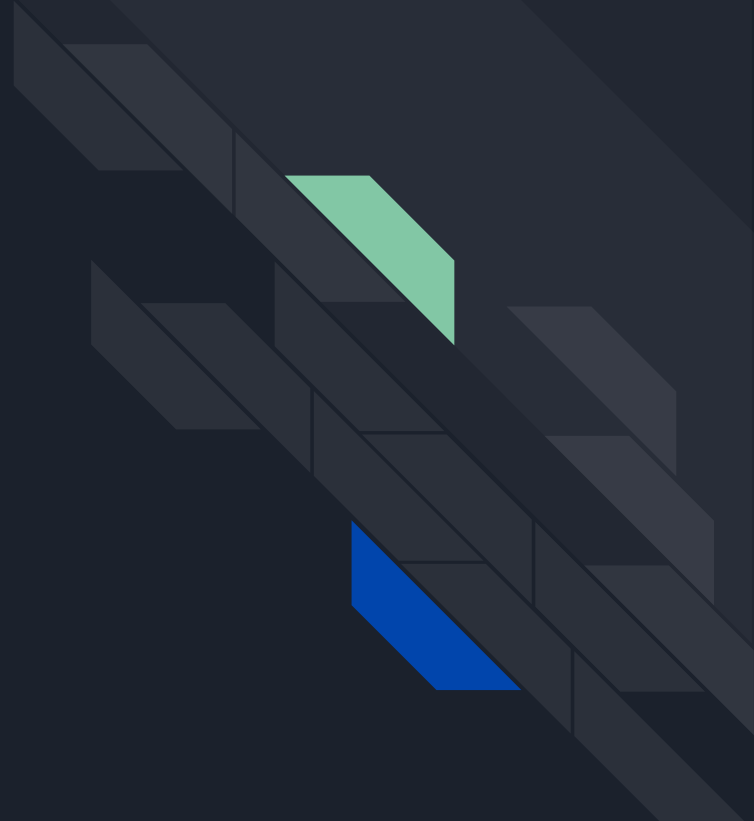

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one in front of the green one.

# Duniter Network Protocol: WebSocket to Peer

WS2P : Protocole réseau  
de Dunitier, basé sur  
l'utilisation de  
Websockets





# Duniter Network Protocol: WebSocket to Peer v1



# Profil d'un Endpoint

API	UUID	HOST	PORT	PATH
-----	------	------	------	------

API: WS2P ou WS2PTOR (ws2p avec tor)

UUID: Identifiant du noeud (+clé publique)

HOST + PORT: adresse (ip ou domaine) et port pour accéder au noeud

PATH: Chemin supplémentaire (optionnel)



# Connexion à d'autres nodes

WS2P ne prend pas en charge la découverte initiale d'autres nodes. Il faut fournir initialement des certificats en base de données du noeud, afin de renseigner d'autres noeuds.

Ensuite, le noeud va effectuer un certain quota de connexions toutes les 10 min.

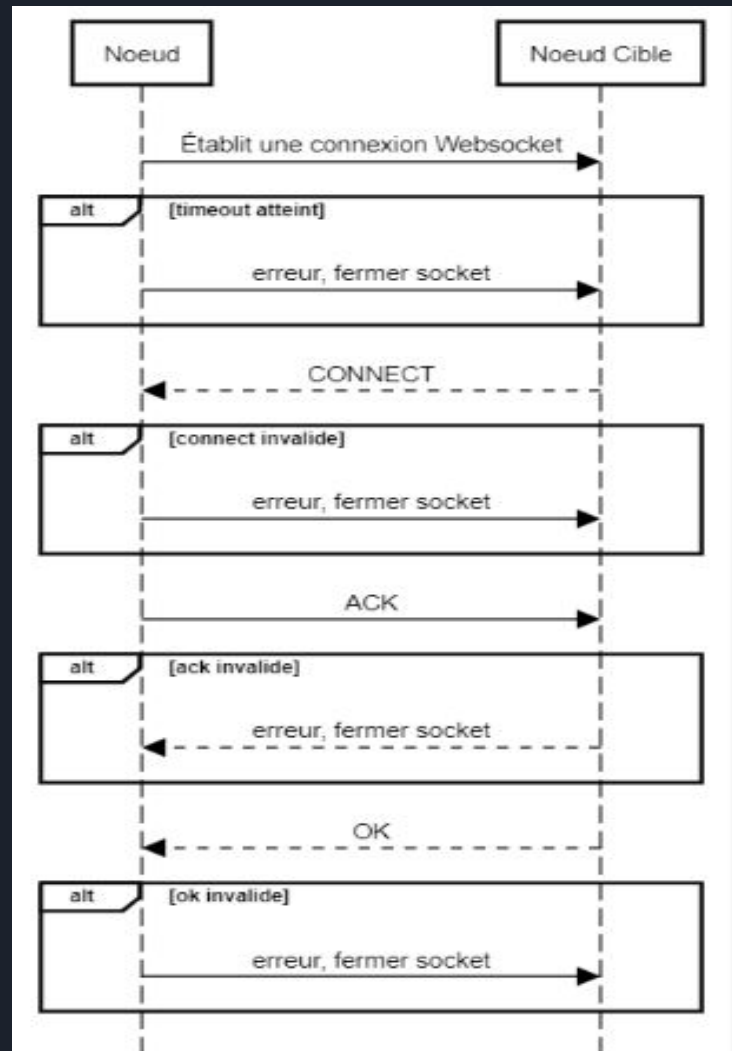
Si le noeud utilise Tor, il fera passer en priorité les connexions avec les autres noeuds qui utilisent Tor.

