



# BLOCKCHAIN: THE HIDDEN GEM OF DATA NETWORKING

**BLAN**

**MOUSTAFA AMIN**

# Blockchain: The Hidden Gem of Data Networking

## Volume1 BLAN

---

### Contents

[Introduction](#)

[1. Blockchain](#)

[1.1. Centralization & Decentralization](#)

[1.2. Blockchain main Functions](#)

[1.2.1. Asset Creation & Transactions](#)

[1.2.2. Data Storage & Retrieval](#)

[1.3. Mining over blockchain](#)

[2. VXLAN](#)

[2.1. VXLAN Format, Traffic Flow & ECMP](#)

[2.2. VXLAN Modes of Operation](#)

[2.3. Control-Plane-Less-VXLAN](#)

[2.3.1. Control-Plane-Less-Multicast-VXLAN](#)

[2.3.2. Control-Plane-Less-Unicast-VXLAN](#)

[2.4. Control-Plane-VXLAN](#)

[2.4.1. Controller-based-VXLAN](#)

[2.4.2. EVPN-VXLAN](#)

[2.4.3. Blockchain-VXLAN 'BLAN'](#)

[3. BLAN Value Proposition](#)

[4. BLAN Way Forward](#)

[5. What's next?](#)

## **About the Author**

Moustafa Amin, CCIE No. 14712 / OCSA No. 10516 / CBP, is a seasoned Network Consultant, with 18 years of experience in the Data Networking industry and an 'Emerging Technologies' Enthusiast.

Specializing in IP Routing Protocols, MPLS, Data Center, Cloud and SDN Technologies, Moustafa has designed or assisted in the design of many large-scale Telco Carrier, Service Provider and Enterprise networks all over the world.

Throughout his professional career, Moustafa has successfully managed elite groups of highly-skilled professionals & experts, developed creative business opportunities, technical articles, contents and delivered numerous courses, workshops & bootcamps.

## **Publication**

The 'Blockchain-VXLAN' proposal that will be referred to in this volume as 'BLAN' has been published by 'Moustafa Amin' (the creator and author) as an IP.COM Prior Art Technical Disclosure Document with number: IPCOM000250654D on the 16<sup>th</sup> of August, 2017.

## **Copyright**

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

## Introduction

Blockchain....How many times have you heard about it?

Probably for some people; they have heard about blockchain once or twice before and for others (including myself) they hear this term tens or maybe hundreds of times every day (I am among this group because of my regular job as an Emerging Technologies Business Developer and not because of my work on this series of books)

Some would say: we haven't actually heard about blockchain, for these folks; I want to ask them first: Have you heard about crypto currencies or maybe digital currencies? Something like: bitcoin, ether, litecoin, monero, dogecoin...etc

If you have heard about these currencies then you've heard about blockchain but in an indirect way simply because blockchain is the underlying technology for these digital/crypto currencies.

If you're willing to learn more about blockchain; you can find tons of valuable resources out there: books, articles, news and more importantly (for this unique technology): blogs, posts and forums.

In this regard, I highly recommend everyone to start with Satoshi Nakamoto's document:

<https://bitcoin.org/bitcoin.pdf>

<Reference from Wikipedia>

*Satoshi Nakamoto is the name used by the unknown person(s) who designed bitcoin and created its original reference implementation. As part of the implementation, they also devised the first blockchain database.*

</Reference from Wikipedia>

Back to our eBook; this is the first volume of the '**Blockchain: The Hidden Gem of Data Networking**' series of eBooks.

Note: This series assumes some degree of familiarity with the Data Networking

industry, so terms like: LAN, WAN, VLAN, VPN, MPLS, Control Plane, Overlay, Layer-2, Layer-3, Routing, Cloud, Data Center and others will be used directly without any further explanation.

In this series, we will thoroughly define blockchain and see it in action; we'll see how 'blockchain' could be leveraged to enhance the technologies that exist in the Data Networking, i.e. LAN and WAN technologies.

In this first volume with subtitle '**BLAN**' (short for Blockchain-VXLAN); we'll see how we could leverage blockchain to create an architecture that can connect geographically dispersed layer-2 islands over any available infrastructure (even the global public Internet).

