The Metanet

A Blockchain-based Internet

Technical summary

Copyright

Contents

1 Introduction

We propose a new protocol for a distributed peer internet - the Metanet - that uses an underlying blockchain to store data in a directed graph structure that can be easily searched and accessed by a browser. The protocol is designed not only to replicate much of the functionality of the internet but also to leverage many additional advantages associated with blockchain technology. It is a layer-2 solution that does not require any changes to the protocol or consensus rules of the underlying blockchain.

The Metanet protocol allows for the creation of a directed graph for data and content that is not necessarily related to payments themselves. In this specification, any internet-like or other content data is to be considered a part of the Metanet, as these data and their related uses are beyond the simple payment use case of the Blockchain.

In summary, we propose the structuring of transactions to be interpreted as Metanet nodes, and using digital signatures to create edges as links between them. This allows a graph to be reconstructed off-chain, by anybody, using on-chain data and a standardised protocol.

This graph then allows on-chain data to be usefully related, handled and distributed by network peers in an internet-like model where trust is decentralised, transparency increased, and data integrity, authenticity and validity secured by the Blockchain and proof-of-work mining.

2 The Metanet protocol

The Metanet is a layer-2 protocol, designed such that any data can be stored immutably on a blockchain in a searchable and flexible manner. This is achieved primarily by structuring all Metanet data in a directed graph, governed by public key addresses.

The data involved in the Metanet protocol can be broadly categorised as:

I. The Metanet flag – a 4-byte umbrella prefix to be used in every Metanet transaction. This flag signifies that a transaction is to be considered a part of the Metanet graph. All other related sub-protocols can be encompassed by the Metanet flag.

II. Attributes – any relevant and useful metadata relating to content data. This can include indexing, permissioning and encoding information about given content data.

III. Content – the payload data of interest. This may consist of files, web-pages, links and any such content data may be raw, compressed or encrypted as needed.

The specific methods for inserting and structuring content and attribute data can be flexible and adapted to different sub-protocols and use cases. However, the common thread that defines the Metanet is that all insertion methods are made consistent with the node and edge structure specified in the over-arching Metanet protocol.

2.1 Node and edge structure

We will now detail the protocol for structuring Metanet transactions that allows for addressing, permissions, and version control. The structure of this distributed peer internet is analogous to the existing internet.

The aim of this directed graph structure is to

(i) Associate related content in different transactions;
(ii) Allow users to find content using human-readable keyword searches; and
(iii) Build server-like structures within a blockchain.

Our approach is to structure data associated with the Metanet as a directed graph. The nodes and edges of the Metanet graph correspond to

Node - A transaction associated with the Metanet protocol.

A node stores content. A node is created by including an OP_RETURN that is immediately followed by <Metanet Flag>.

Each node is assigned a public key P_{node}, also in OP_RETURN. The combination of this public key and transaction ID uniquely specify the index $ID_{node} := H(P_{node}||TxID_{node})$ of a Metanet node.

Edge - An association of a child node with a parent node.

An edge is created only when a signature $Sig\ P_{parent}$ appears in the input of a Metanet transaction, and therefore only a parent can give permission to create an edge. All nodes may have at most one parent, and a parent node may have an arbitrary number of children[1].

It should be emphasised that creating an edge between two nodes is the creation of a logical link between them, and does not necessarily mean one spending an output of the other. This allows the creation of edges to remain as flexible as possible over time.

A valid Metanet node (with parent) is given by a transaction of the following form:

$TxID_{node}$	
Inputs	Outputs
< $Sig\ P_{parent}$> <P_{parent}>	OP_RETURN <Metanet Flag> <P_{node}> <$TxID_{parent}$>

Figure 1. A Metanet node transaction.

[1] In the language of graph theory, the indegree of each node is at most 1, and the outdegree of each node is arbitrary.

This transaction contains all the information needed to specify the index of the node and its parent

$$ID_{node} = H(P_{node}||TxID_{node}) \ , \qquad ID_{parent} = H\left(P_{parent}||TxID_{parent}\right) .$$

Moreover, since the signature of the parent node is required, only a parent can create an edge to a child. This is a fundamental design element of the Metanet protocol, which intentionally facilitates more complex permissioning and addressing of content.

