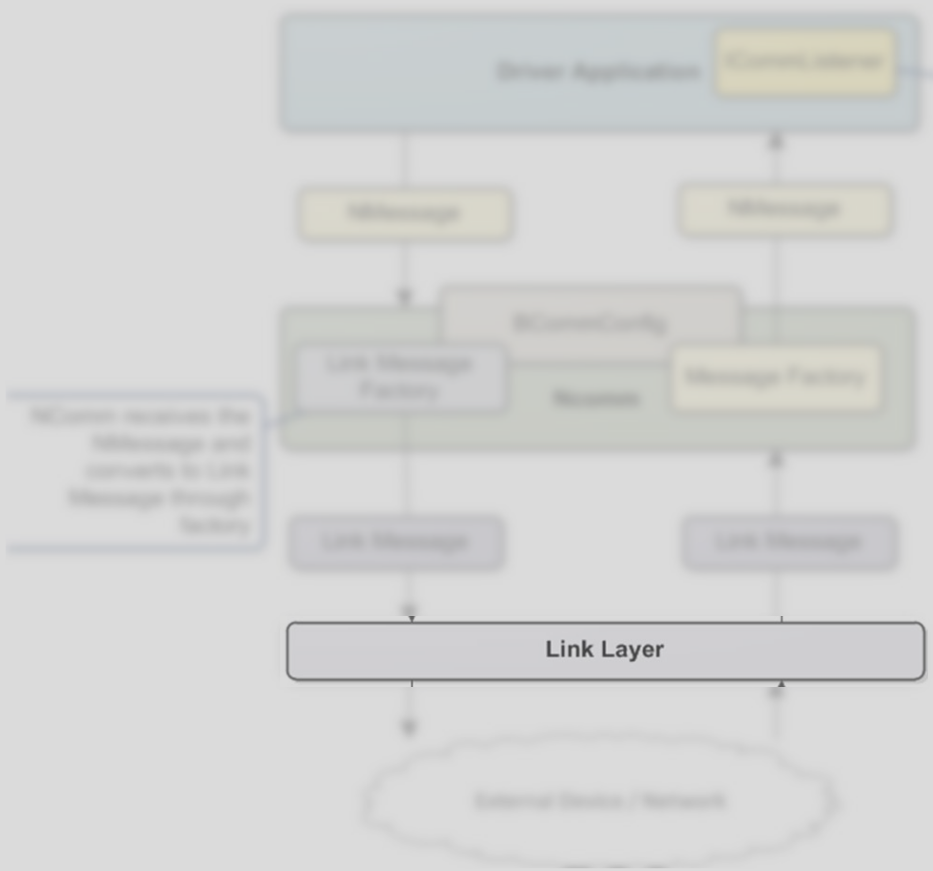


TLS TCP Comm Config Implementation

```
@NiagaraType
@NiagaraProperties...
public class BTlsTcpCommConfig
  extends BTcpCommConfig
{                                          TlsTcpLinkLayer} to be used by NComm.
  ...

  public ILinkLayer makeLinkLayer(NComm comm)
  {
    return new TlsTcpLinkLayer(comm,  commConfig: this);
  }
```

ILinkLayer Interface




```
/**
 * ILinkLayer is interface implemented by linkLayers which are coupled to
 * NComm.
 */
public interface ILinkLayer
```

```
/**
 * Comm config parameter change.  Check if link layer needs restart. - called
 * from NComm.verifySettings()
 */
void verifySettings(BCommConfig comCfg) throws Exception;

/**
 * Start linklayer - called from NComm.start()
 */
void start() throws Exception;

/**
 * Stop linklayer  - called from NComm.stop()
 */
void stop();
```

```
/**
 * Provide some spy debug - called from NComm.spy()
 */
void spy(SpyWriter out) throws Exception;

/**
 * Reset any statistical counters - called from NComm.resetStats()
 */
void resetStats();
```

```
/**
 * Entry point for NComm to send message to link layer
 */
void sendMessage(LinkMessage msg) throws Exception;
```

Implementing ILinkLayer

```
/**
 * ILinkLayer implementation for TCP interface with Secure Socket (SSL/TLS).
 * Send and receive encrypted messages using TLS over TCP sockets.
 */
class TlsTcpLinkLayer
  implements ILinkLayer
{
  /**
   * Constructs an instance of TlsTcpLinkLayer.
   *
   * @param ncomm      the {@link NComm} instance.
   * @param commConfig the {@link BTlsTcpCommConfig} for this link layer.
   */
  TlsTcpLinkLayer(NComm ncomm, BTlsTcpCommConfig commConfig)
  {
    this.ncomm = ncomm;
    this.commConfig = commConfig;
    linkSessionsByAddress = new ConcurrentHashMap<>( initialCapacity: 30);
    logger = Logger.getLogger( name: commConfig.getResourcePrefix() + "LinkLayer");
  }

  // { ... }
```

```
private final NComm ncomm;
private final Map<String, TlsLinkSession> linkSessionsByAddress;
private final Logger logger;
private BTlsTcpCommConfig commConfig;
private Statistics statistics = new Statistics();
```

