NEX Troubleshoot, Manage & Monitor ECP/EDX

Version: 1.0.2

Date: 16.05.2024

1 Overview

This document aims to

- explain core functionality of ECP in order to understand troubleshooting
- explain stepwise troubleshooting, starting from high level and going to low level
- offer some tools to help manage and troubleshoot
- explain monitoring

1.1 Some terminology

Short form	Explanation
BA	Business Application, usually connected to EDX-Toolbox
CD	Component Directory – stores vital information about 'components', which is both Endpoints and Brokers.
BR	ECP Central Broker, necessary to transmit messages from one EP to another EP
ECP	Most often this will refer to ECP-endpoint, but it could also be used to refer to the platform or other parts of the platform.
EDX	EDX-Toolbox, an application that sits between BA and EP.
EP	ECP-Endpoint, we consider EDX (EDX-Toolbox) as an add-on on top of EP.
IBR	ECP Internal Broker, an ActiveMQ-broker which runs within the EP
MP	Message Path, used to route message to an EP
MT	Message Type, import/mandatory header-information for ECP-messages
SC	EDX Service Catalogue, an important concept for addressing (think of DNS vs IP)
TSO	Transmission System Operator (Statnett, SvK, Energinet, Fingrid)

2 Core Concept: Certificates

When something goes wrong, it can be related to certificates. If you suspect that to be the case it might be worthwhile to get an overview of how certificates are used, and therefore how they might fail/interfere.

2.1 Which certificates are created and their usage

When an ECP-endpoint (EP), also known as a Component, is registered in Component Directory (CD) it makes 3 certificates. They have all a private part kept in the database of the EP and a public part shared with the CD. These certificates are:

- Authentication (AUTH): Used to authenticate the EP when it creates HTTPS-connection to the CD and when it creates AMQPS-connection to the ECP Central Broker (BR).
- Encryption (ENC): Used to encrypt the message content
- Signing (SIGN): Used to make a signature of the message headers

When an BR, also a Component, is registered in CD it makes 1 certificate:

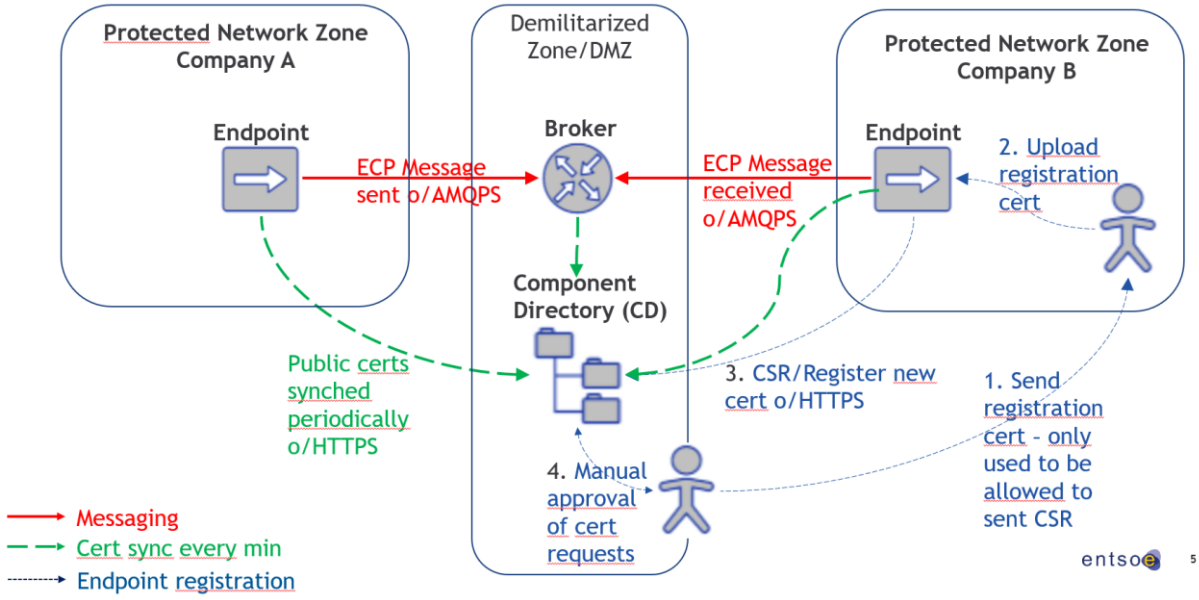
- Authentication (AUTH): Used in the same way as the EP to connect to the CD. It is also used in the AMQPS-connection, but only for EPs to trust it – not for EPs to authenticate the broker.

When a CD, also a Component, is created, it too makes an AUTH-certificate. It is used in the HTTPS-connection to all other Components, but only for trust.

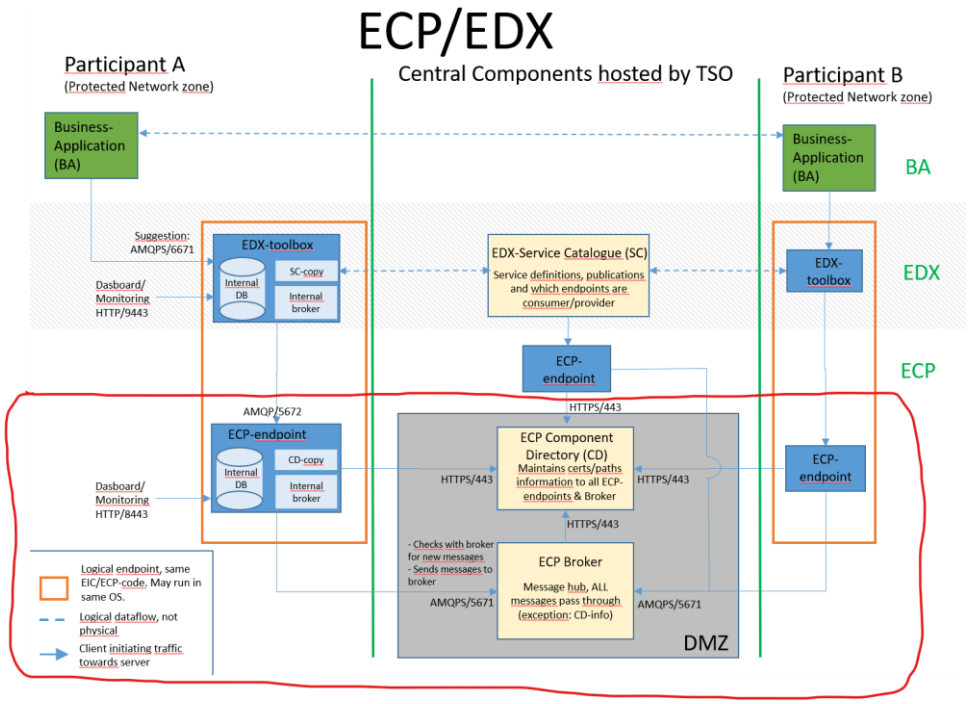
The CD keeps track of all the public parts of these certificates and let every Component in the ECP-network know about them. If a certificate is renewed, all Components will know within minutes.

2.2 Certificate creation, distribution and usage in messaging

Below you see 3 processes: The blue process is "certificate creation"; how to register a new EP with 3 certs. The green process is showing how certificates are distributed (called 'synchronization' in ECP) over HTTPS (using AUTH cert). The red process is how messages are sent from A to B (using AUTH-cert for AMQPS, and ENC+SIGN for A-to-B messaging).



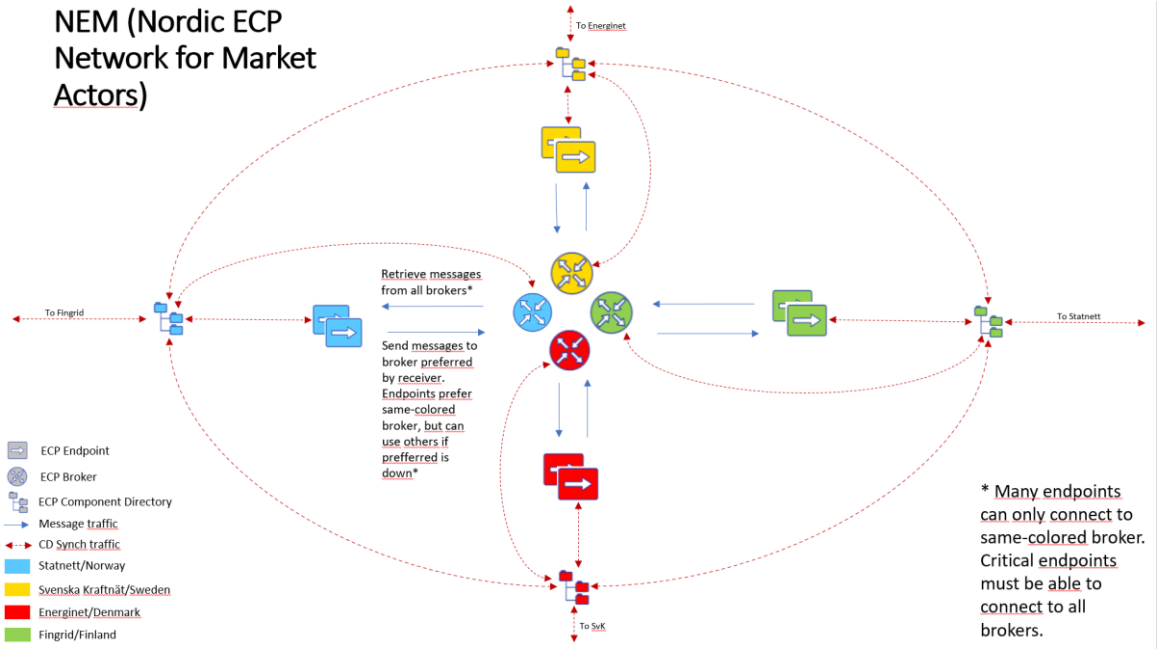
The drawing above is similar to the drawing in the NEX Installation Guide (shown below) but focus only on part of it (shown with red ring).



Most problems related to certificates come down to AUTH-certificates not being renewed (expired) or not distributed.

2.3 NEM – Nordic ECP Network for Market – multiple CD/BR

In NEM we have multiple CDs and BRs, which makes things a little bit more complicated. One CD is responsible for the renewal of one set of EPs (Statnett is responsible for Norwegian EP, SvK for Swedish EP, etc). These CDs synchronize all Component-information with each other every minute. And each endpoint can connect to all BRs if they want. This drawing gives an overview:

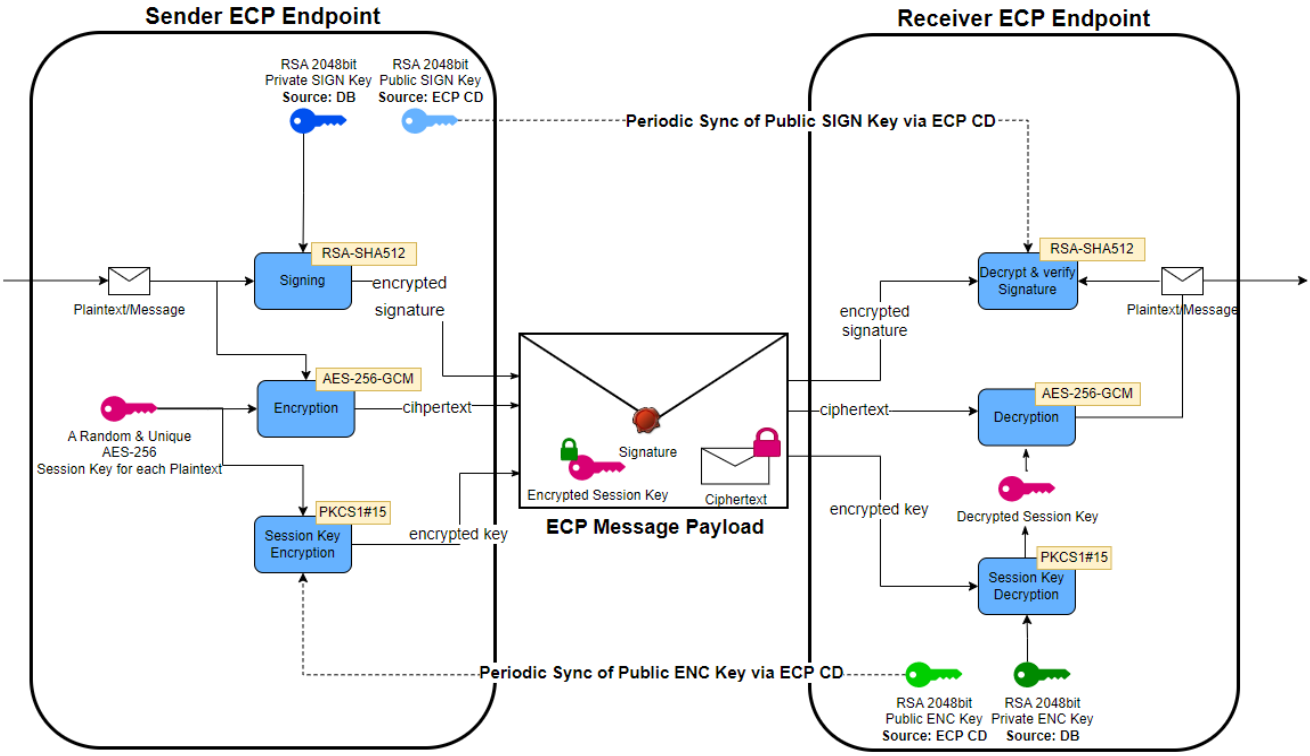


Pay close attention to the arrows. The red arrows show the CD-synchronization traffic, which is only about exchanging certificate information and other type of meta data. The blue arrows show the actual message exchange, but to avoid clutter not all possible combinations from endpoint to brokers are shown. Also note that the CDs will synch information about all endpoints to all other CDs.