If the <$TxID_{parent}$> field is not present in a node, or it does not point to a valid Metanet transaction, then the node is considered an orphan.

Additional attributes may be added to each node. These may include flags, names and keywords. We will discuss this in a following subsection.

We have seen how the index of a node can be broken down into

a) Public Key P_{node}, which we interpret as the address of the node; and
b) Transaction ID $TxID_{node}$, which we interpret as the version of the node.

There are two interesting features that emerge.

1. **Version control -** If there are two nodes with the same public key, then we interpret the node with transaction ID with greatest proof of work as the latest version of that node.

   If the nodes are in different blocks, then this can be checked with the block height. For transactions in the same block, this is determined by the Topological Transaction Ordering Rule (TTOR).

2. **Permissioning -** A child of a node can only be created if the owner of the public key P_{node} signs the transaction input in the creation of a child node. Therefore P_{node} not only represents the address of a node but also the permission to create a child node. This is analogous to a standard bitcoin transaction – a public key is not only an address but also the permission associated with that address.

   Note that since the signature of the parent node appears in a UXTO unlocking script it is validated through the standard miner validation process at the point when the transaction is accepted to the network. This means that the permission to create a child node is validated by the bitcoin network itself.

It is worth noting that standard Internet Protocol (IP) addresses are unique only within a network at a certain point in time. On the other hand, the index of a node in the Metanet is unique for all time and there is no notion of separate networks, which allows data to be permanently anchored to a single object ID_{node}.

The node and edge structure allow us to visualise the Metanet as a graph (overleaf). This graph can be reconstructed off-chain by anybody with access to the relevant on-chain data.