Implementing ILinkLayer - LinkSession

```
/**
 * TlsLinkSession represents a single TLS over TCP/IP connection.
 */
private class TlsTcpLinkSession
  implements Runnable
{
  /**
   * Constructs an instance of this session.
   *
   * @param address The {@link BIpAddress} for this session.
   * @throws Exception If an error occurred obtaining the {@link ICryptoManager}.
   */
```

```
TlsTcpLinkSession(BIpAddress address)
  throws Exception
{
  cryptoManager = CertManagerFactory.getInstance();
  this.address = (BIpAddress)address.newCopy( exact: true);
  timeOut = commConfig.getSendSocketTO() * 1000;
}

private final BIpAddress address;
private final ICryptoManager cryptoManager;
private final int timeOut;
private Socket socket;
```

Implementing ILinkLayer - LinkSession

```
private class TlsTcpLinkSession
  implements Runnable
{
  // { ... }

  /**
   * Sends the given {@link LinkMessage}.
   *
   * @param linkMessage The message to send.
   * @throws IOException If an error occurred sending the message.
   */
  synchronized void send(LinkMessage linkMessage)
    throws IOException
  {
    // If not connected, connect the socket before writing to output stream.
    if (socket == null || socket.isClosed())
    {
      connectSocket();
    }
    if (socket == null || !socket.isConnected())
    {
      throw new IOException(String.format("Can't connect to %s", address));
    }
    // Start a new thread on which responses will be read.
    Thread receiverThread = new Thread( target: this,  name: commConfig.getResourcePrefix() + ".LinkSession");
    receiverThread.start();
    socket.getOutputStream().write(linkMessage.getByteArray(),  off: 0, linkMessage.getLength());
    }
    finally
    {
      commConfig.getLinkMessageFactory().releaseLinkMessage(linkMessage);
    }
  }

  // { ... }
}
```

Implementing ILinkLayer - LinkSession

```
private class TlsTcpLinkSession implements Runnable {

  // { ... }

  /**
   * Receive the response to the message that was sent.
   */
  @Override
  public void run() {
    try { socket.setSoTimeout(timeOut); }
    catch (SocketException e) { logger.log(Level.SEVERE, String.format("Unable to set socket timeout %s", address), e); }

    LinkMessage linkMessage = commConfig.getLinkMessageFactory().getLinkMessage();
    linkMessage.address = address;
    InputStream in = socket.getInputStream();
    boolean complete = linkMessage.receive(in);
    if (complete) {
      ncomm.receiveMessage(linkMessage);
      statistics.messagesReceived++;
    } else {
      commConfig.getLinkMessageFactory().releaseLinkMessage(linkMessage);
      terminate();
    }
    }
    catch (Throwable e) {
      logger.log(Level.WARNING, String.format("Exception caught in LinkReceiver for %s", address), e);
      terminate();
    }
  }

  // { ... }
}
```

Implementing ILinkLayer - LinkSession

```
private class TlsTcpLinkSession
  implements Runnable
{
  // { ... }

  /**
   * Obtains a {@link Socket} to a remote host that has been connected.
   */
  private void connectSocket() {
    try {
      ClientTlsParameters clientTlsParameters = makeClientTlsParameters();
      socket = commConfig.getUseStartTls()
        ? connectAndDoStartTlsUpgrade(clientTlsParameters)
        : connectSecure(clientTlsParameters);
    } catch (Exception e) {
      if (socket != null) {
        try {
          socket.close();
        } catch (IOException ignored) {}
        socket = null;
      }
      logger.log(Level.SEVERE, msg: "Unable to establish secure connection to {0}", address);
    }
  }

  // { ... }
}
```

Obtaining an SSLSocketFactory – The Easy Way

- *BajaSSLSocketFactory*
  - Public API implementation of *javax.net.ssl.SSLSocketFactory*
  - Creates *SSLSocket* instances using 'supported' Cipher Suite.
  - Does not allow further configuration

- *ICryptoManager.getClientSocketFactory(ClientTlsParameters)*
  - Creates *SSLSocket* instances configured using the provided *ClientTlsParameters*
  - Supports creation of *SSLSocket* instances that can be used to:
    - Negotiate a Minimum TLS Protocol
    - Configure a Certificate Alias and Private Key password that a client can use as authentication credentials if using mTLS.