Sinon, il applique une pondération:

+1 si la clé publique est membre, +2 si la clé est privilégiée, et +4 si la clé est identique à celle du noeud. (ce point requerrait une clarification)

## Ouverture d'une connexion ws2p

Diagramme de séquence de la moitié d'une connexion ws2p. L'autre moitié est symétrique. >





# Format des messages

En JSON pour récupérer les données, en RAW pour signer.

**CONNECT**: CONNECT, clé publique, challenge et signature

le challenge est le texte ajouté à la string à signer dans le ACK et le OK qui suivront

la signature est une signature ED25519 vérifiable (préciser avec quoi ? -> avec la clé publique non ? à vérifier)

**ACK**: ACK, clé publique, et signature

la signature est réalisée grâce au challenge fourni dans le CONNECT

**OK**: OK, clé publique, signature

la signature est réalisée grâce au challenge fourni dans le CONNECT



# Réponse

Un noeud essaie toujours de répondre à un CONNECT, sauf si son timeout interne expire.

Elle refusera la connexion si l'uuid n'est pas le même, si la clé publique est bannie, si le noeud n'accepte que des connexion privilégiées et que ce n'en n'est pas une, si une connexion existe déjà avec cette clé, si le quota de connexions est atteint, et si il n'y a que des connexions moins prioritaires en attente.





# Messages HEAD

Les messages head sont un moyen pour les noeuds de tenir le réseau au courant de leur état. En effet chaque noeud envoie son état et les états connus à d'autres noeuds. Un nouvel état reçu d'un noeud écrase les états précédents de celui ci connus par les autres. Ainsi, tout le réseau est toujours en connaissance des derniers états des noeuds.

Ils existent en de nombreuses versions rétrocompatibles. On détaillera la dernière version.



# Structure de HEADv2

JSON: Message, Signature, Message v2, Signature v2, Step

Les signatures sont les signatures respectives des messages raw des versions correspondantes. Le champ step contient la distance entre le noeud et le noeud décrit par HEAD. Ce champ n'est pas sécurisé et est informatif.

Le message version 1 contient:

API:MESSAGE\_TYPE:1:PUBKEY:BLOCKSTAMP:WS2PID:SOFTWARE:SOFT\_VERSION:POW\_PREFIX

Le message version 2 contient:

API:MESSAGE\_TYPE:2:PUBKEY:BLOCKSTAMP:WS2PID:SOFTWARE:SOFT\_VERSION:POW\_PREFIX:FREE\_MEMBER\_ROOM:FREE\_MIRROR\_ROOM



# Détail des champs du message HEAD

API = sécurité du réseau en deux temps

Privée: OCA -> ne pas joindre les endpoints TOR, OTM -> joindre les endpoints TOR via TOR,

OTA -> Tout joindre par TOR, OTS -> Ne joindre que les endpoints TOR, et cela via TOR.

Publique: IC -> endpoint normal, IT: endpoint TOR

MESSAGE\_TYPE = HEAD (message head)

PUBKEY = clé publique

BLOCKSTAMP = Numéro du bloc et hash

WS2PID = ID de la connection ws2p

SOFTWARE = duniters-ts

SOFT\_VERSION = version

POW\_PREFIX = préfixe du nonce utilisé dans la preuve de travail

FREE\_MEMBER\_ROOM = Nombre de connexions restantes

FREE\_MIRROR\_ROOM = Nombre de connexions restantes sans compter celles en attente



# Documents

Les documents décrits dans le protocole de dunitier sont envoyés via des messages DOCUMENT.

Le type est spécifié via un numéro, parmi ceux dans cette table:

DOCUMENT_TYPE_ID	DOCUMENT_TYPE_NAME
0	peer
1	transaction
2	membership
3	certification
4	identity
5	block

Les documents peer ne sont pas décrits dans le protocole et suivent leur propre structure, listant l'ensemble des endpoints disponibles, le statut, la version, le dernier bloc, le tout signé.



# Gestion des HEAD

Chaque Noeud conserve en RAM un dictionnaire associant les noeuds (clés publiques et ID) et leur dernière HEAD. Ce dictionnaire peut être consulté par un utilisateur via HTTP et BMA.

La propre HEAD du noeud est changée si un nouveau bloc vient se greffer à la chaîne, ou si une résolution de conflits est faite.