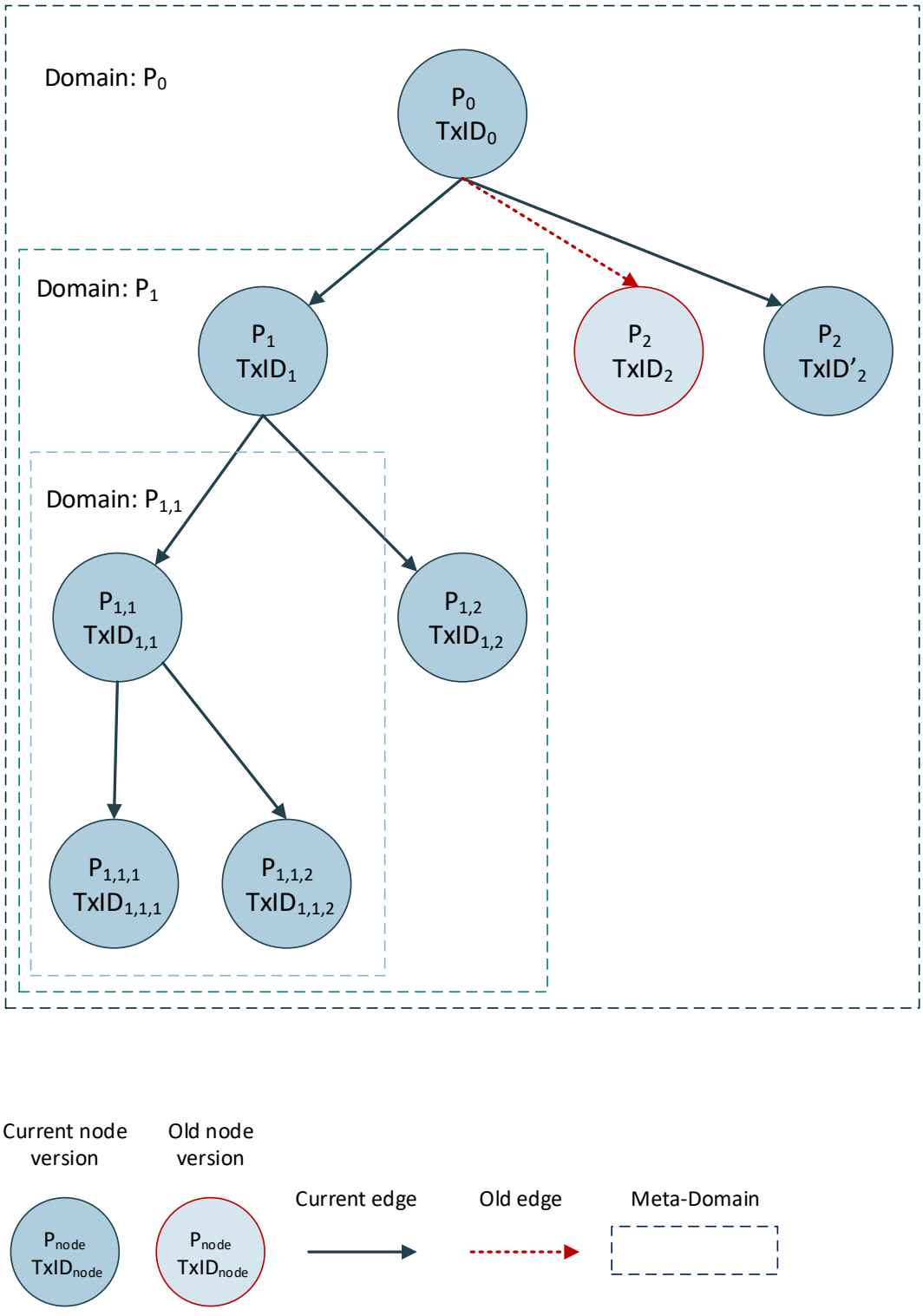

Figure 2. The Metanet graph structure.

2.2 Domains, naming and locating

The hierarchy of the Metanet graph allows a domain-like structure to emerge. We interpret an orphan node as a top-level domain , a child of an orphan node as a sub-domain, a grandchild as a sub-sub-domain etc., and a childless node as an end-point. See figure 2 above. Each top-level domain in the Metanet may be thought of as a tree with the root being the orphan node and the leaves being the childless nodes. The Metanet itself is a global collection of trees which form a graph.

The Metanet protocol does not stipulate that any node contains content data, but leaf (childless) nodes represent the end of a directed path on the data tree, and thus will be used generally to store content data. However, content may be stored at any node in the tree. Protocol-specific flags, included in nodes as attributes, may be used to specify the role of nodes in a data tree (disk space, folders, files or permissioning changes).

Recall that the internet uses the Domain Name System (DNS) to associate human-readable names to Internet Protocol (IP) addresses. The DNS is in a sense decentralised, although in practice it is controlled by a small number of key players, such as governments and large corporations. Depending on your DNS provider the same name may take you to different addresses. This issue is inherent when mapping short human-readable names to computer generated numbers.

Let us assume that an equivalent distributed system exists that maps a human-readable top-level domain name to the decentralised index ID_{root} of a root node. In other words, there exists a 1-1 function κ that maps human-readable names to Metanet root node indexes, for example

$$\kappa('bobsblog') = ID_{bobsblog} \left(= H\left(P_{bobsblog} || TxID_{bobsblog}\right)\right) .$$

The input to the left-hand-side is human-readable word, whereas the output on the right-hand-side is a hash digest, which will typically be a 256-bit data structure. Note that $P_{bobsblog}$ and $TxID_{bobsblog}$ are also not human readable in general. In the standard IP protocol this would be a map from $www.bobsblog.com$ to the IP address of the corresponding domain within the network.

The map κ should be interpreted as a measure to ensure backwards-compatibility of the Metanet with the internet in replicating the human-readability of DNS-issued domain names, but the naming and addressing scheme that provides the directed graph structure of the Metanet is not explicitly dependent on this map.

Possible existing forms of the mapping function κ include the DNSLink[2] system employed by IPFS or the OpenNIC service [2] but in general these sacrifice some element of decentralisation [3] in order to provide a map that is 1-1.