2.4 End-to-end Message encryption

Having covered authentication certificate usage, we now focus on message encryption and signing. With the above explanation it's possible to see how your endpoint could be fully connected to a CD and BR, but still have the wrong certificates for a remote EP. This will cause a problem when trying to decrypt a message from that remote EP.

It's not necessary to understand the drawing below fully, but it is a useful reference to show how the certificates are being used. In some special cases it can be necessary to understand this to determine which certificate is missing. The whole scheme is based upon RSA + AES.



- You can disregard the pink key for the time being – it is created for every message (=session) and does not concern us with regards to the SIGN/ENC-certificates.
- The blue keys are the private and public SIGN certificates and shows where they are used. They are used to convince the receiver of the sender's identity.
- The green keys are the private and public ENC certificates. They are used to protect the content from being visible from any other than the receiver. Even the sender cannot decrypt the message.
- Note lock-symbol on the pink session key in the envelope in the middle. It shows how the ENC key is being used to encrypt the pink key, and the pink key is in turn used to encrypt the actual content. The reason for this apparent unnecessary complication is performance.

3 Core Concept: Message Path (MP)

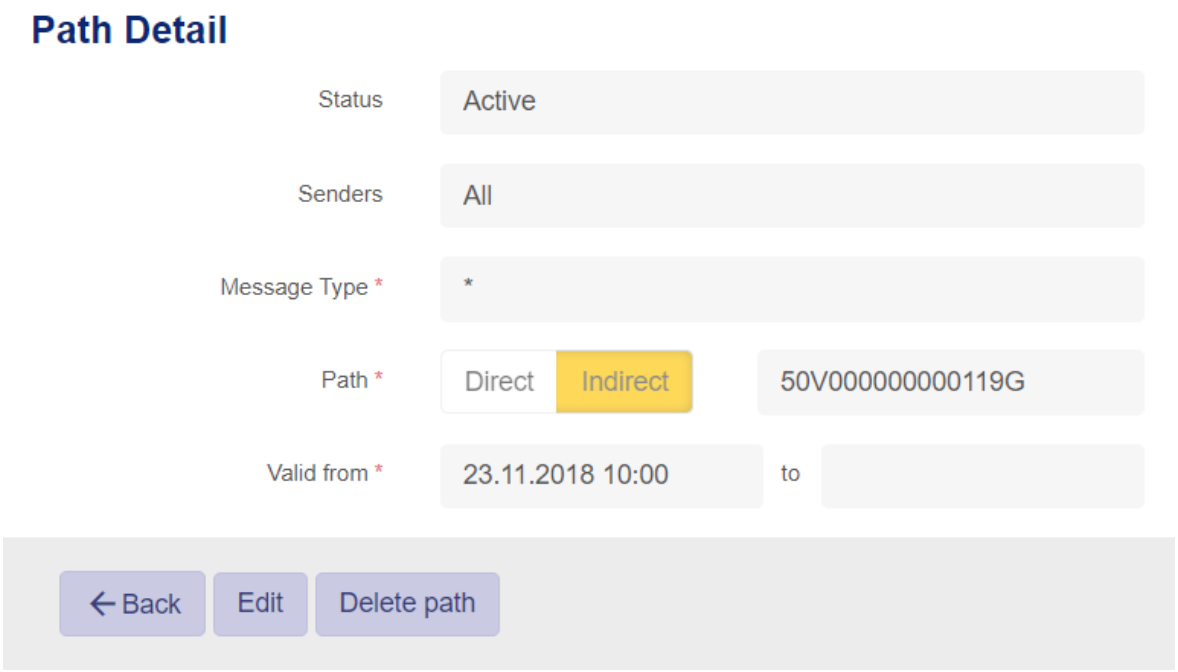
3.1 Definition

A Message Path (MP) is used for an EP to tell how it can be reached by other EPs; a routing mechanism in other words. A formal definition is something like this:

An MP is applied to a certain EP to tell:

- Which Sender/Remote EP is allowed to use the MP to reach the EP
- Which Message Types (MT) is allowed to be sent to the EP
- Which BR to use in order to reach the EP
- For which timespan this path is valid

The screenshot shows a typical MP where every EP is allowed to use all kinds of MT to reach this particular EP (code is not shown here) using the BR 50V000000000119G.

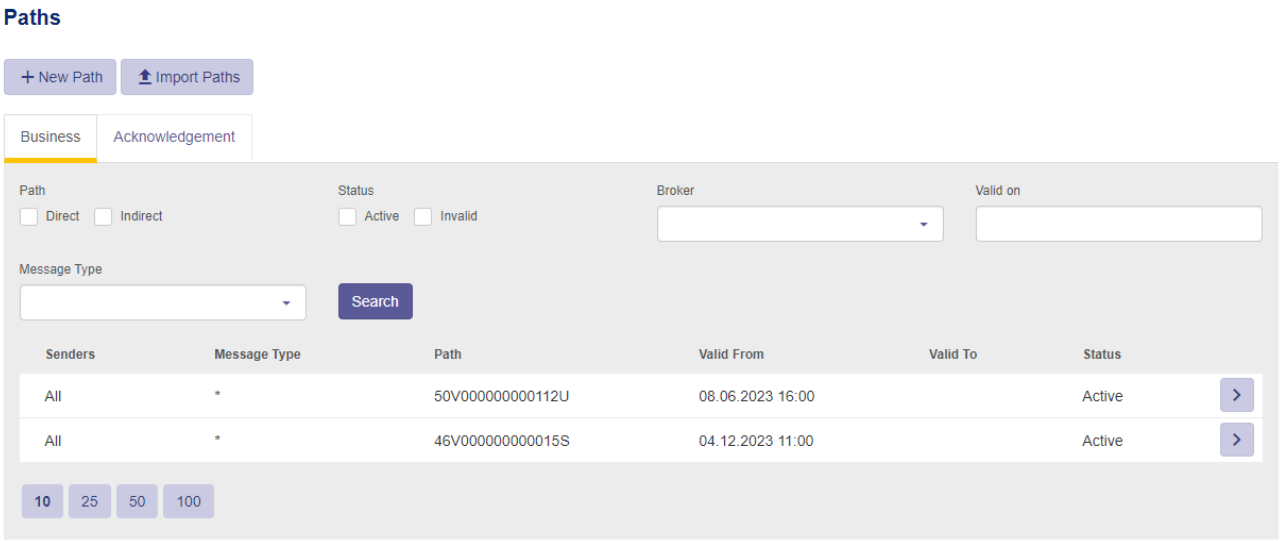


What you must understand about MP is this:

- The MPs ONLY tells you how to reach a specific EP
- In your own EP:
  - You specify NO instruction on how to send to others
  - You specify ONLY how your own EP will be reached
- The MPs you create will be synchronized with the CD every minute
- Consequently, your EP will receive all MPs for all other EP every minute
- Your EP will lookup other EP's MP to determine how to send to them

3.2 Multiple MP

You can create more than one MP. The best reason for doing so is failover. See this example which is considered a standard setup for an endpoint connected to Statnett in NEM-TEST.



In this situation, if a remote EP sends to this EP, it will use the first MP in the list. But if BR 50V000000000112U is down/not connected, then the remote EP will try the next MP which uses another BR (46V…15S).

One could create quite complex and quite many MPs if one wanted to. That is not recommended. All EP should follow the guidelines from NEX (stated in the NEX Installation Guide).

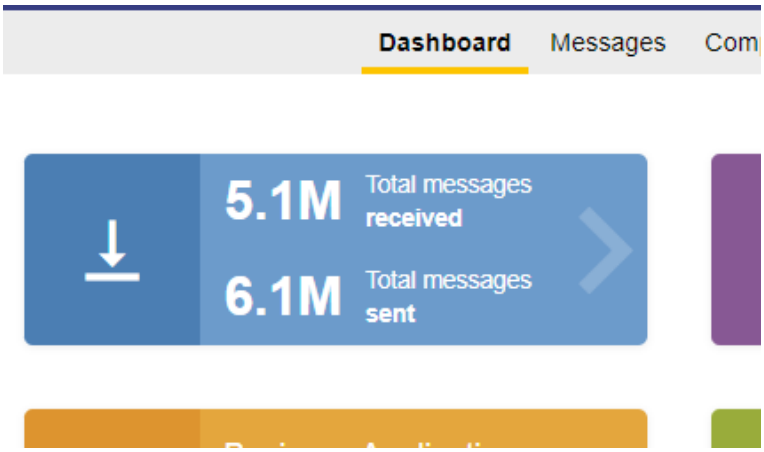
3.3 Broker connectivity

The consequence of MP system described earlier, and the fact that we have multiple BR (and CD) in NEM, is that EP-A **might** send a message to EP-B through BR-B, but when EP-B responds a sends a message back to EP-A it **might** go through BR-A! The traffic thus can pass through two different brokers on it's way back and forth.

This will happen when

- Messaging between EPs listed in two different CD (= different TSO/BR). Study the drawing of NEM in chapter 2.3 and imagine how to EPs registered in two different TSO will communicate.
- Failover occurs

Your EP will keep track of all the BRs it has sent/received messages through at some point. You will find this overview by clicking in the ECP GUI Dashboard:



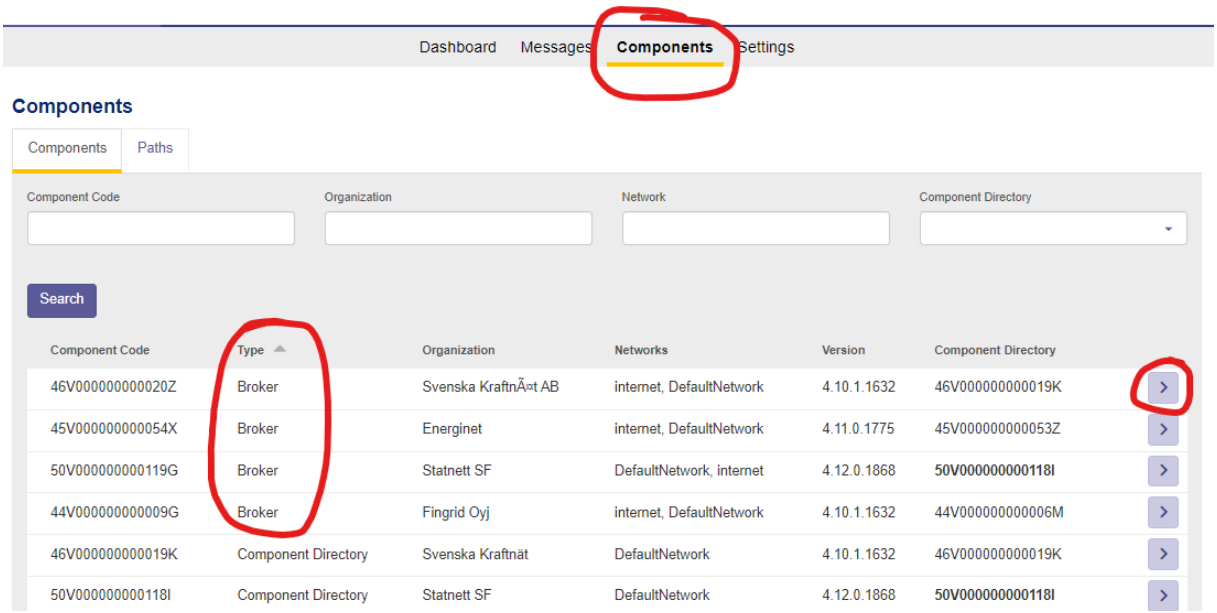
Message statistics

Component code	Connected	Sum message in	Last operation in	Sum messages out	Last operation out
50V000000000119G	true	6.9M	16.03.2024 17:04	6.3M	16.03.2024 17:04
46V000000000020Z	true	118	30.08.2023 06:20	2.8M	14.03.2024 10:47
45V000000000054X	true	0		99.0k	16.03.2024 17:00
44V000000000009G	true	0		112.0k	16.03.2024 17:00