Configuring SSLSocketFactory - Configuring ClientTlsParameters

```
private class TlsTcpLinkSession
  implements Runnable
{
  // { ... }

  /**
   * Constructs an instance of {@link ClientTlsParameters} from the {@link BTlsTcpCommConfig}.
   *
   * @return the ClientTlsParameters that will be used to configure {@link SSLSocketFactory} instances.
   */
  private ClientTlsParameters makeClientTlsParameters() {
    if (commConfig.getUseClientAuth()) {
      clientTlsParameters = new ClientTlsParameters(
        commConfig.getMinTlsProtocol().getTag(),
        commConfig.getClientTlsCredentials().getAlias()
      );
      if (commConfig.getClientTlsCredentials().isPasswordSet()) {
        try (SecretChars passwordChars =
               AccessController.doPrivileged(
                 (PrivilegedAction<SecretChars>)commConfig.getClientTlsCredentials()
                                                          .getPassword()::getSecretChars
               )
        ) {
          clientTlsParameters.setKeyPassphrase(passwordChars.get());
        }
      }
    } else {...}

    if (commConfig.getUseClientAuth()) {...}
    else {
      clientTlsParameters = new ClientTlsParameters(commConfig.getMinTlsProtocol().getTag());
    }

    // { ... }
  }
```

TLS Client Authentication – Updating module-permissions.xml

```
<permissions>
  <niagara-permission-groups type="station">
    <req-permission>
      <name>NETWORK_COMMUNICATION</name>
      <purposeKey>Outside access for Driver</purposeKey>
      <parameters>
        <parameter name="hosts" value="*"/>
        <parameter name="ports" value="*"/>
        <parameter name="type" value="all"/>
      </parameters>
    </req-permission>
  </niagara-permission-groups>
</permissions>
      <purposeKey>Access to UserKeyStore for TLS Client Authentication</purposeKey>
      <parameters>
        <parameter name="keystores" value="userKeyStore" />
        <parameter name="actions" value="read" />
      </parameters>
```

Configuring SSLSocketFactory

```
private class TlsTcpLinkSession implements Runnable
{
  // { ... }

  /**
   * Obtain an {@link SSLSocket} using the given ClientTlsParameters.
   * @param clientTlsParameters The clientTlsParameters to use when configuring the SSLSocketFactory.
   * @return An SSLSocket to the remote host in the connected state.
   * @throws IOException If an exception occurs obtaining or connecting the SSLSocket.
   */
  private SSLSocket connectSecure(ClientTlsParameters clientTlsParameters)
    throws IOException
  {
    try
    return AccessController.doPrivileged((PrivilegedExceptionAction<SSLSocket>)() -> {
      SSLSocketFactory secureSocketFactory =
        (SSLSocketFactory)cryptoManager.getClientSocketFactory(clientTlsParameters);
      SSLSocket secureSocket =
        (SSLSocket)secureSocketFactory.createSocket(address.getInetAddress(), address.getPort());
      secureSocket.setUseClientMode(true);
      secureSocket.startHandshake();
      return secureSocket;
    });
    catch(PrivilegedActionException e)
    {
      throw new IOException("Unable to establish secure connection to " + address, e.getCause());
    }
  }

  // { ... }
}
```

Configuring SSLSocketFactory - STARTTLS

```
private Socket connectAndDoStartTlsUpgrade(ClientTlsParameters clientTlsParameters)
  throws IOException
{
  Socket plainSocket =
    SocketFactory.getDefault().createSocket(address.getInetAddress(), address.getPort());
  BufferedReader in =
    new BufferedReader(new InputStreamReader(plainSocket.getInputStream()));
  BufferedWriter out
    = new BufferedWriter(new OutputStreamWriter(plainSocket.getOutputStream()));
  out.write( str: "STARTTLS"); // Send the STARTTLS message, and read the response.
  out.newLine();
  out.flush();
  String response = in.readLine();

  { ... }

}
```

Configuring SSLSocketFactory - OK STARTTLS

```
private Socket connectAndDoStartTlsUpgrade(ClientTlsParameters clientTlsParameters)
  throws IOException
{
  { ... }
  if ("OK STARTTLS".equals(response)) {
    try {
      return AccessController.doPrivileged((PrivilegedExceptionAction<SSLSocket>)() -> {
        SSLSocketFactory secureSocketFactory =
          (SSLSocketFactory)cryptoManager.getClientSocketFactory(clientTlsParameters);
        SSLSocket secureSocket =
          (SSLSocket)secureSocketFactory.createSocket(
            plainSocket,
            plainSocket.getInetAddress().getHostAddress(),
            plainSocket.getPort(),
             autoClose: true
        );
        secureSocket.setUseClientMode(true);
        secureSocket.startHandshake();
        return secureSocket;
      });
    } catch(PrivilegedActionException e) {
      throw new IOException("Unable to complete STARTTLS upgrade", e.getCause());
    }
  } else {...}
}
```