Connecting layer-2 islands is not a new approach, we've seen many successful solutions that fall under the 'Overlay Networking Technologies' umbrella like: VXLAN, NVGRE...In this volume, we'll see 'BLAN' that is the result of the integration of blockchain with VXLAN.

VXLAN was originally drafted as an overlay technology that can work without a control-plane and its scope is usually within a single administrative domain i.e. Data Center or Cloud.

BLAN can greatly enhance the operation of the 'VXLAN' technology by adding a control-plane component to it, and it can also extend VXLAN working domain beyond the boundary of a local Data Center (or even a public cloud).

BLAN operation across the public Internet makes it more appealing as a viable WAN option for many customers (SME, large enterprises, service providers, Telcos...) in front of conventional WAN options (like: Dedicated links, MPLS or VPN across the Internet) or modern options (like: SD-WAN)

## 1. Blockchain

In this era of technology and change; the majority of us are 'Visual Learners'; we become excellent at visualizing objects, plans & outcomes, thus we'll not only limit ourselves to the most traditional way of defining things (i.e. Wikipedia) but we'll extend our blockchain definition to include some practical steps on how to run a private blockchain, see it in action and highlight all required features that are relevant to our goal.

So let's start by delving first into Wikipedia's definition of blockchain.

Kindly note that the output will be truncated to be relevant to the characteristics & features we need to highlight about blockchain

<Reference from Wikipedia>

### Description

*A blockchain is a decentralized and distributed digital ledger that is used to record transactions across many computers so that the record cannot be altered retroactively without the alteration of all subsequent blocks and the collusion of the network. This allows the participants to verify and audit transactions inexpensively. They are authenticated by mass collaboration powered by collective self-interests. The result is a robust workflow where participants' uncertainty regarding data security is marginal*

<Output Truncated>

*A blockchain database consists of two kinds of records: transactions and blocks. Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the original genesis block (first block). Some blockchains create a new block as frequently as every five*

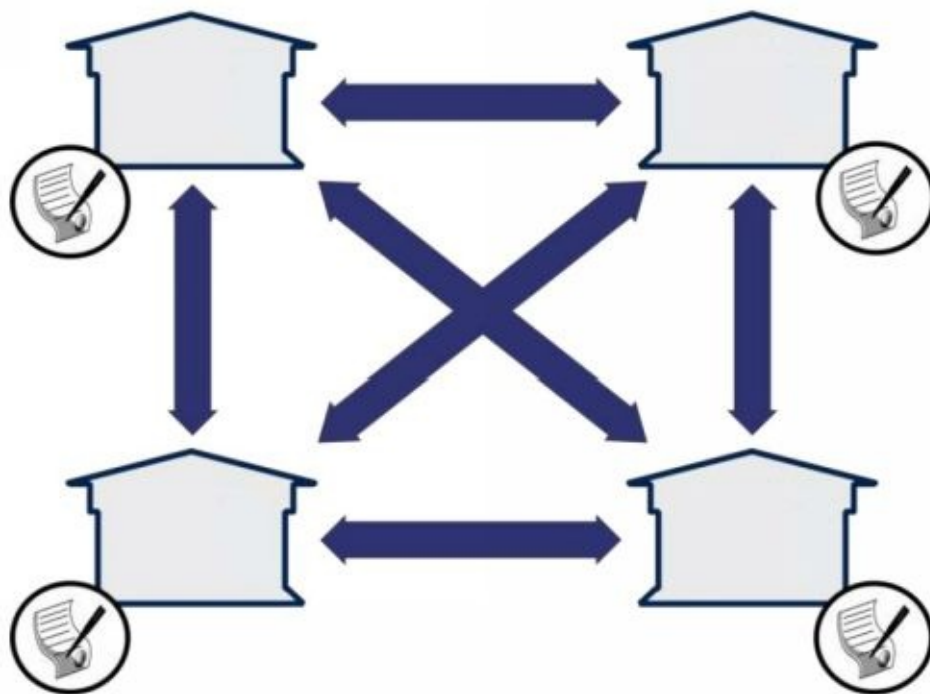


*seconds.*

<Output Truncated>

</Reference from Wikipedia>

<Simple diagram for clarification>



Distributed Ledger where every node has the 'exact' copy of the ledger

</Simple diagram for clarification>

<Reference from Wikipedia>

### Decentralization

*By storing data across its network, the blockchain eliminates the risks that come*