Une HEAD est mise à jour dans le dictionnaire si une head reçue est valide, non modifiée, si elle a un bloc aussi récent ou plus récent que l'actuelle correspondante, ou si elle est identique avec une step plus petite. Si elle n'est pas membre (clarifier: "membre" ?) il faut vérifier qu'une connexion est établie avec elle.

Une fois la mise à jour effectuée, la nouvelle information est retransmise avec un step augmenté.

De même, si un document est correct, il est retransmis à toutes les connexions.



# Requêtes


Certaines requêtes peuvent être effectuées via Ws2p pour obtenir des informations supplémentaires. Ce sont des requêtes asynchrones dont la réponse est rendue avec un identifiant unique lié à la requête.

getCurrent: obtenir le dernier bloc

getBlock: obtenir un bloc donné

getBlocks: obtenir plusieurs blocs

getRequirementsPending: obtenir le statut de ce qui est requis pour les identités ayant un certain nombre de certifications.



# Duniter Network Protocol: WebSocket to Peer v2

(clarification: est il déjà en usage, il ne semble pas complet ?)



# Nouvelle structure et sécurité

Ws2pv2 utilise une nouvelle couche de sécurité, via le protocole PKSTL v1 qui encapsule les messages ws2pv2 dans sa charge binaire.

Les messages ws2pv2 sont binarisés via le procédé CBOR (ou bincode parfois)

Tous les messages ws2pv2 partagent une structure json commune:

- indication de la version de ws2p, du nom de la monnaie, de l'id du noeud.
- La clé publique (avec le détail de l'algorithme de cryptographie, et la valeur de la clé dans un vecteur de bytes).
- La charge, avec le type de message et son contenu.





# Nouveau format d'Endpoint

API	VERSION	NF [...]	AF	HOST	IPv4	IPv6	PATH
-----	---------	----------	----	------	------	------	------

API: WS2P, VERSION: 2

NF et AF:

Network features (Liste de networks supplémentaires): +1 si HTTP, +10 si Websocket, +100 si TLS/SSL, +1000 si TOR

API features: +10 si support des connexions lentes, +100 si support du format Bincod rust

HOST: domaine, IPv4-6: ips, PATH: chemin d'accès optionnel

La peer card contient en conséquence ce nouveau format d'endpoint, en plus d'une signature apposée sur la clé publique et le blockstamp des certifieurs



# Changements dans le processus d'établissement de connexions

- Une synchronisation des peer cards est réalisable afin de partager celles ci à travers le réseau
- Le CONNECT du noeud cible n'est envoyé qu'à la réception d'un premier connect, en même temps que le premier ACK.
- Les messages OK peuvent être remplacés par des messages SECRET\_FLAGS
- Lors d'une connexion visant à synchroniser le réseau, des stamps sont envoyées, ainsi que des sync\_info (point à détailler ?)



# CONNECT v2

Le message CONNECT contient un challenge, comme en v1, un type de connexion:

- incomming : réponse à une connexion
- client: connect d'un client
- server: connect entre deux noeuds
- sync: synchronisation
- syncaskchunk: synchronisation pour un chunk (précision: c'est à dire ?)

ces deux dernier cas requièrent la précision d'un blockstamp

Il contient également des API features, en cas de mise à jour du type de connexion, et une peer card (précision: pourquoi ?)



## Autres messages de connexion: version 2

Le message ACK ne contient plus que la signature du challenge

Le message SECRET\_FLAGS peut contenir un flag signalant une mauvaise connexion réseau, et remplace OK dans ces situations.

Le message OK contient seulement le préfixe du noeud, si celui ci accepte de le révéler, et témoigne de l'établissement de la connexion.

le message SYNC\_INFO contient l'ensemble des blocs et des peer cards que les noeuds se partagent pour synchroniser le réseau de noeuds.

Le message KO contient un numéro d'erreur (0=timeout, 1 = full, 2=low, 3=autre branche)



# Requêtes v2

Les requêtes ont un ID de 4 bytes, un type et au plus deux paramètres.

Les types sont spécifiés ici:

0	EMPTY_REQUEST	-	-
1	CURRENT_BLOCKSTAMP	-	-
2	BLOCKS_HASHS	begin_block_id (u32)	blocks_count (u16)
3	CHUNK	begin_block_id (u32)	blocks_count (u16)
4	CHUNK_BY_HASH	chunkstamp (Blockstamp)	-
5	WOT_POOL	folders_count (u16)	min_cert (u8)

Elles permettent d'obtenir le dernier hash, les hash de blocs, de chunks...

(précisions pour wot\_pool...)

La réponse est renvoyée, avec un code de statut , le détail est dans les tableaux en fin de documentation.



# HEAD v3

Le format des head a été modifié:

api\_outgoing\_conf: indique les types de connexions permises

api\_incoming\_conf: indique les types de connexions permises

free\_mirror\_rooms: places restantes pour les connexion à ce noeud

low\_priority\_rooms: places occupées par des connexions moins prioritaires

node\_id: id

pubke: clé publique

blockstamp: hash du bloc courant

software\_name/soft\_version

String signature : signature

step : nombre de pas entre les deux noeuds