The purpose of this naming system is for a user to be able to identify a top-level domain in the Metanet by a memorable word (for example a company name) rather than a hash digest. This will also make searching for the domain faster as it is quicker to search for a keyword rather than a hash digest.

[2] This mapping can be stored in an existing TXT record as part of the DNS. This is similar to a DNSLink in the Interplanetary File System (IPFS) [1].

Vanity addresses

The public key used as the address of a Metanet node is not generally a human-readable object. However, it is possible to create human-recognisable public key addresses – vanity addresses P_{vanity} – which include a plaintext prefix that can be interpreted directly by a user.

An example of a vanity address with a desirable prefix is

$P_{bobsblog}$: bobsblogHtKNngkdXEeobR76b53LETtpyT

Prefix: bobsblog

Suffix: HtKNngkdXEeobR76b53LETtpyT

The combination of a chosen address P_{vanity} with a $TxID$ that together form ID_{node} is also beneficial as it means there is no central issuer of domain names ($TxID$s are generated by decentralised proof-of-work) and the names are recoverable from the blockchain itself. There are no longer the points of failure that exist within the internet DNS.

Since Metanet domains already come with a permissions system (the public key) there is no need to issue a certificate to prove ownership. The use of a blockchain for this purpose has already been explored in namecoin [4] for example. In our case we do not need a separate blockchain for this as we are doing everything within one blockchain.

Locating resources

Given that we have a map from a domain name to a node index we can build up a resource locator similar to that of a Uniform Resource Locator (URL) for the internet. We will call this a Metanet URL (MURL), and takes the form

$$\text{MURL} = \text{'mnp:'} + '//\text{domain name}' + '/\text{path}' + '/\text{file}'.$$

Each of the components of a URL – protocol, domain name, path and file – have been mapped to the structure of a MURL, making the object more user-intuitive and integrable with the existing structure of the internet.

This assumes that each node has a 'name' associated with its public key (address) that is unique at the level within the domain tree. This name is always the right-most component of the MURL for a given node, analogous to the name of a particular file in a directory. If two nodes at the same level in the tree have the same name, then they will have the same public key and so the latest version (according to $TxID$) is taken.

It is envisaged that many protocols may be developed for locating on-chain resources, such as the 'b://' and 'c://' protocols already in use. In the same way that the <Metanet Flag> is an umbrella protocol flag for all on-chain Metanet data, it is envisaged that a generalised resource location protocol 'mnp://' for the Metanet can be used as an umbrella locator by linking domain names to node identifiers ID_{node}.

This means, if we are able to associate an entity with an ID_{node}, then we can construct MURLs to locate resources simply by following a path down a Metanet tree of nodes associated with the root ID_{node}. This concept is generalised to support more sophisticated content-addressing or $TxID$-addressing as resource locators that can resolve such queries based on a node ID.

2.3 Searching the Metanet

We have defined our Metanet graph structure such that each node has a unique index ID_{node} and may also have a name attributed to it. This allows for content to be located using a MURL. In order to also enable quick search functionality, we allow for additional keywords to be attributed to a node.

The fixed attributes of a node are the index (ID_{node}) and index of parent node (ID_{parent}), each comprising two fields ($P, TxID$), and the optional attributes can include the name and keywords.

Node attributes:
{
index: $H(P_{node}||TxID_{node})$;
index of parent: $H(P_{parent}||TxID_{parent})$; (NULL if orphan)
name: 'bobsblog';
kwd1: 'travel';
kwd2: 'barbados';
⋮
}

A practical method to search the Metanet is to first use a block explorer to trawl through the blockchain and identify all transactions with the Metanet flag, check that they are valid Metanet nodes, and if so record their indexes and keyword in a database. This database can then be used to efficiently search for nodes with desired keywords. Once the index of the node(s) with the desired keywords is found its content can be retrieved from the block explorer and viewed.

By way of example, consider the P_1 branch of figure 3, where the nodes corresponding to public keys P_0, P_1 and $P_{1,1}$ represent a home page, topic page and sub-topic page respectively. These nodes are given the names 'bobsblog', 'summer' and 'caribbean', and their attributes are shown below.