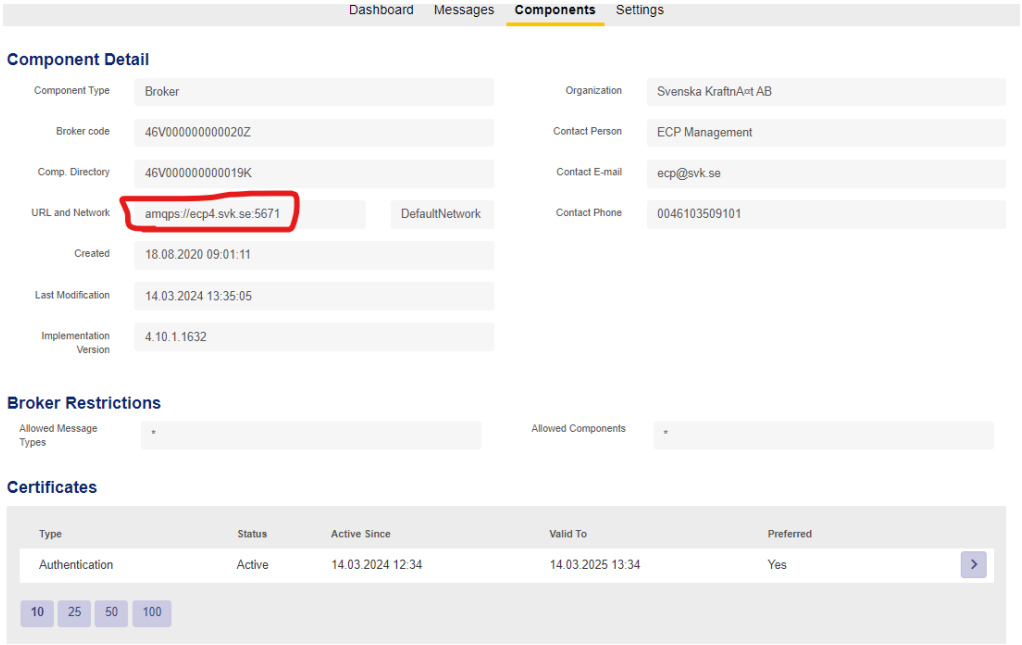
If the Connected-status is true, that means that the EP will be able to **download (receive/in)** messages to that broker. **However!** The status can be true, but still not possible to **upload (send/out)** message from that same broker because sometimes the EP loses 1 out of the necessary 2 connections! You can check this by issuing a standard command "netstat -an" (Linux) or "netstat -a -n" (Windows) and then search for connections to the BR which almost always are using the port 5671 (AMQPS):

```
[                                    ]$ sudo netstat -anp | grep 5671 | sort -n -k5
tcp6       0      0                .31.240:44120     52.233.244.188:5671     ESTABLISHED 7620/java
tcp6       0      0                .31.240:49088     52.233.244.188:5671     ESTABLISHED 7620/java
tcp6       0      0                .31.240:50754     192.121.1.148:5671      ESTABLISHED 7620/java
tcp6       0      0                .31.240:35834     195.204.145.170:5671    ESTABLISHED 7620/java
tcp6       0     90                .31.240:59708     195.204.145.170:5671    ESTABLISHED 7620/java
tcp6       0      0                .31.240:35998     195.234.135.44:5671     ESTABLISHED 7620/java
tcp6       0      0                .31.240:36006     195.234.135.44:5671     ESTABLISHED 7620/java
```

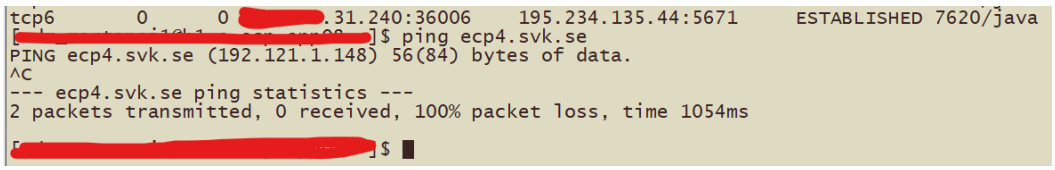
By studying the two screenshots we see that we have one missing connection to 192.121.148. To answer the question, which BR is behind IP, one must look in the Components List:



Then open for example the component details for BR 46V….20Z and review the information there:



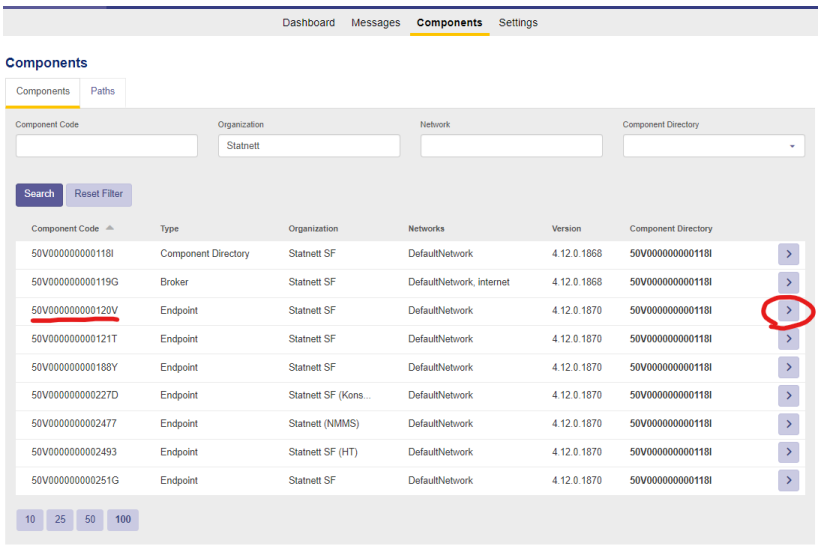
Then check if that matches with IP:



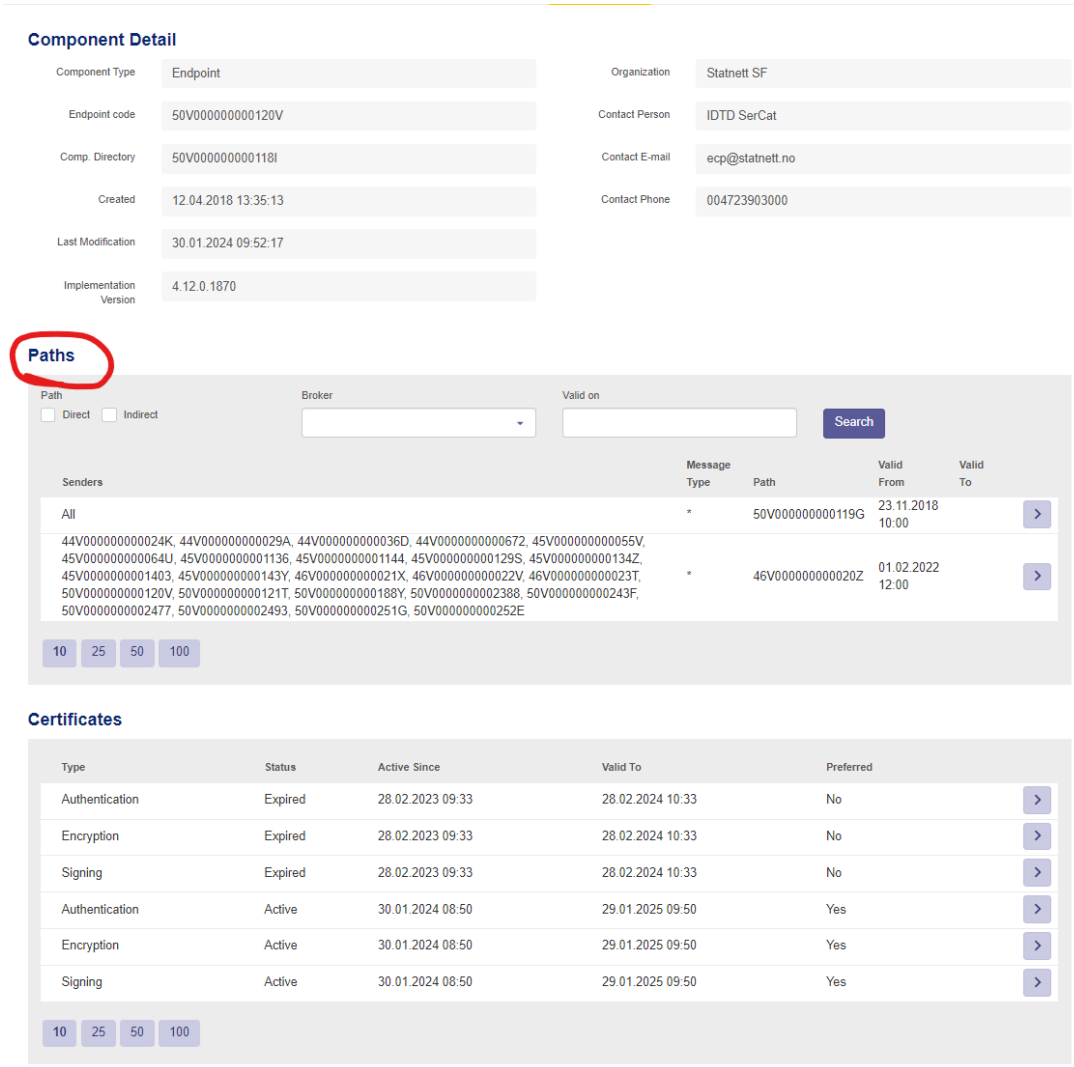
At this point we know that our EP may not able to upload to that 46V…20Z. But this will quite often be fixed as soon as we sent a message. If not, then restart the ECP-endpoint.

3.4 Sending to a remote EP

To understand which BR is used when sending to a remote EP open the Components List and find the EP you are sending to (ex 50V…120V):



Then check the details and the Message Path listed:



The first MP will be used, if the EP lists this BR as "Connected = true" (see screenshot in previous sub chapter). If not, the next MP will be used. Always look out for MPs that are not yet valid.

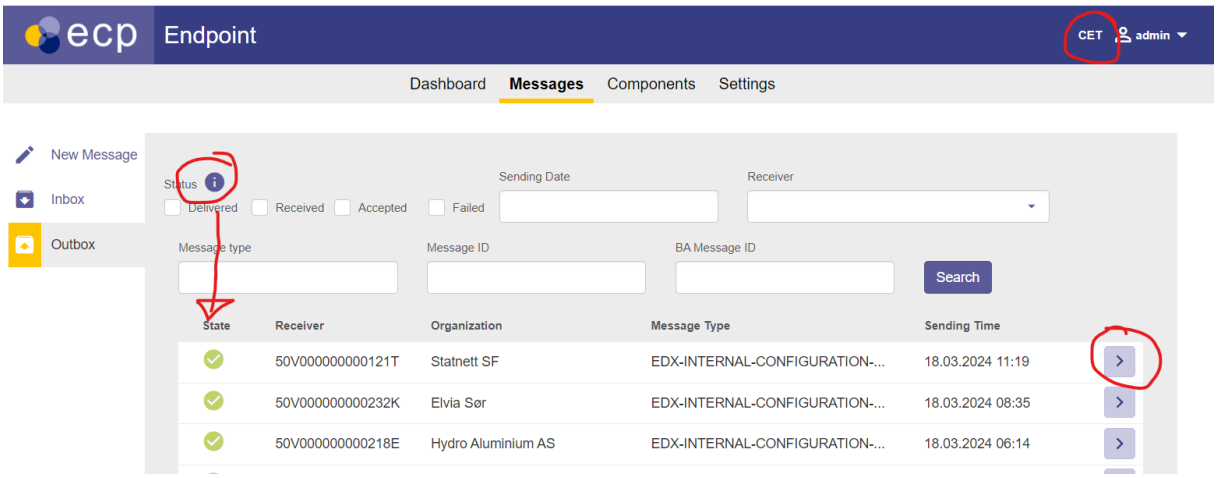
Armed with this knowledge it will now be easier to troubleshoot ECP-issues.

4 "Something is wrong with ECP"

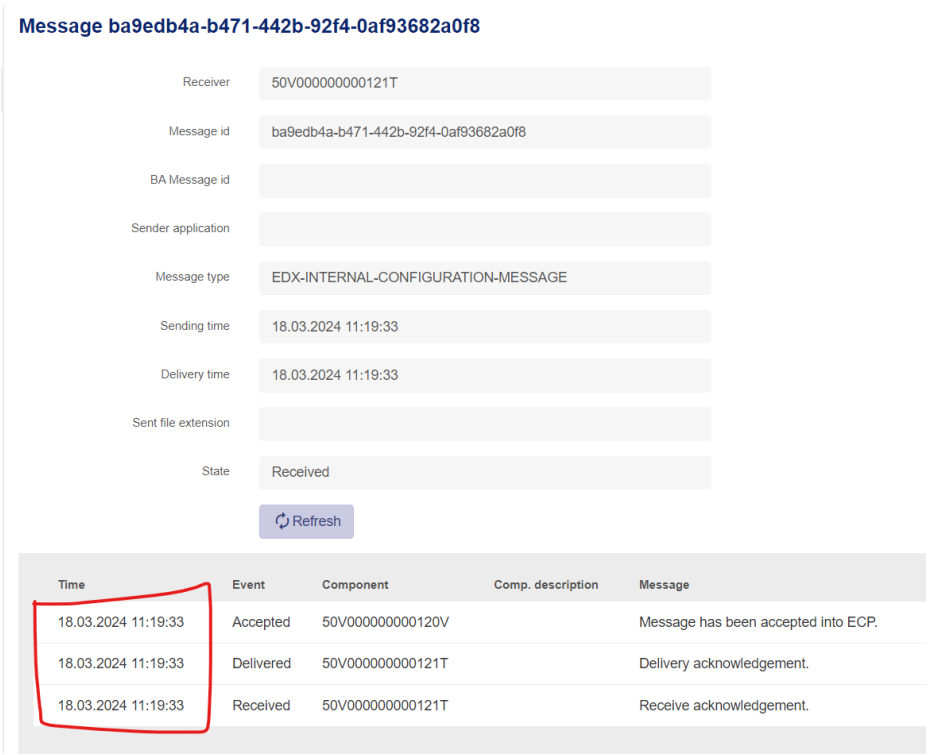
Quite often one can hear the statement mentioned or variations thereof. The first step is to clarify if that statement is correct, because often it is not the source of the problem.