Configuring SSLSocketFactory – ERR STARTTLS

```
private Socket connectAndDoStartTlsUpgrade(ClientTlsParameters clientTlsParameters)
  throws IOException
{
  { ... }
  if ("OK STARTTLS".equals(response)) {...}
  else {  // Server returned an Error...
    if (commConfig.getRequireTls()) {
      throw new IOException(
        String.format("Unable to complete STARTTLS upgrade, error response: %s", response)
      );
    } else {
      logger.warning( msg: "Unable to complete STARTTLS upgrade, returning non-secure socket");
      return plainSocket;
    }
  }
}
```

Implementing ILinkLayer – LinkSession Termination

```
private class TlsTcpLinkSession
  implements Runnable
{
  // { ... }

  /**
   * Closes this session's Socket, and removes the session from the session map.
   */
  void terminate()
  try
  {
    if (socket != null)
    {
      socket.close();
    }
  }
  catch (IOException e)
  {
    logger.log(
      Level.WARNING,
      msg: "Socket for: {0} couldn't be closed during session termination",
      address
    );
  }
  linkSessionsByAddress.remove(address.toString());
  }

  // { ... }
}
```

Implementing ILinkLayer – Sending Messages

```
class TlsTcpLinkLayer
  implements ILinkLayer
{
  // { ... }

  /**
   * Send a {@link LinkMessage}.
   * @param linkMessage The message to send.
   * @throws Exception If there was an error sending the given LinkMessage.
   */
  @Override
  public void sendMessage(LinkMessage linkMessage)
    throws Exception
  {
    TlsTcpLinkSession linkSession = linkSessionsByAddress.computeIfAbsent(linkMessage.address.toString(), key -> {
      try
      {
        return new TlsTcpLinkSession((BIpAddress)linkMessage.address);
      }
      catch (Exception e)
      {
        throw new BajaRuntimeException("Unable to create TlsTcpLinkSession for address: " + key, e);
      }
    });
    logger.log(Level.FINE, msg: "Sending message {0} to {1}", new Object[]{ linkMessage, linkMessage.address });
    linkSession.send(linkMessage);
    statistics.messagesSent++;
  }

  // { ... }
}
```

Implementing ILinkLayer – Verifying Comm Config Changes

```
class TlsTcpLinkLayer
  implements ILinkLayer
{
  // { ... }

  /**
   * Called when comm config parameter changes. Will disconnect all active sessions.
   *
   * @param commConfig the communication stack configuration to be verified.
   */
  @Override
  public void verifySettings(BCommConfig commConfig)
    throws Exception
  {
    logger.log(Level.FINE,  msg: "Verifying TlsTcpLinkLayer CommConfig ");
    this.commConfig = (BTlsTcpCommConfig)commConfig;
    // Stop linkSessions
    for (TlsTcpLinkSession t : linkSessionsByAddress.values())
    {
      t.notifyAll();
      t.terminate();
    }
    linkSessionsByAddress.clear();
    resetStats();
  }

  // { ... }
}
```

Implementing ILinkLayer - Life Cycle Methods

```
class TlsTcpLinkLayer
  implements ILinkLayer
{
  // { ... }

  /**
   * Starts this Link Layer.
   */
  @Override
  public void start()
    throws Exception {
    logger.log(Level.FINE, msg: "Starting TlsTcpLinkLayer");
  }

  /**
   * Stops this Link Layer, shutting down all TlsTcpLinkSessions.
   */
  @Override
  public void stop() {
    logger.log(Level.FINE, msg: "Stopping TlsTcpLinkLayer");
    // Stop linkSessions
    for (TlsTcpLinkSession t : linkSessionsByAddress.values())
    {
        t.notifyAll();
        t.terminate();
    }
    linkSessionsByAddress.clear();
  }

  // { ... }
}
```

Troubleshooting/Debugging Tips

- Bouncycastle JSSE Debug Logging (category: org.bouncycastle.jsse, level: FINE)
- Reference TLS Alert Protocol (Alert IDs) to help decipher often cryptic log messages
- OpenSSL *s_client*
- Packet Analysis Tools (i.e., Wireshark)

Security Considerations

Test your implementation to see how it behaves when presented with abnormal SSL certificates

Don’t modify application code or disable certificate validation for testing with self-signed certificates.

Don’t rely on default settings to be the best fit, always explicitly set the options specific to the application

Resources and Further Learning

- Java Secure Socket Extension (JSSE) Reference Guide - JDK 8
  - https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html
- Transport Layer Security (TLS) Protocol Overview - JDK 8
  - https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/tls.html
- Network UPS Tools Project Website
  - https://networkupstools.org
- Niagara Community
  - https://www.niagara-community.com

TRIDIUM

NS2024

POWER OF PARTNERSHIP