Home page node P_0
MURL: mnp://bobsblog
{
index: $H(P_0||TxID_0)$;
index of parent: $NULL$
name: 'bobsblog';
kwd1: 'travel';
kwd2: 'barbados';
⋮
}

Topic page node P_1
MURL: mnp://bobsblog/summer
{
index: $H(P_1||TxID_1)$;
index of parent: $H(P_0||TxID_0)$;
name: 'summer';
kwd1: 'travel';
kwd2: 'barbados';
⋮
}

Sub-topic page node $P_{1,1}$
MURL: mnp://bobsblog/summer/caribbean
{
index: $H(P_{1,1}||TxID_{1,1})$;
index of parent: $H(P_1||TxID_1)$;
name: 'caribbean';
kwd1: 'travel';
kwd2: 'barbados';
⋮
}

In this example the leaf nodes $P_{1,1,1}$, $P_{1,1,2}$ and $P_{1,1,3}$ are given the names 'beaches', 'nightlife' and 'food' respectively and are used to store separate blog posts. The full domain structure is shown on the diagram overleaf (figure 3), including the MURL search path pertaining to each node in the tree.

We note that the Metanet can also incorporate a content addressable network (CAN) by storing a hash of the content stored by a node transaction as an additional attribute. This means Metanet nodes may also be indexed and searched for by content hash.