4.1 Investigation of Message Status

Go to ECP-GUI and check Message Status in the Outbox:



Click the (i) button to get information about the various states. The main take-away there is that if you have RECEIVED (green check-mark shown above) or DELIVERED state (yellow-brown), then you know that the remote EP did receive your message. If status is ACCEPTED (also green) then you **cannot** know whether the problem is in your EP or somewhere else. Thus you must continue investigation in the next sub-chapters. Look closely at the timestamps and the timezone-setting (upper right corner) which should be CET. You could also study the details of the particular message:



This shows the timing of the various states, so it can shown if there has been a delay from the time the message was sent (Accepted – first row) and when the remote EP responded with Delivered/Received ACK (second/third row).

Investigating the Inbox part is not so useful, since you cannot see messages that you haven't received. However, by looking at the timestamps or searching for certain Message Type, you can perhaps deduce that there is problems with receiving/downloading message. In that case read the next sub-chapters.

4.2 Investigation of CD connectivity

Go to ECP-GUI-Settings and click on "Check Connectivity" under "Component Directory". If status is OK, then you know that your endpoint is connected and synchronized with the CD (it synchronizes every minute). That means that your endpoint will know all the **public certificates** and the **message paths** of all the other endpoints in the network (see previous Core Concept chapters for more explanation of these terms).

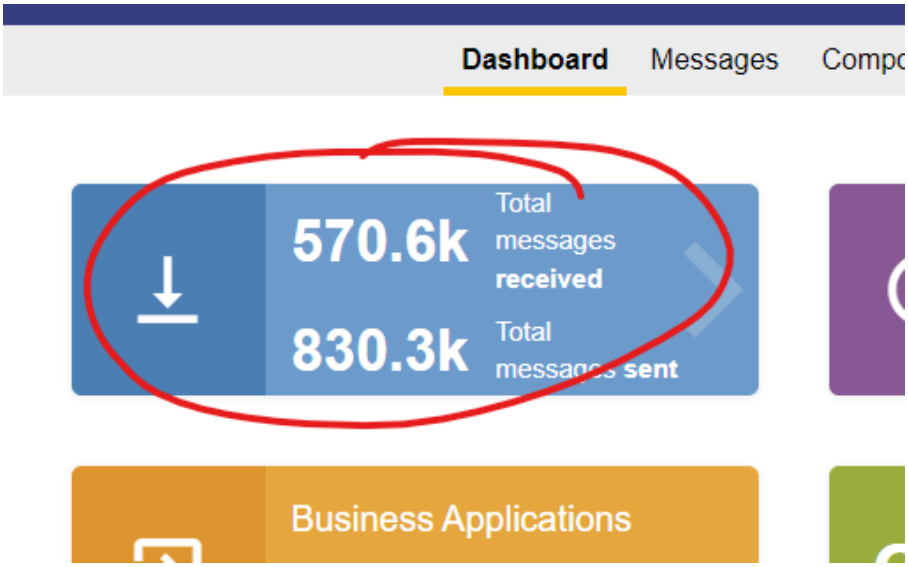
If status is not OK, then you should check the file ecp.log to see error messages related to traffic towards the CD. Even if you **do not** have connection to the CD, the message traffic may still flow well, because you have a local copy of the CD. If connection to the CD has been away for many hours/days, then your CD-copy may be outdated and hold expired certificates. Up until v4.12 of ECP new certificates (in the whole ECP-network) is being preferred as soon as they are created, thus a problem could occur when certificates are not synchronized (copied) throughout the network through the CD. Later versions will improve this, to make CD-downtime an almost non-existent problem.

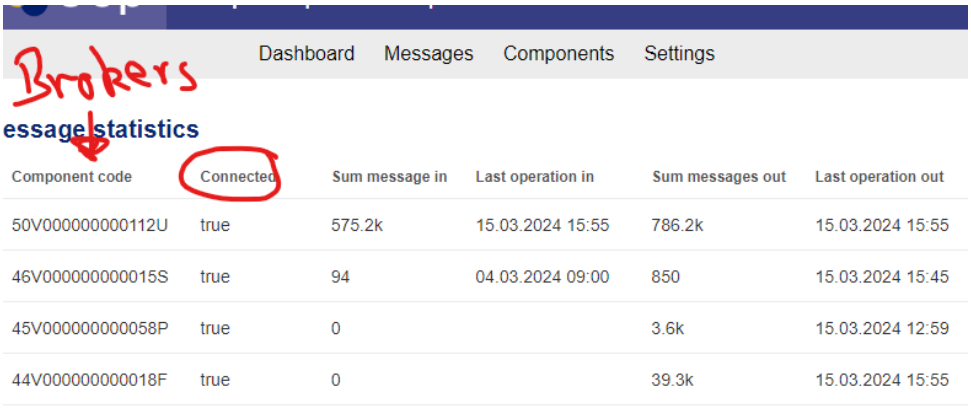
If you do not have connection to the CD it falls into two categories:

- Network problems (firewall? - see NEX Installation Guide)
- Certificate problems (see chapter 6)

4.3 Investigation of Message Connectivity

Go to ECP-GUI-Dashboard->Click on the Blue tile:





The list shows (one or more) ECP Central Brokers your endpoint is connected to. If all have Connected = true, your endpoint is most likely connected. To examine this thoroughly and avoid any doubt, please read chapter 3.3.

Go to ECP-GUI-Setting and click on "Check Connectivity" under "Messaging Connectivity". Choose the endpoint you want to send to and specify TEST as Message Type. Then send. If you get "OK", then you know that ECP to ECP traffic is working well.

If you get "NOT CONNECTED" you can check the file ecp.log to search for reasons, but you can also try to send to other ECP-endpoints. This type of testing will not show up in ECP GUI of the receiving endpoint, so it is totally fine to use for testing. If some endpoints respond, then you know that your ECP-endpoint works fine, but some other endpoints fail.

You can also check in ECP-GUI-Components -> Choose details on the receiving component. Here you can see if that endpoints has defined Message Paths. The message path tells which broker your endpoint must connect to, to send to it an other endpoint. Then go to ECP-GUI-Dashboard -> Click on blue tile. Check status on the various brokers you're connected to and make sure you can connect to the broker mentioned in the Message Path.

If you have identified that your EP is not connected to the necessary BR (where you expect to send/receive a message), then it's basically the same as for CD connectivity problems, but it could also be problems related to

- Network problems (firewall? - see NEX Installation Guide).
- Certificate problems (see chapter 6)

4.4 Investigate Internal Broker

Each EP has its own "Internal Broker" (IBR). This where the EP stores the messages which are in transit, either outgoing (sending) or incoming (receiving). The number one problem with the IBR is that it can run full. And this can be caused by just a few messages. Check ecp.log to see if there are error messages related to something being 100% full or "flooded". Another way to check is to examine the activemq-*.xml governing the IBR in your EP. These files are found among your configuration files for the EP. There are many of those, most likely all the same setup – and you can search for " <storeUsage>" tag. This tag will show a limit to how much space the IBR can take. Usually in a percent of the available space on the disk. The IBR itself can be found in a similar place like this:

```
[                    data]# pwd
/var/lib/ecp-endpoint/internalBroker/data
[                    data]# ls -l
total 4424
-rw-r-----. 1 ecp-endpoint ecp-endpoint 33554432 Mar 16 19:30 db-69101.log
-rw-r-----. 1 ecp-endpoint ecp-endpoint  2220032 Mar 16 19:30 db.data
-rw-r-----. 1 ecp-endpoint ecp-endpoint   279064 Mar 16 19:30 db.redo
-rw-r-----. 1 ecp-endpoint ecp-endpoint        8 Mar 14 10:47 lock
[                    data]#
```

If you think some queue is full, then go to chapter 7.

4.5 Investigate OS Resources

4.5.1 High CPU/IO-wait

EP is very CPU & I/O intensive. 10 msg/sec is rule of thumb for maximum limit given the hardware recommendation given in NEX Installation Guide. Approaching this limit you may experience problems with throughput, delays and queues filling up. No endpoint in NEM has this level of traffic on average, but if something goes wrong or is halted for a time, then you can get a huge "burst" (ketchup bottle effect).

You can inspect the traffic pattern in several ways:

- Check ECP GUI and count messages pr minute in/out
- Check ecp.log, identify patterns in the log – count such patterns (with some grep-tool)
- Use ekit.jar to analyze ECP/EDX logs (see chapter 11.3.1)
- Setup a Grafana dashboard (see chapter 11.2)

4.5.2 High CPU

Another reason why CPU is high, while I/O might not be high, could be a background job in EP. You can list the background jobs if you go to ECP GUI Dashboard and click on the "Jobs" tile or go to Settings-page and locate the Jobs-section. It looks like this:

Background Jobs

Job	Status	Start	End	Duration	Failure
Synchronization	Completed	18.03.2024 09:58:15.001	18.03.2024 09:58:18.315	3314 ms	
Message Deleting	Completed	18.03.2024 09:14:45.493	18.03.2024 09:14:45.670	177 ms	
Registration Status Checker	Completed	18.03.2024 09:59:00.003	18.03.2024 09:59:00.131	128 ms	
Statistics Synchronization	Completed	18.03.2024 09:58:30.001	18.03.2024 09:58:30.105	104 ms	
Message Path Synchronization	Completed	18.03.2024 09:58:45.001	18.03.2024 09:58:45.048	46 ms	
Cluster Status Manager	Completed	18.03.2024 09:59:13.001	18.03.2024 09:59:13.003	1 ms	
Certificate Renewal	Completed	18.03.2024 09:50:00.001	18.03.2024 09:50:00.002	1 ms	
Expired Certificates Cleaning	Completed	18.03.2024 09:10:00.002	18.03.2024 09:10:00.004	1 ms	
Routes Management	Completed	18.03.2024 09:59:00.001	18.03.2024 09:59:00.002	1 ms	
Configuration Reload	Completed	18.03.2024 09:59:10.000	18.03.2024 09:59:10.001	0 ms	
Message Expiration	Completed	18.03.2024 09:13:49.232	18.03.2024 09:13:49.232	0 ms	
Database Compression	Disabled				
Directory Cache Reload	Waiting for run				
Database Message Compression	Disabled				

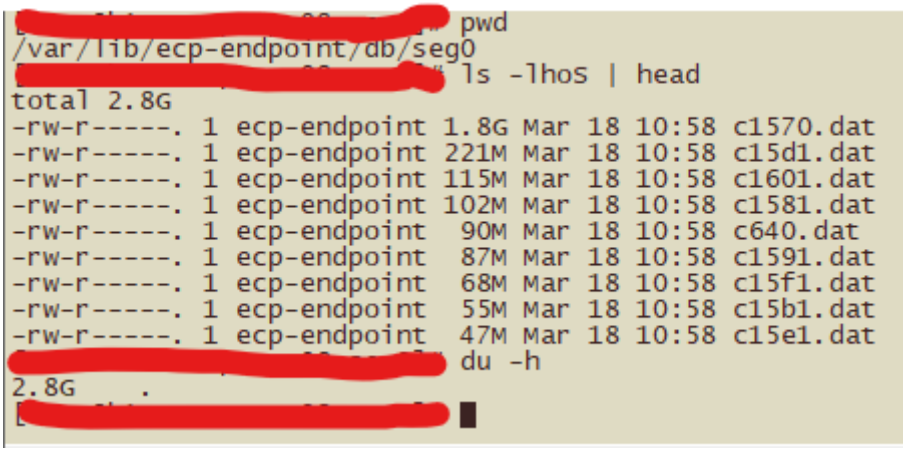
10 25

Here we show all jobs sorted by "Duration". The jobs that is most likely to take lot of resources are those related to the database, marked with red: "Message Deleting" and "Message Expiration". Database Compression should be disabled if you've followed the NEX Installation Guide. The database keeps track of all messages sent/received and is what you see in Outbox/Inbox in the GUI. If the database has grown very large, or if you have changed some setting regarding message expiration or message retention (see Unicorn ECP Administration Guide), then these jobs have been known to run "forever". But, the list above shows only the completed jobs, not the currently running jobs. If a job has been running for a long time but not completed yet, thus consuming a lot of CPU, it will not show in the table above!

The default behaviour of "Message Deleting" and "Message Expiration" is to run once every hour. If the timestamp show for last start/end is older than 1 hour, then you should think about reducing the database. In that case go to chapter 8.

4.5.3 Disk utilization

If disk is filling up you should check the size of the database by checking the size of the db-folder "seg0":



This database contains 224K message-entries (not the message payload, only metadata), which gives approximately 12KB pr msg which is good rule of thumb. If you have reason to believe that the disk usage is much higher than 12KB pr msg, database compression could be performed – see chapter 8.1.