*with data being held centrally.*

*Its network lacks centralized points of vulnerability that computer hackers can exploit or any central point of failure. Blockchain security methods include the use of public-key cryptography. A public key (a long, random-looking string of numbers) is an address on the blockchain. Value tokens sent across the network are recorded as belonging to that address. A private key is like a password that gives its owner access to their digital assets or otherwise interact with the various capabilities that blockchains now support. Data stored on the blockchain is generally considered incorruptible.*

*Every node or miner in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication and computational trust. No centralized "official" copy exists and no user is "trusted" more than any other. Transactions are broadcast to the network using software. Messages are delivered on a best effort basis. Mining nodes validate transactions, add them to the block they're creating, and then broadcast the completed block to other nodes....*

*</Reference from Wikipedia>*

## **1.1. Centralization & Decentralization**

### **Mainframe, Cloud Computing, IP (Internet Protocol) & SDN (Software Defined Networking)**

I am always using these four distinct things when I want to illustrate the concepts of Centralization and Decentralization.

A **Mainframe**: originally used to be a large cabinet that housed a 'huge' early computer with powerful processor, lots of memory & storage (I will not share any figures about the first mainframes here as you'll find them funny compared to the capabilities of a regular laptop nowadays)...The idea was simply to use this mainframe to access, compute and store all data centrally on this huge machine.

This term is not obsolete; it is still used to distinguish these high-end computational machines.

On the other hand, **Cloud Computing**: is an Information technology paradigm that allows access to shared pools of configurable resources such as: computing

nodes, networks, servers, storage, applications and services

Cloud computing relies on the concept of sharing the resources to achieve the required economy of scale.

You can clearly see that mainframe was all about the idea of centralizing the computational resources while Cloud is all about decentralizing it.

The same happens on the Data Networking side:

The **IP protocol**: is the principal communications protocol that essentially establishes the Internet.

The IP protocol allows the nodes themselves to be responsible of the delivery of data from an end to the other across a network. The nodes have the intelligence to do that on their own without the need of a central authority.

On the other hand, **SDN**: is a Networking approach that allows a single controller (or for the sake of redundancy; a cluster of controllers) to programmatically initialize, control, change and manage networking nodes via open interfaces.

You can clearly see that IP could be perceived as the standard definition of the networking decentralization while SDN is the typical definition of centralization.

Bottom line; there is nothing wrong about centralizing or decentralizing things... Every step taken during the different technologies life-cycle was taken purposely to solve an issue or to make life easier...Mainframes still exist (in some specific cases) along with Cloud Computing that is becoming dominant.

The IP is still (and will be for more decades to come) the main communications protocol for the Internet while SDN is growing & being adapted by thousands of environments every day.

## **1.2. Blockchain main Functions**

Based on the above definition of blockchain; It's now worth mentioning that blockchain transactions don't only carry assets (or tokens) exchanged between the nodes but they can carry any Data or information that can be appended by the different nodes & that's exactly what we'll use blockchain for in this series of eBooks.

To elaborate more on the above, we can say that basically; there are three main functions of blockchain:

- Storage for digital records (that will be our feature of interest in this series of eBooks)
- Exchanging digital assets (coins and/or tokens)
- Executing smart contracts (quick definition):
  - Ground rules - Terms & conditions recorded in code.
  - The Distributed Network will execute contract and monitor compliance.
  - Outcomes are automatically validated without third party.