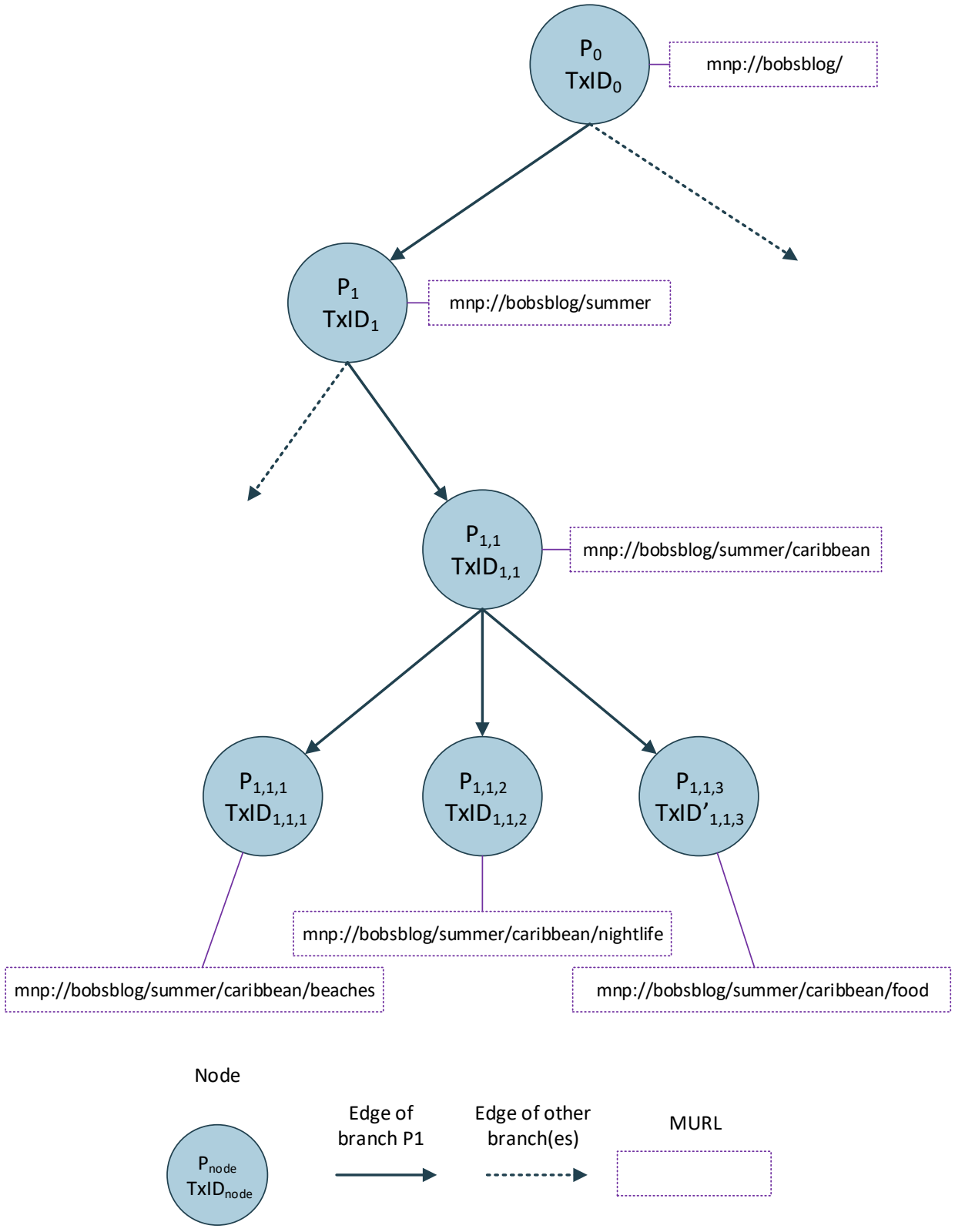

Figure 3. A Metanet graph tree for the domain 'bobsblog' including MURL search paths.

2.4 Data insertion

In section 2.1 we alluded to the fact that the Metanet protocol, which gives rise to a directed graph structure for on-chain content, can support many different methods for inserting data. For instance, in many cases it will be feasible to store content in a single transaction, but in others it may be sensible to split data across multiple transactions that are linked using the Metanet protocol.

Additionally, there may be users who wish to upload all their data in OP_RETURN payloads, whereas others may wish to upload content data in spendable outputs as: <Content> OP_DROP.

As mentioned previously, all these methods are compatible with and supported by the Metanet protocol, given that the basic form of a Metanet node transaction is employed in all cases.

Recall that the transaction shown in figure 1 specifies the minimum information required in a transaction for it to be considered part of the Metanet and to be interpretable according to the directed graph structure. Provided the elements shown in that diagram are present, any method for embedding the content data and its attributes may be used.

We provide here (non-exhaustively) some examples of how data might be embedded in ways that are consistent with the Metanet protocol. In what follows, the essential features of the Metanet protocol are highlighted in blue.

2.4.1 Using OP_RETURN alone

The diagram below shows a Metanet node containing all content data within a single OP_RETURN output.

$TxID_{node}$	
Inputs	Outputs
< $Sig\ P_{parent}$ > <P_{parent}>	OP_0 OP_RETURN <$Metanet\ Flag$> <P_{node}> <$TxID_{parent}$> <file_name> <file_type> <file_data>

Figure 4. A Metanet node using a single OP_RETURN.

The file name and file type are included as optional attributes in the OP_RETURN output, followed by the actual payload of file data itself.

Any set of attributes and schemas may be included depending on the file type or use case to encode the relevant metadata, and the file data may be split into several discrete chunks within the output itself.

2.4.2 Using OP_RETURN and OP_DROP

The diagram below shows a Metanet node in which the file data has been split across two spendable outputs in OP_DROP statements. This insertion method may be used if a large file is to be embedded, or simply if the user wishes to create fewer provably unspendable outputs.

<table>
<tr><td colspan="2">$TxID_{node}$</td></tr>
<tr><td>Inputs</td><td>Outputs</td></tr>
<tr><td>< $Sig\ P_{parent}$> <P_{parent}></td><td>OP_RETURN <$Metanet\ Flag$> <P_{node}> <$TxID_{parent}$>
<sub-protocol identifier>
<file_name> <file_type>
<recombination_scheme></td></tr>
<tr><td></td><td><file_data 1> OP_DROP</td></tr>
<tr><td></td><td><file_data 2> OP_DROP</td></tr>
</table>

Figure 5. A Metanet node using an OP_RETURN in conjunction with OP_PUSHDATA.