5 "Something is wrong with EDX"

This is not often heard, since EDX is not so widely known. But, if everything checks out fine with ECP, but still message flow is interrupted, then it is time to look at EDX. The problems you may encounter here are usually one of these:

1. EDX does not connect to the ECP Internal Broker (see last drawing in chapter 2.2 where EDX-Toolbox is connected to ECP Endpoint over AMQP).
2. EDX does not have a EDX Service Catalogue (SC) copy
3. EDX does not have the correct EDX Service Catalogue (SC) copy (there could be multiple)
4. EDX has received an SC with errors in it (the TSO is responsible)
5. EDX Internal Broker (IBR) is full (this is another IBR than for ECP)
6. EDX does not route message to the correct BA
7. EDX or BA has terminated connection, perhaps because some expired cert

Of these, only 5 is something we would expect to happen without any reconfiguring/restarting, although all of them can happen "out of the blue" given the right circumstances.

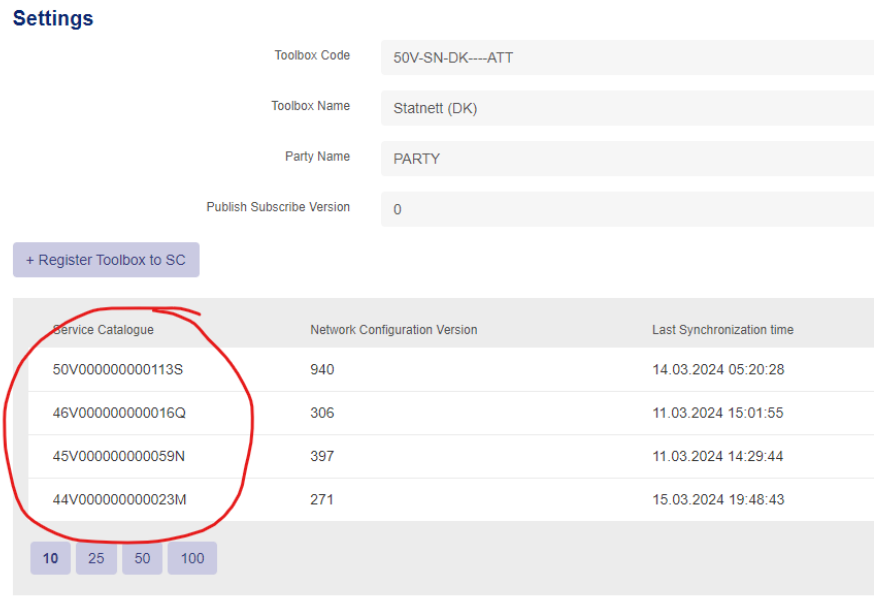
5.1 EDX to ECP IBR connectivity

Go back to NEX Installation Guide and check both ecp.properties and edx.properties carefully to see that edx.properties for ecpBroker matches the settings you have in ecp.properties for internalBroker. The EDX is a client to the ECP IBR. Be especially careful with ecp.broker.url in edx.properties – it alone determine whether EDX will try to use AMQPS or AMQP.

If you have decided to go against the advice in NEX Installation Guide and use AMQPS for this connection, check if the certificate used for the connection has expired or not.

5.2 EDX SC

An EDX might be registered in multiple SC. In the following we assume that BA are addressing correctly (see https://ediel.org/wp-content/uploads/2023/10/NEX-Addressing-Guidelines.pdf), so the problem is reduced to having the correct EDX SC. This shows an EDX registered in 4 SC (EDX GUI – Settings):



If no SC is listed, then it must be fixed. If the EDX is new, is very likely that the responsible TSO has not added the EDX to the EDX SC – you could try to notify the TSO. Another option is that the code you've specified for the property edx.serviceCatalogue.code in edx.property is incorrect. Check NEX Installation Guide again.

Once you made a change or believe the TSO has done a change, restart EDX. This will force it to request a SC-copy again upon startup. You can see the message being sent and possibly also the response in ECP-GUI Outbox/Inbox.

If you do receive a response in ECP-Inbox, you can further track the parsing of this response in the file edx.log. If something goes wrong with the parsing of the SC, you can see it there.

At this point you should have at least one default SC listed in Settings.

You may need more than the default SC, because your EDX access EDX-services and toolboxes outside of the default SC. If you know/suspect that you are missing some SC in your list, then you must contact that TSO which is responsible – there is no way to add it manually as of now, but the TSO can trigger their SC to be sent to your EDX.

The TSO can make errors in the SC and they will be propagated to every EDX within minutes. One mistake could be to remove your EP/EDX-code from the SC. Other mistakes could be to change some addressing information within the SC. Notify the TSO in this case.

5.3 Investigate Internal Broker

Each EDX has its own "Internal Broker" (IBR). This is where the EDX stores the messages which are in transit, either outgoing (sending) or incoming (receiving). The number one problem with the IBR is that it can run full. And this can be caused by just a few messages. Check edx.log to see if there are error messages related to something being 100% full or "flooded". Another way to check is to examine the **jms-context-nonha.xml** governing the IBR in your EDX. This file is found in <EDX-config-folder/jms/ or <EDX-config-folder/jms/secured, depending on the edx.properties setting of internalBroker.useAuthentication=false/true. In this file search for " <storeUsage>" tag, it will determine how much disk IBR is allowed to take, usually as a percent of the available space on the disk. You can change it to a specific limit to have more control over the space set aside for the IBR. Here is an example from Statnett-configuration:

```
<systemUsage>
        <systemUsage>
                <memoryUsage>
                        <!-- in-memory buffer for messages -->
                        <memoryUsage limit="250 mb" />
                </memoryUsage>
                <storeUsage>
                        <!-- storage buffer for messages -->
                        <storeUsage limit="1000 mb"/>
                </storeUsage>
                <tempUsage>
                        <tempUsage limit="200 mb" />
                </tempUsage>
        </systemUsage>
</systemUsage>
```

The IBR itself can be found in a similar place like this:

```
[redacted]]# pwd
/var/lib/edx-toolbox/edx-activemq-data/data
[redacted]]# ls -lh
total 4.5M
-rw-r-----. 1 edx-toolbox edx-toolbox  32M Mar 16 20:37 db-8587.log
-rw-r-----. 1 edx-toolbox edx-toolbox 1.1M Mar 16 20:37 db.data
-rw-r-----. 1 edx-toolbox edx-toolbox 301K Mar 16 20:37 db.redo
-rw-r-----. 1 edx-toolbox edx-toolbox    8 Mar 14 10:47 lock
[redacted]]#
```

If you think some queue is full, then go to chapter 7.

5.4 Investigate OS Resources

The problems for EDX is almost identical to that of EP (ECP) – see chapter 4.5, but apply that chapter to EDX logs/jobs/database.

5.5 EDX Routing

The edx.yml contains information on how to route to correct BA (or rather to correct "integration channel"). This configuration is described in NEX Installation Guide in some detail. However, if the SC changes **and** you have routing based on "Service", then it is possible that the routing fails and the message is sent to "receiveProcessDefaultRoute" (usually never to be seen again!). This is very likely scenario for most EDX setups. If you route based on Message Type (which is not so much recommended) then it is more likely that the routing fails because the BA that sends the message has changed the Message Type.

To understand how the routing is being executed you must study the edx.log carefully. Check also for "RoutesConfig" log lines during startup of EDX, because they tell which routes are actually active. Sometime an error in edx.yml can lead to that no routes are read/used/applied.

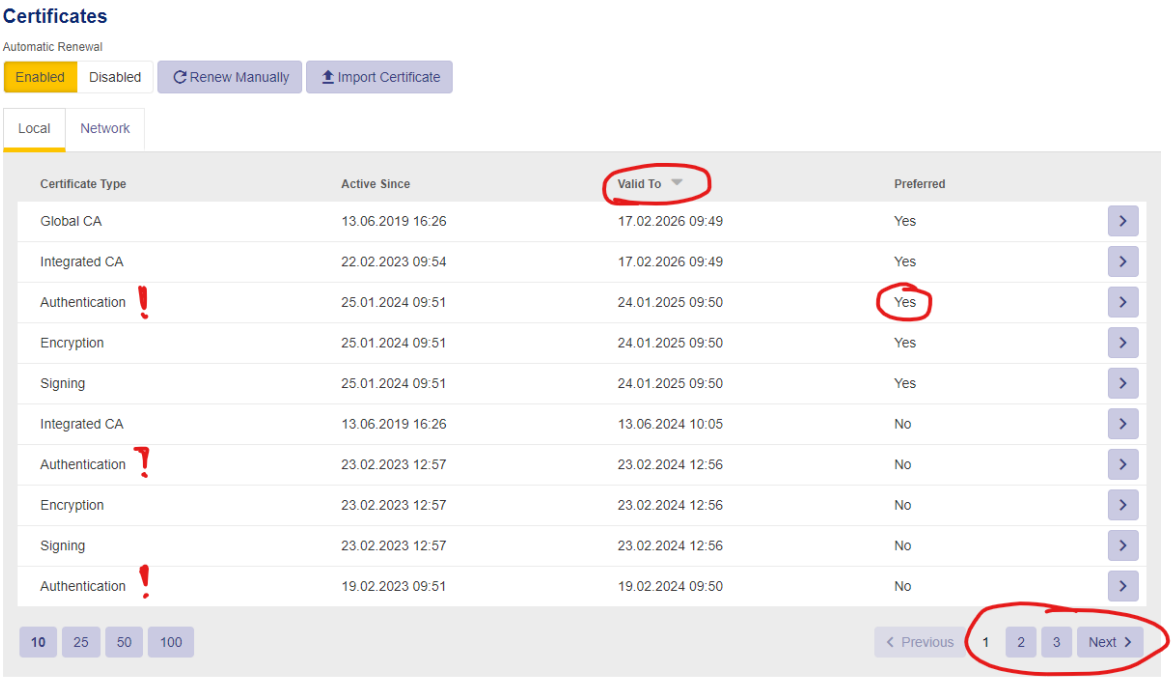
5.6 EDX BA Connectivity

It is recommended to have AMQPS or otherwise TLS-communication between BA and EDX. This can lead to break in communication because of expired cert. There is of course a whole range of possible issues here, but that is outside the scope of this guide.

6 Fix certificates

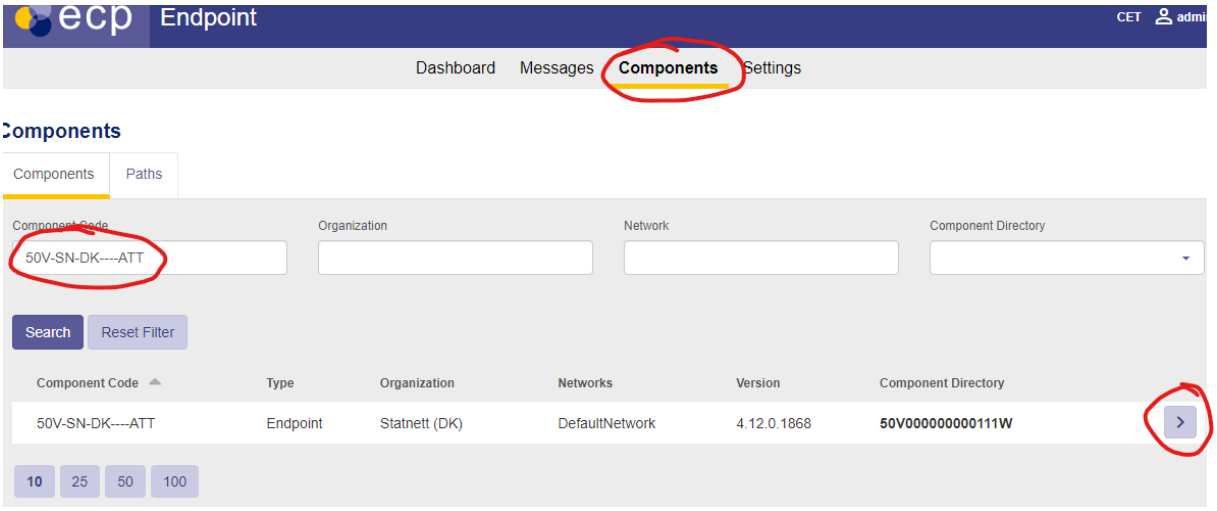
6.1 Comparison

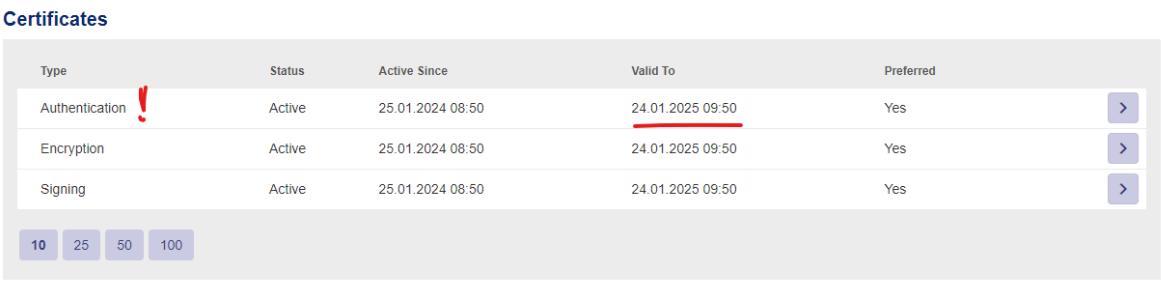
What you can do first is to compare certificate information in your EP with the CD-copy you have. The Core concept chapter about certificates (chapter 2) tries to explain which certificates are created, what they are for and how they are distributed. One basic task is to verify as best as we can that this process has succeeded. We can do that by looking at the ECP GUI Settings page and find the "Certificates"-section:



The thing we focus on first is to find the valid certificates. No expired certificates are useful. Then we identify the Authentication certificates, because this is the certificate used to establish network-connection (HTTPS or AMQPS). One of them is preferred, but there could be some that are not preferred, but still valid – so they can be used.