For the purpose of illustrating the blockchain operation; we used an open platform for blockchain applications named 'multichain' (multichain.com)

Basically we need two server nodes in order to download and install multichain on each, but definitely in a live & production environment; there will be hundreds & even thousands of nodes attached to the same blockchain as the goal is to make it as 'distributed' as it can be (there are thousands of nodes currently operating on bitcoin blockchain network)

For the sake of simplicity and quick operation:

I used a 'windows-based' laptop as Server-1 and an Ubuntu Linux Virtual Machine (VM) (created over Oracle Virtual Box) as Server-2.

I downloaded & installed 'multichain' on both Servers (Server-1 and Server-2) & here we go!

We'll take a close look and examine the first couple of functions of blockchain; digital assets exchange and digital data recording.

### 1.2.1. Asset Creation & Transactions

1. On Server-1; let's create a new blockchain & give it the name 'chain20'

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-util create chain20
MultiChain utilities build 1.0 beta 2 protocol 10008

Blockchain parameter set was successfully generated.
You can edit it in C:\Users\Moustafa\AppData\Roaming\MultiChain\chain20\params.d
at before running multichaind for the first time.

To generate blockchain please run "multichaind chain20 -daemon".
```

2. We got the instruction that the chain parameter was successfully generated but we'd need to run a certain command to generate the blockchain itself

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichaind chain20 -daemon
MultiChain Core Daemon build 1.0 beta 2 protocol 10008
Looking for genesis block...
Genesis block found
Other nodes can connect to this node using:
multichaind chain20@192.168.1.7:6447
This host has multiple IP addresses, so from some networks:
multichaind chain20@192.168.56.1:6447
Node started
```

3. Now, we've been told that the genesis block (first block) was found & we've also been presented with the IP Address & port number to use if we'd like to connect to this node from other nodes (Server-2, for example, as we'll do right below)

```
root@tika-VirtualBox:/usr/local/bin# multichaind chain20@192.168.1.7:6447
MultiChain Core Daemon build 1.0 beta 2 protocol 10008
Retrieving blockchain parameters from the seed node 192.168.1.7:6447 ...
Blockchain successfully initialized.
Please ask blockchain admin or user having activate permission to let you connect and/or transact:
multichain-cli chain20 grant 1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN connect
multichain-cli chain20 grant 1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN connect,send,
receive
```

4. The chain20 has been initialized on Server-2 but it gave us the message that we should ask the administrator of the blockchain to grant Server-2 'connectivity' as well as 'send' and 'receive' rights on chain20.

Note that the cryptographic address of Server-2 has been presented (the address usually starts with '1') that is:

**1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN**, so now we have to copy this address precisely and paste it on Server-1 in order to grant the 'access' rights to the correct node

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 grant 1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN connect
{"method": "grant", "params": ["1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN", "connect"], "id": 1, "chain_name": "chain20"}
b1d16c59d08ee8cbeb3079c3901c03fb21a01c0fa6b562d672b5aad18804684f
```

5. Connecting again from Server-2, Node started!

```
root@tika-VirtualBox:/usr/local/bin# multichaind chain20 -daemon
MultiChain Core Daemon build 1.0 beta 2 protocol 10008

MultiChain server starting
Retrieving blockchain parameters from the seed node 192.168.1.7:6447 ...
Other nodes can connect to this node using:
multichaind chain20@192.168.1.8:6447

Node started
```

6. Back to Server-1, and because we are going to create an asset and send it between nodes (via transactions), let us see what node/address has the permission to create an asset on chain20

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 listpermissions issue
{"method":"listpermissions","params":["issue"],"id":1,"chain_name":"chain20"}
[
  {
    "address" : "17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1",
    "for" : null,
    "type" : "issue",
    "startblock" : 0,
    "endblock" : 4294967295
  }
]
```

7. We can see that Server-1 address (17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1) has the permission to create 'issue' assets, so let's create a new asset named 'asset1' on Server-1 with 1000 units, each of which can be subdivided into 100 parts and send it to self

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 issue 17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1 asset1 1000 0.01
{"method":"issue","params":["17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1","asset1",1000,0.01000000],"id":1,"chain_name":"chain20"}
53416c73af4d1cf42d54ce1bbd5d11f95712b1a7478471e5ddda6277365ca75f
```

8. On Server-1, let's verify that this new asset named 'asset1' is listed