In this scenario, we have assumed it is sensible to include all the metadata and attributes within the OP_RETURN output, which then acts as a 'header' for the file.

However, it is just as plausible that additional attributes could be included in the spendable outputs if, for example, there were attributes that applied to <file_data 1> but not <file_data 2>.

Note that we have even included an additional identifier <sub-protocol identifier> to signify the use of a specific hypothetical data-insertion protocol. We see that the sub-protocol fits within the Metanet protocol itself, and can be considered an overlay protocol on top of the umbrella protocol that is the Metanet.

The only condition on the form of this node is still that it adheres to the basic Metanet node transaction format. This means that the Metanet flag, the node public key P_{node} and the parent transaction identifier $TxID_{parent}$ appear first in the OP_RETURN output, and that the public key and signature $Sig\ P_{parent}$ of the parent node appear in the input.

2.4.3 Using multiple transactions

It is possible to use any of the previous insertion techniques whilst also splitting data across multiple transactions, such that the two transactions are related according to the Metanet protocol.

The diagram below shows a pair of child transactions, Tx_1 and Tx_2, which both have the same parent according to the Metanet, and are therefore sibling nodes.

$TxID_{node\ (1)}$	
Input	Output
< $Sig\ P_{parent}$ > < P_{parent} >	OP_RETURN < $Metanet\ Flag$ > < $P_{node\ (1)}$ > < $TxID_{parent}$ > <sub-protocol identifier> <file_name> <file_type> <data_index 1> <recombination_scheme>
	<file_data 1> OP_DROP

$TxID_{node\ (2)}$	
Input	Output
< $Sig\ P_{parent}$ > < P_{parent} >	OP_RETURN < $Metanet\ Flag$ > < $P_{node\ (2)}$ > < $TxID_{parent}$ > <sub-protocol identifier> <file_name> <file_type> <data_index 2> <recombination_scheme>
	<file_data 2> OP_DROP

Figure 6. A pair of Metanet sibling nodes used to insert a large file split across two transactions.

These two transactions have a common parent, whose signature therefore appears in both their respective inputs. It is important to remember that this signature does not need to spend an output of the parent node itself, it simply has to be a valid signature from the parent public key.

In the case where the input of a child does in fact spend an output of its parent, the Metanet edge created is exactly the same as if the child were to spend an arbitrary output using the parent public key P_{parent} to sign.

2.5 The Metanet and the Internet

The Metanet is a protocol for structuring on-chain content data in a way that facilitates the use of the Blockchain as the base-layer for a distributed peer internet.

The following table gives an analogy between the Metanet protocol and the Internet Protocol:

Internet	Metanet
Website/file	**Node**
Owner	**Public Key** P_{node}
IP Address (non-unique)	**Node index (unique)** $ID_{node} = H(P_{node}\|\|TxID_{node})$
Domain structure	**Node tree structure**
Domain Name System	**Map from root node name to node index**
URL http://www.bobsblog.com/path/file	**MURL** mnp://bobsblog/path/file

Table 1. Summary of the analogies between the Internet and Metanet Protocols.

The principle here is to ensure the Metanet can provide at least the same functionalities as the existing internet, in addition to the inherent benefits and new use cases that come from the blockchain that underlies the Metanet.

3 References

Number	Reference
[1]	"IPFS Documentation", *Docs.ipfs.io*, 2019. [Online]. Available: https://docs.ipfs.io/guides/concepts/dnslink/. [Accessed: 22- May- 2019].
[2]	"OpenNIC Project", *Opennic.org*, 2019. [Online]. Available: https://www.opennic.org/. [Accessed: 22- May- 2019].
[3]	"Ten terrible attempts to make the Inter Planetary File System human-friendly", *Hacker Noon*, 2019. [Online]. Available: https://hackernoon.com/ten-terrible-attempts-to-make-the-inter-planetary-file-system-human-friendly-e4e95df0c6fa. [Accessed: 22- May- 2019].
[4]	"Namecoin", *Namecoin.org*, 2019. [Online]. Available: https://namecoin.org/. [Accessed: 22- May- 2019].