Next, we check our copy of the CD. Go to ECP GUI Components and find **your own** endpoint code there and click the details-button:





We can verify that the CD has the same Authentication certificate as our own EP, by matching the timestamp. We can now reasonably expect that all other components in the ECP-network will also know our EP's public certificates. **At the very least you can be absolutely convinced that the CD has this information and that the HTTPS-connection to the CD will not be blocked because of certificate issues!** If you don't find a match, then it's still possible that your CD has not been able to synchronize the information back to your ECP. But it's also possible that the CD do not have the certificate, and then communication will never work. You are then forced to do option 5 in next sub-chapter.

If you experience problems with the AMQPS-connection to the BR, and you know the CD has the correct information about your EP and the CD is synchronizing every minute, then two possibilities are left:

- The BR has not received cert-info about your EP from the CD
- The BR has not renewed/published its own certificates to the CD

Check the certificates you have for the BR you're communicating with (you may have to understand MP to know which BR to investigate – see chapter 3) and make sure they are valid. If the certificate is not about to expire (renewal happens 30 days before expiry in NEM), then most likely the connection problem is not due to certificate mismatch.

6.2 Options

If you find in ecp.log problems indicating a "certificate problem" (SSL-handshake, "expired"-messages, failed-message due to signature etc) you can try to go through this list. The goal is to make sure the CD has received the certificates from our EP.

Each option can potentially solve the issue. BUT! Option 4 and 5 will renew your own certificate. Option 5 also requires assistance from TSO. Options 4 and 5 should not be executed unless you are pretty sure that it is your own ECP-endpoint that is to blame. It could very well be the other ECP-endpoint or the central ECP-broker that has outdated information from the CD. In those cases, no option will help, and 4 and 5 should be avoided.

1. Restart ECP-endpoint. Test message flow.
2. Go to ECP-GUI-Settings.
    a. Push configuration. If it fails, it's useful to divide into two categories:
        - No matching certificates between your EP and the CD. Read chapter 6.1.
        - Something else: Firewall/DPI, Routing/DNS/Hardware, CD-downtime
    b. Restart ECP-endpoint
    c. Wait 3 minutes and test message flow
3. Control that your firewall is not interfering (deep packet inspection) with the SSL-traffic between ECP-endpoint and/or CD/Broker. Check by running something similar to this command (openssl s_client -connect ecp4.statnett.no:5671 -showcerts). The firewall must

not touch/change anything in SSL-traffic. You can get hold of openssl for Windows by downloading Git for Windows (https://git-scm.com/download/win), openssl is part of the "Git Bash" shell.

4. Go to ECP-GUI-Settings. Renew Manually. Wait 3 minutes and test message flow. This option should usually not be helpful, because either you're are connected to the CD and have valid certificates or not. If it's the latter, then it will not be possible to renew manually – you can only renew if you have existing valid certificates.
5. Re-register: Go to ECP-GUI-Settings and press button for "Initiate Registration". You **may** need to ask the TSO (see NEX Installation Guide for email address) for a new registration keystore and you **must** ask the TSO (again see NEX Installation Guide for email address) to approve a new registration after you've completed your part.

7 Fix queues

7.1 Purge/Remove messages

To purge/remove messages seems a little drastic, but most serious business procedures must expect an ACK from the BA itself before it considers a message delivered. Therefore it can be an option to purge messages from time to time, to get things running. There are several ways to do it:

7.1.1 The Hawtio option

This can be done manually following these steps:

1. Install Hawtio (see chapter 10.1). Wait 30 sec.
2. Connect to your ECP or EDX GUI but change the URL path to "/hawtio/".
3. The top-menu should show "ActiveMQ" – click on that choice.
4. All queues will be listed, find queues where queue size > 0
5. Consider purging the queue (click on queue – choose "Delete"-submenu – Purge) or delete specific messages.

7.1.2 Delete IBR option

You can delete the folder named "internalBroker" (EP) or "edx-activemq-data" (EDX) and restart EP/EDX. In this case you will lose all messages in transit, but the IBR will be built anew upon restart. This is the "nuke" option, simple and quick.

7.1.3 Ekit option

If you want to purge a specific queue, you can consider using Ekit:

1. Install Ekit (see chapter 10.2)
2. Read documentation on QP tool, by running this command:
   - `java -jar ekit.jar QP`
3. Suggested command for purge of queue QUEUENAME is
   - `java -jar ekit.jar QP -q0 -t0 QUEUENAME amqp://endpoint:password@localhost:5672`

The command could be run scheduled (Linux crontab or Windows scheduling). The example shows how to connect with EP (ECP) where Ekit runs on the same host. But you could also connect to EDX IBR or connect remotely, or via AMQPS.

7.2 Advanced queue protection

The Hawtio option above can be used to remove specific messages, but it's a completely manual process. It's also possible to move a message from one queue to the other and back again. But these procedures does not scale well. Ekit offers more features when it comes to so-called "Queue protection". Install Ekit (see chapter 10.2). See next sub-chapters for use cases:

7.2.1 Remove old messages

If a queue is being consumed, but the client consuming is unstable, you can set Ekit to consume any message that is older than 1 hour (3600 seconds). This operation will run until aborted, because of the -f option.

```
>java -jar ekit.jar QP -f -q0 -t3600 QUEUENAME amqp://endpoint:password@localhost:5672
```

7.2.2 Burst protection

This case has only been tested on ecp.endpoint.download queue, where all messages are coming into EP. If the queue is being flooded with messages from one or more senders, it will delay other messages. You can decide to remove messages from such troublesome senders. Example: if a sender

has produced more than 50 msg in the download-queue and those messages are more than 10 seconds old:

```
>java -jar ekit.jar QP -f -q0 -s50 -t10 ecp.endpoint.download amqp://endpoint:password@localhost:5672
```

Think of this as DoS-protection.

7.2.3 Burst protection and restore

This case has only been tested on ecp.endpoint.download queue. Instead of purging the messages, it's possible to store them to disk and then restore them back to the queue later on. This is of course described in the documentation of the QP tool if you run

```
>java -jar ekit.jar QP
```

One example of such a configuration is where we start to remove all message older than 20 sec when there is more than 10 msg on the queue, but restore them later – that is, as soon as the burst is over (fewer messages than 10 on the queue or no messages older than 20 sec).

```
>java -jar ekit.jar QP -e -f -m msg-buffer -o qp.json -q10 -s0 -t20 ecp.endpoint.download
amqp://endpoint:password@localhost:5672
```

The main idea here is to let through new messages, prioritizing "real-time" traffic over old/queued messages.

8 Fix database

The main fix for the database is to reduce its size. There could of course be more exotic problems which could be resolved with the right SQL, but that is outside the scope of this document for now. You need to log in to the database following the steps in chapter 10.3. Then choose the chapter below you want to execute:

8.1 Compress (no loss, but could be slow)

A compression would remove empty space in the database, but all records will be retained. A compression should usually not be necessary, but if you have reduced the message retention time to reduce the database, compression is needed. The database will never reduce in size by itself!

Compression of ECP-tables:

- `call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('ECP', 'COMPONENT_DIRECTORY', 1);`
- `call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('ECP', 'MESSAGE', 1);`

Compression of EDX-table:

- `call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('EDX', 'TOOLBOX_MESSAGE', 1);`

Warning: This operation can take many minutes if the database has gigabyte size. Be patient.

8.2 Truncate (loss, but very quick)

For EDX there is a set of SQL statements that can be run to quickly truncate the large tables. This will delete all the message records (only metadata/logdata, no payload is lost). Run in order:

Remove foreign keys:

- `alter table EDX.TOOLBOX_MESSAGE drop constraint FK_TOOLBOX_MESSAGE_TOOLBOX_MSG;`
- `alter table EDX.PULL_MESSAGE drop constraint FK_PULL_MESSAGE_TOOLBOX_MSG;`
- `alter table EDX.TOOLBOX_MESSAGE_LOG drop constraint FK_TOOLBOX_MSG_LOG_TOOLBOX_MSG;`
- `alter table EDX.INFLIGHT_EXTERNAL_PROCESSING  drop constraint FK_INFL_EXT_PROC_TOOLBOX_MSG;`

Truncate tables:

- `truncate table EDX.TOOLBOX_MESSAGE_LOG`
- `truncate table EDX.PULL_MESSAGE;`
- `truncate table EDX.INFLIGHT_EXTERNAL_PROCESSING;`
- `truncate table EDX.TOOLBOX_MESSAGE;`

Add foreign keys back:

- `alter table EDX.TOOLBOX_MESSAGE add constraint FK_TOOLBOX_MESSAGE_TOOLBOX_MSG foreign key (PARENT_MESSAGE_ID) references EDX.TOOLBOX_MESSAGE(ID);`
- `alter table EDX.PULL_MESSAGE add constraint FK_PULL_MESSAGE_TOOLBOX_MSG foreign key (TOOLBOX_MESSAGE_FK) references EDX.TOOLBOX_MESSAGE(ID);`
- `alter table EDX.TOOLBOX_MESSAGE_LOG add constraint FK_TOOLBOX_MSG_LOG_TOOLBOX_MSG foreign key (MESSAGE_ID) references EDX.TOOLBOX_MESSAGE(ID);`
- `alter table EDX.INFLIGHT_EXTERNAL_PROCESSING add constraint FK_INFL_EXT_PROC_TOOLBOX_MSG foreign key (MESSAGE_ID) references EDX.TOOLBOX_MESSAGE(ID);`

These commands are very quick, but it is advised to test it first on a test-endpoint. These commands have been tested on EDX 1.13, but will most likely work on older versions of EDX as well.

For EP (ECP) no truncate commands has been explored so far.

9 Reset ECP/EDX from scratch

Apart from re-installing, there is a way to reset the ECP/EDX more or less from scratch. Assuming configuration is otherwise correct, but the system has run into a state which is hard to figure out, then a reset can be useful.

9.1 Hard reset

It consists of two things:

- Delete IBR of both ECP/EDX (see chapter 7.1.2)
- Delete database of both ECP/EDX

Upon restart all these resources will be built anew, but content is of course lost. Deleting IBR will delete messages in transit, that is, messages in the queues. So it can cause a loss of messages. Usually business process can tolerate that.

To delete the databases then delete "db"-folders under ECP/EDX-installation. However, to recover from this can take time:

For EP:

- No longer be connected to the CD, all private certificate information is lost
- Re-registration must be performed (see NEX Installation Guide) and the **TSO must approve manually**.

For EDX:

- No copy of the necessary Service Catalogues, without them EDX will not work at all.
- When EDX starts it will request the default SC (see edx.properties 'edx.serviceCatalogue.code'). Until the SC-response has been received by EDX, all messages that is processed will Fail.
- If your EDX depend on other SCs than the default SC then you must ask the TSOs to trigger the SC to be sent to you. **This is a manual process.**

As you can see, this is not really a good option, only a last resort.

9.2 Soft reset

This may not remove all problems in the database, but otherwise it may often solve problems.

- Delete IBR of both ECP/EDX (see chapter 7.1.2)
- Truncate EDX (see chapter 8.2)
- Compress ECP (see chapter 8.1)

10 Tools

10.1 How to install Hawtio

1. Add/change the property "spring.jmx.enabled=true" to ecp.properties and edx.properties. This will make it possible for Hawtio to see the queues. Restart ECP or EDX if change was necessary.
2. Download hawtio: https://repo1.maven.org/maven2/io/hawt/hawtio-web/1.5.11/hawtio-web-1.5.11.war
3. Rename the file to "hawtio.war" and place the file in the webapps-folder of ECP and/or EDX – it will automatically install.
4. After you've done using Hawtio you should remove the war-file – Hawtio is a liability security-wise. You should also consider reversing the jmx-settings introduced in 1.

10.2 How to install Ekit

EKit is short for "ECCoSP Toolkit" and is a software developed by Statnett (Morten Simonsen) to solve issues related to ECP and EDX. The software has of course no guarantees, but it is running in Statnett. The software can be downloaded from ediel.org: https://ediel.org/nordic-ecp-edx-group-nex/nex-statnett/

It requires at JRE 17 to run (same as for ECP 4.12/EDX 1.13). It runs like this:

>java -jar ekit.jar

which will return this – which explains briefly the 3 tools it contains:

```
ekit v1.0 is short for ECCoSP Kit, which is a toolkit build by Statnett (Morten Simonsen) for ECP/EDX v4.8+/v1.9+.
The kit offers the following

1) QueueProtector (QP) v1.0.0 - protect an ECP/EDX queue from flooding/filling; can also restore messages
2) ECPMsgDelay (EMD) v1.6.0 - analyzes ECP-logs to calculate delay for messages transmitted over the ECP-network
3) ECPPerfAnalyzer (EPA) v1.3.7 - analyzes ECP/EDX-logs to calculate throughput of the EPC/EDX-endpoint

To get more help on usage, run 'java -jar ekit.jar <QP|EMD|EPA>'
```

The tool usage is further explained in chapter 7.1.3, 7.2 (QP Tool) and 11.3 (EMD + EPA Tool).