```

D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 listassets
{"method":"listassets","params":[],"id":1,"chain_name":"chain20"}
[
  {
    "name" : "asset1",
    "issuetxid" : "53416c73af4d1cf42d54ce1bbd5d11f95712b1a7478471e5ddda6277365ca75f",
    "assetref" : "60-265-16723",
    "multiple" : 100,
    "units" : 0.01000000,
    "open" : false,
    "details" : {
    },
    "issueqty" : 1000.00000000,
    "issueraw" : 100000,
    "subscribed" : false
  }
]

```

9. Doing the same from Server-2

```

Interactive mode
chain20: listassets
{"method":"listassets","params":[],"id":1,"chain_name":"chain20"}
[
  {
    "name" : "asset1",
    "issuetxid" : "53416c73af4d1cf42d54ce1bbd5d11f95712b1a7478471e5ddda6277365ca75f",
    "assetref" : "60-265-16723",
    "multiple" : 100,
    "units" : 0.01000000,
    "open" : false,
    "details" : {
    },
    "issueqty" : 1000.00000000,
    "issueraw" : 100000,
    "subscribed" : false
  }
]

```

10. Let's check the asset1 balance from Server-1; it should be: 1000 (the full balance)

```

D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 gettotalbalances
{"method":"gettotalbalances","params":[],"id":1,"chain_name":"chain20"}
[
  {
    "name" : "asset1",
    "assetref" : "60-265-16723",
    "qty" : 1000.00000000
  }
]

```

11. Now, let's check the asset1 balance from Server-2; it should be: 0 (no output meaning no balance has ever been transferred to this node yet)

```
chain20: gettotalbalances
{"method": "gettotalbalances", "params": [], "id": 1, "chain_name": "chain20"}
[
]
```

12. Let's send 100 units from asset1 from Server-1 (the asset creator and owner) to Server-2 wallet (using Server-2 wallet address)

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 sendasset 1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN asset1 100
{"method": "sendasset", "params": ["1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN", "asset1", "100"], "id": 1, "chain_name": "chain20"}
error code: -704
error message:
Destination address doesn't have receive permission
```

13. We got an error message that Server-2 (the destination wallet) doesn't have 'receive' permission, so let's grant Server-2 both 'receive' as well as 'send' permissions

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 grant 1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN receive,send
{"method": "grant", "params": ["1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN", "receive,send"], "id": 1, "chain_name": "chain20"}
0d5ecac1f9ac0aa2617c6c7beb1eedeb119885bc7d40718cd4f9cee3b87aae7a
```

14. Let's re-try sending the 100 units again from Server-1 to Server-2

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 sendasset 1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN asset1 100
{"method": "sendasset", "params": ["1NfAmUDLp8mwILDoRwJroiEBsmis8Zzg7UHhXN", "asset1", "100"], "id": 1, "chain_name": "chain20"}
c3a43b2f57dd011a23f7c4ae95ac5d41407882ad2e631ab34883885d9d95c06c
```

15. Now, let's check the asset1 balance on Server-1 (it should now be 900 after deducting the 100 units we just sent to Server-2)

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 gettotalbalances
{"method": "gettotalbalances", "params": [], "id": 1, "chain_name": "chain20"}
[
  {
    "name": "asset1",
    "assetref": "60-265-16723",
    "qty": 900.00000000
  }
]
```

16. And the balance of asset1 from Server-2 (it should be the 100 units just received)



```
chain20: gettotalbalances
{"method": "gettotalbalances", "params": [], "id": 1, "chain_name": "chain20"}

[
  {
    "name" : "asset1",
    "assetref" : "60-265-16723",
    "qty" : 100.00000000
  }
]
```

### 1.2.2. Data Storage & Retrieval

1. On Server-1; let's create a new blockchain & give it the name 'chain100'

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-util create chain100
MultiChain utilities build 1.0 beta 2 protocol 10008
Blockchain parameter set was successfully generated.
You can edit it in C:\Users\Moustafa\AppData\Roaming\MultiChain\chain100\params.
dat before running multichaind for the first time.
To generate blockchain please run "multichaind chain100 -daemon".
```

2. As usual, we got the instruction that the chain parameter was successfully generated but we'd need to run a certain command to generate the blockchain.

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichaind chain100 -daemon
MultiChain Core Daemon build 1.0 beta 2 protocol 10008
Looking for genesis block...
Genesis block found
Other nodes can connect to this node using:
multichaind chain100@192.168.1.7:7179
This host has multiple IP addresses, so from some networks:
multichaind chain100@192.168.56.1:7179
Node started
```

3. The genesis block (first block) was found & we've also been presented with the IP Address & port number to use if we'd like to connect to this node from other nodes, i.e. Server-2 as follows:

```

root@tika-VirtualBox:/usr/local/bin# multichaind chain100@192.168.1.7:7179

MultiChain Core Daemon build 1.0 beta 2 protocol 10008

Retrieving blockchain parameters from the seed node 192.168.1.7:7179 ...
Blockchain successfully initialized.

Please ask blockchain admin or user having activate permission to let you connect and/or transact:
multichain-cli chain100 grant 1JpJNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP connect
multichain-cli chain100 grant 1JpJNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP connect,send,receive

```

4. The chain100 has been initialized on Server-2 but it gave us the message that we should ask the administrator of the blockchain to grant Server-2 'connectivity' as well as 'send' and 'receive' rights on chain100.

Note that we have to copy and paste Server-2 address precisely on Server-1 in order to grant the 'access' rights to the correct node

```

D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 grant 1JpJNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP connect
{"method": "grant", "params": ["1JpJNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP", "connect"], "id": 1, "chain_name": "chain100"}
b3c6248613e64f503d9c5a3b63a913949b40197e91ff14741e0ef279a5de0c37

```

5. Connecting again from Server-2....Node started!

```

root@tika-VirtualBox:/usr/local/bin# multichaind chain100 -daemon

MultiChain Core Daemon build 1.0 beta 2 protocol 10008

MultiChain server starting
Retrieving blockchain parameters from the seed node 192.168.1.7:7179 ...
Other nodes can connect to this node using:
multichaind chain100@192.168.1.8:7179

Node started

```

6. Back to Server-1, let's look at some details of chain100:

Note the number of generated blocks so far for chain100, the total assets (zero as we're now demonstrating the key & data exchange between the different nodes on the same blockchain)

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 getinfo
{"method": "getinfo", "params": [], "id": 1, "chain_name": "chain100"}
{
  "version" : "1.0 beta 2",
  "nodeversion" : 10000202,
  "protocolversion" : 10008,
  "chainname" : "chain100",
  "description" : "MultiChain chain100",
  "protocol" : "multichain",
  "port" : 7179,
  "setupblocks" : 60,
  "nodeaddress" : "chain100@192.168.1.7:7179",
  "burnaddress" : "1XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXc5xBW1",
  "incomingpaused" : false,
  "miningpaused" : false,
  "walletversion" : 60000,
  "balance" : 0.00000000,
  "walletdbversion" : 2,
  "reindex" : false,
  "blocks" : 35,
  "timeoffset" : 0,
  "connections" : 1,
  "proxy" : "",
  "difficulty" : 0.00000006,
  "testnet" : false,
  "keypoololdest" : 1504377971,
  "keypoolsize" : 2,
  "paytxfee" : 0.00000000,
  "relayfee" : 0.00000000,
  "errors" : ""
}
```

7. Let's now create a stream & call it 'stream100' on Server-1

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 create
stream stream100 false
{"method":"create","params":["stream","stream100",false],"id":1,"chain_name":"chain100"}
677609b6e7689f60ab03983f16159f1eb23ca522bedd0c3abbfab04df0bd9360
```

8. Let's publish some information on Stream100 (a key and some hexadecimal data)

Please note that the chosen hexadecimal data

**‘637573746f6d65722031353520766e6964203130343030’** isn’t random, indeed it has some meaning (but let’s postpone its interpretation for now)

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 publish
stream100 key1555 637573746f6d6572203135353520766e6964203130343030
{"method": "publish", "params": ["stream100", "key1555", "637573746f6d6572203135353520766e6964203130343030"], "id": 1, "chain_name": "chain100"}
f6c0a07d4d2a5168eb555004edfc077d680920dda6195b5352315f89c06299e6
```