10.3 How to connect to Database (Derby)

If you have followed the NEX Installation Guide you are running the default installation database which is a Java-based database named Derby. You can connect to this database following these steps:

1. Download Derby client: https://db.apache.org/derby/derby_downloads.html (download 10.15.2.0 if you're running ECP/EDX 4.10/1.11 or 10.16.1.1 if you're running ECP/EDX 4.12/1.13)
2. Unzip into a folder, ex /opt/derby
3. Change folder to your ECP og EDX, ex /var/lib/ecp-endpoint (your database you should now be found on "db"-folder in the folder you are located)
4. Stop ECP/EDX - you cannot edit the DB from more than one user at the time (or make a copy of the DB-folder and work on that)
5. Run '/opt/derby/db-derby-10.15.2.0-bin/bin/ij' or '/opt/derby/db-derby-10.16.1.1-bin/bin/ij'
    a. ij> connect 'jdbc:derby:db';

   b. The "db" marked in red above is the name of the folder, so this works if you want to copy the database to another folder and work on it while the ECP endpoint is running. While inside the ij-tool you can do all SQL and some other commands:
   c. ij> help;
   d. ij> show tables;
   e. ij>DO-SOMETHING-USEFUL
   f. ij> exit;
6. After exit and if you've edited the database (by INSERT/UPDATE/DELETE) you should check that file permission/ownership is the same as before – and if not, change back so that the ECP/EDX process can read/change the database.

11 Monitoring

There are many possibilities to monitor your endpoint. NEX Installation Guide mentions the Connectivity Check which is a useful and basic monitoring of your endpoint's connectivity. With v4.12 of ECP it's now also possible access multiple URLs which can be fed into monitoring software. These URLs are:

EP (ECP):

- /ECP_MODULE/actuator/prometheus **(NB! Can be very slow)**
- /ECP_MODULE/actuator/info
- / ECP_MODULE/actuator/health
- / ECP_MODULE/actuator/readiness

EDX:

- /actuator/prometheus
- /actuator/info
- /actuator/health
- /actuator/readiness

Of these, we will focus on the prometheus option, since the other contain much less information and are meant for container-environment. NB! If your ECP-database contains hundreds of thousands of entries, the prometheus-URL will be very slow, because it scans the largest table 8 times. In this situation it is not recommended to run the prometheus-URL or at the very least, do scraping rarely. We have measured up to 60s response time on this URL on large DB.

11.1 Prometheus

Prometheus-interface returns several hundred metrics. These metrics can be categorized into three groups:

- OS resources
- Application Server (Java/Tomcat/Spring) resources
- ECP/EDX resources

To further reduce the task at hand we focus mostly on the ECP/EDX metrics. The reason for this is that OS-resources is a generic problem that is most likely solved/monitored already. The Application Server is a relatively stable part of the stack and should not have much trouble unless in high traffic situation – and that will be detected in the ECP/EDX metrics.

Focusing on the ECP/EDX resources we suggest that it is almost exclusively the IBR (queue) metrics that matter. Put bluntly: If you have no problems with the queues, then ECP/EDX will work fine. As this document shows, this is an oversimplification, but it covers a lot of the issues in everyday operation. Many of the issues in this document is more about special/initial problems. If you follow the advice from NEX Installation Guide to have Connectivity Checks and then add Prometheus metrics to monitor the queues, you have come a long way to know the state of ECP/EDX. ECP/EDX Administration Guide has a chapter explaining these metrics.

11.1.1 The most useful metrics

If queue-size on any queue is above 0 for a long time it could be a problem

```
ecp_internal_broker_queue_queue_size
edx_internal_broker_queue_queue_size
```

With dequeue count you can know the rate of consumption from the queue, effectively knowing the processing rate.

```
ecp_internal_broker_queue_dequeue_count
edx_internal_broker_queue_dequeue_count
```

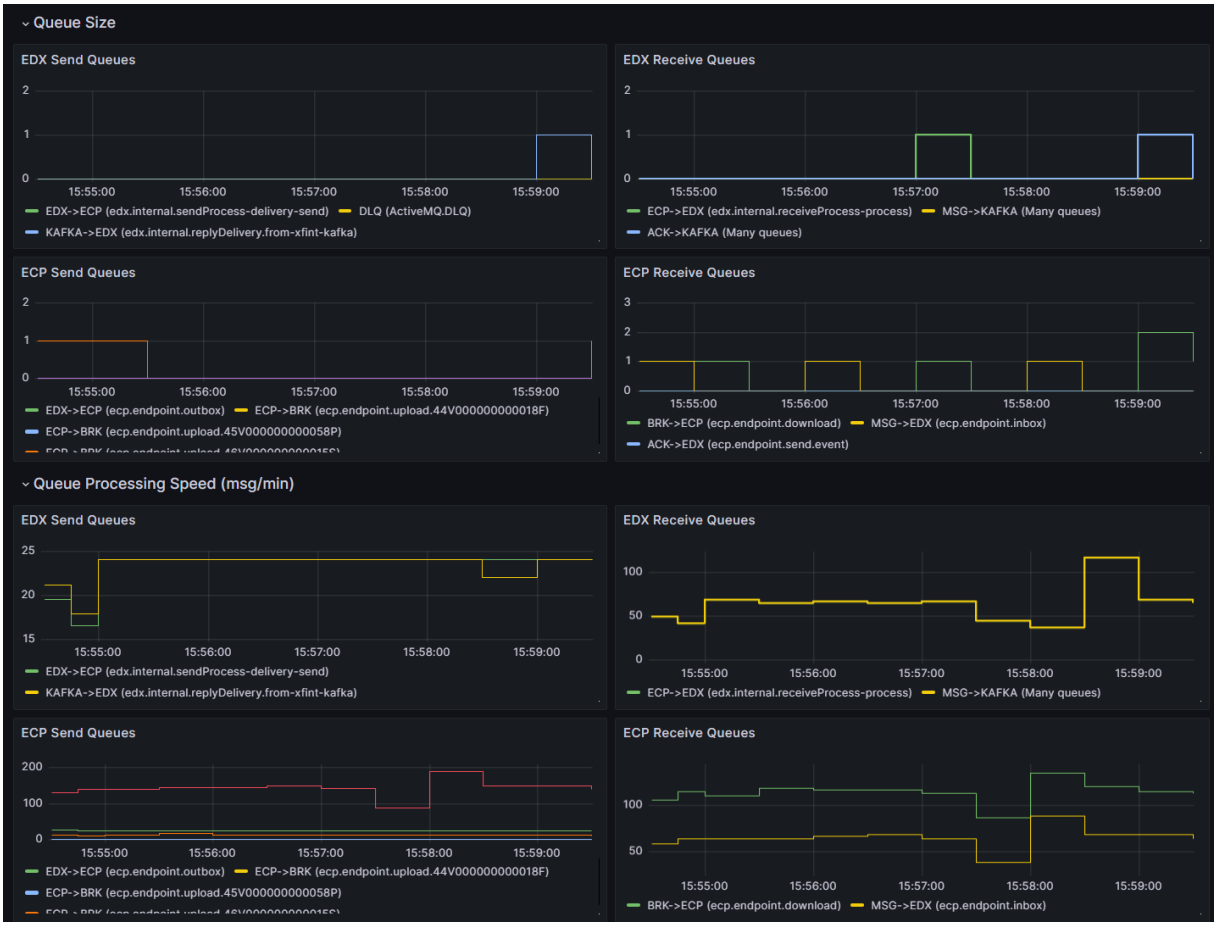
Capture if InternalBroker (IBR) is running full – if 100 the IBR will block, nothing will work.

```
ecp_internal_broker_store_percent_usage
edx_internal_broker_store_percent_usage
```

11.2 Grafana

11.2.1 Queue dashboard

Given the queue-size and dequeue-count metrics mentioned above, it possible to construct a dashboard like this in Grafana:



The 4 panels at the top show the queue-size-metric for various queues (EDX vs ECP, Send vs Receive). The 4 panels at the bottoms shows the dequeue-count-metric, where we measure the change in the count pr min. The same division between ECP/EDX and Send/Receive is applied.

To begin with, one doesn't have to do this so complicated, but can make one panel for queue-size and another for queue-dequeue-count. The only minor complication is to get the rate pr min correct. Use the following formula (ex metric **ecp_internal_broker_queue_dequeue_count**):

```
delta(ecp_internal_broker_queue_dequeue_count{}[1m])
```

You will soon see why it is useful to split it into several panels, it can become very confusing if you don't do it. The reason to split between Send and Receive is that it is good to know in which direction things are queuing up. That said, as soon as and EP receives a message, it will respond with 2 ACK (Delivery and Receive) back to the remote EP. And every time one sends a message, it will receive 2 ACKs. Therefore, it is not easy to know which EP is the **root-cause** of all the traffic. This is actually easier to see if you study the ECP-GUI Inbox/outbox, to get a sense of who is sending the initial business-message and who is responding with a business-message-ACK.

There are a number of queues in ECP/EDX, but these are the most essential:

11.2.2 Essential queues

Sending queues, starting from BA, through EDX, to ECP and to BR:

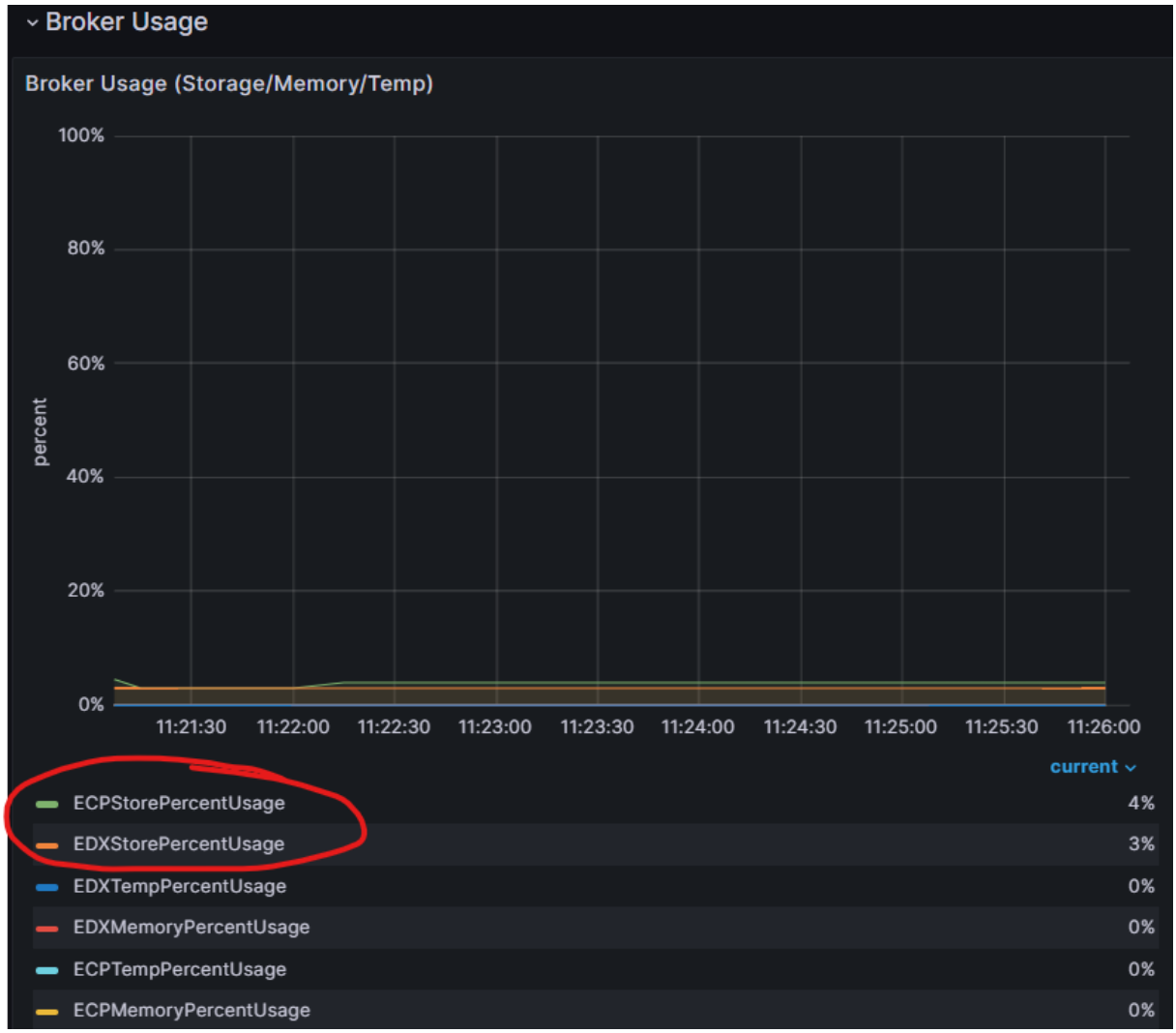
- edx.endpoint.outbox* (From BA)
- edx.internal.sendProcess-delivery-send (To ECP)
- ecp.endpoint.outbox (From EDX)
- ecp.endpoint.upload.* (To BR)

Receiving queues, starting from BR, through ECP to EDX and to BA:

- ecp.endpoint.download (From BR)
- ecp.endpoint.inbox (MSG to EDX)
- ecp.endpoint.send.event (ACK to EDX)
- edx.endpoint.inbox* (To BA)

Communication to/from BA could be other ways than through these queues, it is merely a hint.