9. Let's check if Server-2 can successfully see & fetch information about the published stream100

Please note the number of keys (1) in stream100

```
chain100: liststreams stream100
{"method": "liststreams", "params": ["stream100"], "id": 1, "chain_name": "chain100"}

[
  {
    "name": "stream100",
    "createtxid": "677609b6e7689f60ab03983f16159f1eb23ca522bedd0c3abbfab04d
f0bd9360",
    "streamref": "48-266-30311",
    "open": false,
    "details": {
    },
    "subscribed": true,
    "synchronized": true,
    "items": 1,
    "confirmed": 1,
    "keys": 1,
    "publishers": 1
  }
]
```

10. Server-2 can even dig deeper and fetch more detailed information regarding the keys inside stream100.

It can be seen that key1555 does contain '1' item (that is the published hexadecimal data)

```
chain100: liststreamkeys stream100
{"method": "liststreamkeys", "params": ["stream100"], "id": 1, "chain_name": "chain100"}

[
  {
    "key": "key1555",
    "items": 1,
    "confirmed": 1
  }
]
```

11. Server-2 can successfully and clearly see the hexadecimal data included in Key1555 published by Server-1 (the publisher) inside stream100 on chain100

```
chain100: liststreamkeyitems stream100 key1555
{"method": "liststreamkeyitems", "params": ["stream100", "key1555"], "id": 1, "chain_name": "chain100"}

[
  {
    "publishers": [
      "15yBH99oGQsKjswa8rt9xMD6yeV33JbT9PnRNU"
    ],
    "key": "key1555",
    "data": "637573746f6d6572203135353520766e6964203130343030",
    "confirmations": 11,
    "blocktime": 1504379183,
    "txid": "f6c0a07d4d2a5168eb555004edfc077d680920dda6195b5352315f89c06299
e6"
  }
]
```



12. We want to make sure the stream communication over blockchain is allowed on every node participating to the blockchain.

From Server-1, let's make sure that Server-2 can send & receive over chain100

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 grant 1JpjNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP receive,send
{"method":"grant","params":["1JpjNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP","receive,send"],"id":1,"chain_name":"chain100"}
505fc7cd3be80ce63bf9d205e8675f2ac4b911ebc0aa1e4db23d570b33a30489
```

13. On Server-1, let's also allow Server-2 to publish on stream100

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 grant 1JpjNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP stream100.write
{"method":"grant","params":["1JpjNxbdmvTPvWmtQg5qoyxZLwRv4cenzHd3ZP","stream100.write"],"id":1,"chain_name":"chain100"}
92466201e53026de6ed3d621d2e90b8d7a35fc7845a7c57dd3b6bdd34940369a
```

14. Now, from Server-2; let's publish some hexadecimal data (again, let's leave its meaning for now but rest assured it has a useful meaning)

'637573746f6d6572203135353520766e6964203132353030' on the same stream & key pair (stream100, key1555)

```
chain100: publish stream100 key1555 637573746f6d6572203135353520766e6964203132353030
{"method":"publish","params":["stream100","key1555","637573746f6d6572203135353520766e6964203132353030"],"id":1,"chain_name":"chain100"}
43118d7afd52630b833d6b161929d5b2abe34b4fa698fb26fc8f89568f3f48ed
```

15. Back to Server-1, let's now subscribe to stream100 so that Server-1 can see what other nodes (only Server-2 in our case) can publish on that stream.