11.2.3 Broker-memory panel



This panel is a simple way to know whether your IBR is about to overflow. If the percent reaches 100 then the traffic will be halted. Focus on the StorePercentUsage. This is a very simple panel to make once you have the metrics available in Grafana.

11.3 Logs

The logs contain a lot of information and some of it we can extract using Ekit (see chapter 10.2 for installation). There are two log tools:

- ECPPerfAnalyzer (analyze the flow of traffic through the ECP/EDX-endpoint)
- ECPMessageDelay (analyze the flow of traffic from this EP to remote EPs)

11.3.1 ECPPerfAnalyzer (EPA)

Run Ekit like this to see options on EPA:

```
>java -jar ekit.jar EPA
```

```
ECPPerfAnalyzer (EPA) v1.3.7
Usage  : java -jar ekit.jar EPA [-afl] <ecp-log-directory> <edx-log-directory>

Example: java -jar ekit.jar EPA /var/log/ecp-endpoint /var/log/edx-toolbox
Example: java -jar ekit.jar EPA -af /var/log/ecp-endpoint /var/log/edx-toolbox

The analyzer is compatible with ECP 4.8-4.12 and EDX 1.9-1.13 - it depends upon certain log events. Other
versions of ECP/EDX might not give any useful results. By default it only counts PLAIN/STANDARD msg.
Numbers in parenthesis denotes an incomplete count (not a full minute/hour or log-rotation).

Option: a     Also count the ACKs, not just PLAIN/STANDARD msg
Option: f     Tailing the logs
Option: l     Only process the newest logfile (the gz-files will be skipped)
```

Running EPA you will with the -f option will also provide CPU/Load information (see last part of screenshot – showing 2 minutes run):

```
    MINUTE: 20240319-11:41       4       4 |       4       2 |       0       0 |       0       0 |       4       4 |
    MINUTE: 20240319-11:42      15       6 |      15       8 |      24       5 |      24       4 |      39      39 |
    MINUTE: 20240319-11:43       0       0 |       0       0 |       0       0 |       0       0 |       0       0 |
    MINUTE: 20240319-11:44       0       0 |       0       0 |       4       2 |       4       2 |       4       4 |
    MINUTE: 20240319-11:45      22       8 |      22       6 |      28       4 |      28       4 |      50      50 |
    MINUTE: 20240319-11:46       4       2 |       4       2 |      14       4 |      14       4 |      18      18 |
    MINUTE: 20240319-11:47      15       8 |      15       7 |      10       4 |      10       3 |      25      25 |
    MINUTE: 20240319-11:48       2       1 |       2       2 |       2       1 |       2       1 |       4       4 |
    MINUTE: 20240319-11:49       0       0 |       0       0 |       4       1 |       4       1 |       4       4 |
    MINUTE: 20240319-11:50       4       4 |       4       4 |       5       1 |       5       1 |       9       9 |
    MINUTE: 20240319-11:51      11       5 |      11       6 |       9       3 |       9       4 |      20      20 |
    MINUTE: 20240319-11:52      23       9 |      23       8 |      16       4 |      16       3 |      39      39 |
    MINUTE: 20240319-11:53       4       3 |       4       3 |       2       1 |       2       2 |       6       6 |
    MINUTE: 20240319-11:54       2       2 |       2       2 |       7       2 |       7       2 |       9       9 |
    MINUTE: 20240319-11:55      11       4 |      11       4 |      12       2 |      12       3 |      23      23 |
    MINUTE: 20240319-11:56       4       4 |       4       4 |       1       1 |       1       1 |       5       5 |
    MINUTE: 20240319-11:57      15       8 |      15       6 |      24       5 |      24       5 |      39      39 |
    MINUTE: 20240319-11:58       0       0 |       0       0 |       0       0 |       0       0 |       0       0 |
    MINUTE: 20240319-11:59       2       1 |       2       2 |       4       1 |       4       1 |       6       6 |
      HOUR: 20240319-11        540      31 |     540      31 |     600      44 |     600      44 |    1140    1140 |
-------------------------------------------------------------------------------------------------------------------
                    Time      EDX->ECP      |    ECP->BROKER      |     ECP->EDX        |     EDX->BA         |  EDX     ECP     |      CPU
                              Count  Peak    |    Count  Peak      |     Count  Peak     |     Count  Peak     |  Count   Count   |  PERCENT   LOAD
-------------------------------------------------------------------------------------------------------------------
    MINUTE: 20240319-12:00      31       9 |      31       8 |      32       3 |      32       3 |      63      63 |
18  MINUTE: 20240319-12:01       4       2 |       4       2 |      14       4 |      14       4 |      18      18 |         0    0.41
 1  MINUTE: 20240319-12:02      15       7 |      15       8 |      10       4 |      10       3 |      25      25 |         4    0.30
```

Here you see:

- Traffic going from EDX to ECP and ECP to BR (1. and 2. major column is showing traffic SENT)
- Traffic going from ECP to EDX and EDX to BA (3. and 4. major column is showing traffic RECEIVED)
- Summary of all traffic in/out in ECP and EDX (5. major column)
- CPU/Load information (6. major column)
- One line pr minute summary
- One line pr hour summary
- Peak (minor columns) show the maximum traffic in a second in that particular minute or maximum traffic in a MINUTE in a particular HOUR.

This can be very useful to get an understanding of the flow of traffic in your endpoint. Also useful for identifying bursts in traffic!

11.3.2 ECPMessageDelay (EMD)

Run Ekit like this to see options for EMD:

```
>java -jar ekit.jar EPA
```

```
ECPMsgDelay (EMD) v1.6.0
Usage  : java -jar ekit.jar EMD [OPTION] <ecp-log-directory>

 OPTION:
     -s              : process only 'sent messages' - this is default behaviour, these timestamps are most trustworthy
     -r              : process only 'received messages', -s  will override -r
     -h              : all calculations is based on previous hour. Ideal for hourly cron-job
     -i              : all calculations is up-until previous hour. Ideal for first run (initialization)
     -d<sec>         : The delay-limit columns will show the number based on this limit in seconds. Default is 10
     -e<sec>         : The expire-limit columns will show the number based on this limit in seconds. Default is 60. Disabled
     -o <filename>   : The entries that exceed the delay limit will be appended to this file
     -c <filename>   : Received entries that has been received before sent, indicating clock skew (only with -r)
     -p <filename>   : Append summary pr opposite endpoint pr hour to this file
     -x <filename>   : Produces monthly summary of the file specified. The file must contain the std-out or the opposite fil
     -y<nodays>      : Only valid with -x. Specifies the number of days to include in the summary. Default is last 7 days.

Example 1: Summary of all sent messages, delay/expire-limit is 10/60 sec:
        java -jar ekit.jar EMD /var/log/ecp-endpoint
Example 2: Summary of last hour received messages, delay/expire-limit is 10/60 sec:
        java -jar ekit.jar EMD -r -h /var/log/ecp-endpoint
Example 3: Summary of last hour received messages + 3 more files for different purposes, delay/expire-limit is 20/60 sec:
        java -jar ekit.jar EMD -r -h -d20 -c neg.json -o lim.json -p opp.json /var/log/ecp-endpoint
Example 4: Summary of last hour sent messages + 2 files for different purposes, delay/expire-limit is 10/60 sec:
        java -jar ekit.jar EMD -h -o lim.json -p opp.json /var/log/ecp-endpoint
Example 5: Initial summary, run before you start running hourly cron (-h):
        java -jar ekit.jar EMD -i -o lim.json -p opp.json /var/log/ecp-endpoint
Example 6: Summary of summary, gives daily + monthly summary of the file specified (either stdout or opposite file):
        java -jar ekit.jar EMD -x stdout.json -y7

The delay analyzer is compatible with ECP 4.9-4.12 - it depends upon certain log events. Other
versions of ECP might not give any useful results. It calculates the delays of the message
transmissions in both directons, SEND and RECEIVE:
  * SEND delay is recorded when ACK is received and then compared with SEND tms. New in 1.6.0: If no ACK found for old msg,
  * RECEIVE delay is recorded when STD message is recevied and compared with 'Generated' tms in the msg

The main output is a series of json-objects, one pr hour with a summary of the delays/sizes.
The output is meant to be easy to read for humans and easy to parse for Splunk, etc. The OPTIONS allow you
to investigate the delays in more detail.

Since generated tms inside the message (RECEIVE case) can come from an endpoint with a clock skewed
compared to this endpoint, we might get negative RECIEVE dealys. Those are not counted, but can be
listed using the -c option. The -o option will list all messages that exceed the limits in seconds.
The -p option will give a summary of the traffic for each opposite endpoint pr hour. Most data in the output
is self-explanatory, but the 'troubleIndex' is not. The index is calculated like this:
  troubleIndex = overrepresentation * limitFreqOfEndpoint * limitCount * limitAvgIndex / 100
     overrepresentation is 100 if limits are not overrepresented nor underrepresented
     limitFreqOfEndpoint is 100 if all traffic to the endpoint exceed the limit
     limitCount is simply the number of messages that exceed the limit
     limitAvgIndex is the index (1-100), 1 when limitAvg == limit up to max 100 when limitAvg >= expireLimit (60s default)
The higher the index, the worse the problem is.
```

This is tool measures the delay in the ECP-network. You can specify what you think is classified as "delay" and "expired". It's mostly useful for TSOs or those EP that communicate with many different EPs. One can use output from this analysis to pinpoint if any remote EP has response time problems. The screenshot above says everything, but just to make clear:

- Send-traffic is measured from time sent from EP until first ACK is received back to EP
- Receive-traffic is measured by the "Generated"-timestamp in the logs, which represent when the remote EP actually made the message, and compare it with the local clock. There could be negative timestamps if clocks are not in sync.

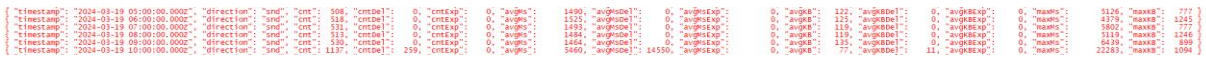
There are many possible output from this tool, but they all share a common set of abbreviations that are necessary to understand the output:

- snd (Send – into ECP-network)
- rcv (Received – from ECP-network)
- cnt (Count)
- ms (Millisecond)
- avg (Average)
- max (Maximum)
- KB (Kilobyte)
- del (Delayed – default: 10-60 sec)
- exp (Expired – default: 60+ sec)
- lim (Over limit – default 10+ sec)

The output is made in json-objects, to better be suited for some tool like Splunk. But it is also made to be human-readable.

11.3.2.1 Standard output

Sorry about the small screenshot, please zoom in:



The standard/console output shows a summary of messages sent (default or -s option) or received (-r option) pr hour.

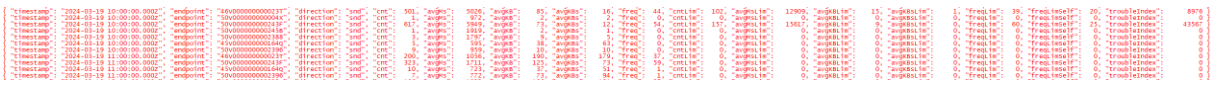
11.3.2.2 Delayed messages output (-o)

With the -o option + <filename> you can retrieve a list of all the messages that was over delay limit (default 10s+). The output is like this (again, please zoom in):

Use this list to identify particular messages, or maybe identify certain trends/patterns. You can see which broker was used, that can sometimes be useful.

11.3.2.3 Opposite summary (-p)

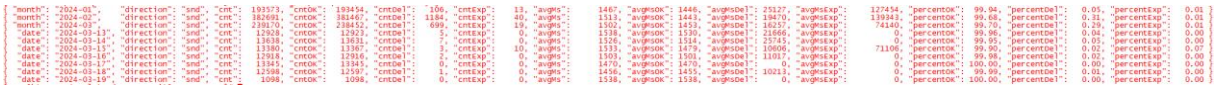
With the -p option + <filename> you can retrieve a summarized list of traffic pr "opposite endpoint" pr hour. With this you can pinpoint certain endpoints with "trouble":



The goal here is to look for those rows with the highest troubleIndex (rightmost column). Explanation of how that is calculated is shown above in the overall explanation of EMD (running ekit.jar EMD without further arguments).

11.3.2.4 Summary of summary (-x)

With the -x option + <filename> AND the <filename> points to a file with Standard-output shown in chapter 11.3.2.1 you can get a summary pr month/day:



The main goal here is to summarize everything into percentages shown at the right-most columns. percentOK is showing how much was sent + ack-received within the delay-limit (default 10 sec).

With -x option + <filename> AND the <filename> points to a file with output from (-p) show in chapter 11.3.2.3 you can get a summary pr opposite endpoint pr month/day (in this example -y is 0, so no days, only monthly summary):