Please note the detailed information when we list streams on Server-1 and the current number of publishers (2) on stream100 (that indicates that both nodes Server-1 and Server-2 are current publishers)

```

D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 subscribe stream100
{"method": "subscribe", "params": ["stream100"], "id": 1, "chain_name": "chain100"}

D:\Program Sources\multichain-windows-1.0-beta-2>
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 liststreams
{"method": "liststreams", "params": [], "id": 1, "chain_name": "chain100"}

[
  {
    "name": "root",
    "createtxid": "30d22936196120b16f4d94190fb71770a7a8ada7dca55af7dfbcb2b21ed1b1",
    "streamref": "0-0-0",
    "open": true,
    "details": {
    },
    "subscribed": true,
    "synchronized": true,
    "items": 0,
    "confirmed": 0,
    "keys": 0,
    "publishers": 0
  },
  {
    "name": "stream100",
    "createtxid": "677609b6e7689f60ab03983f16159f1eb23ca522bedd0c3abbfab04df0bd9360",
    "streamref": "48-266-30311",
    "open": false,
    "details": {
    },
    "subscribed": true,
    "synchronized": true,
    "items": 2,
    "confirmed": 2,
    "keys": 1,
    "publishers": 2
  }
]

```

16. Server-1 can delve into stream100 and see its keys (key1555) as well as the key contents (2 items) these are the two hexadecimal data published by Server-1 and Server-2.

```

D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 liststreamkeys stream100
{"method":"liststreamkeys","params":["stream100"],"id":1,"chain_name":"chain100"}

[
  {
    "key" : "key1555",
    "items" : 2,
    "confirmed" : 2
  }
]

D:\Program Sources\multichain-windows-1.0-beta-2>
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 liststreamkeyitems stream100 key1555
{"method":"liststreamkeyitems","params":["stream100","key1555"],"id":1,"chain_name":"chain100"}

[
  {
    "publishers" : [
      "15yBH99oGQsKjswa8rt9xMD6yeU33JbT9PnRNU"
    ],
    "key" : "key1555",
    "data" : "637573746f6d6572203135353520766e6964203130343030",
    "confirmations" : 37,
    "blocktime" : 1504379183,
    "txid" : "f6c0a07d4d2a5168eb555004edfc077d680920dda6195b5352315f89c06299e6"
  },
  {
    "publishers" : [
      "1JpJNxhdMvTPvWMtQg5qoYxZLvRv4cenzHd3ZP"
    ],
    "key" : "key1555",
    "data" : "637573746f6d6572203135353520766e6964203132353030",
    "confirmations" : 11,
    "blocktime" : 1504380439,
    "txid" : "43118d7afd52630b833d6b161929d5b2abe34b4fa698fb26fc8f89568f3f48ed"
  }
]

```

### 1.3. Mining over blockchain

Mining is simply the process of producing blocks and hard coding them to the blockchain itself.

‘Miners’ are those critical nodes participating on a blockchain that mainly perform two different functions:

- An easy function of verifying the transactions that have been broadcasted over the blockchain network by the different nodes.
- A hard (sometimes extremely hard as the case with Public Crypto Currency blockchain, like bitcoin for example) function of packing the different verified transactions into blocks and hard-code (glue) these blocks to the blockchain.

There are many variations of mining, like PoW (Proof of Work) and PoS (Proof of Stake)...We won’t talk about them here but we’ll take a look on how the

mining process works for our private blockchain (multichain)

1. From Server-1, let's discover who has permission to mine on chain20, it shows that only this node (Server-1) with its address **(17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1)** can mine over this blockchain.

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 listpermissions mine
{"method":"listpermissions","params":["mine"],"id":1,"chain_name":"chain20"}
[
  {
    "address" : "17EtAjgKRWQTKQ6BxwK5HCWSEkZJLyAokGxqu1",
    "for" : null,
    "type" : "mine",
    "startblock" : 0,
    "endblock" : 4294967295
  }
]
D:\Program Sources\multichain-windows-1.0-beta-2>
```

2. Let's now grant Server-2 'mine' permission (remember that Server-2 address is: **1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN**)

```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 grant 1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN mine
{"method":"grant","params":["1NfAmUDLp8mwTLDoRwJroiEBsmis8Zzg7VHhXN","mine"],"id":1,"chain_name":"chain20"}
e7c1e5d38c7df9190619e055e7cff30fa7534bdbec7e6bd2ff5c37467bb8dbe0
D:\Program Sources\multichain-windows-1.0-beta-2>
```

3. From both servers (Server-1 and Server-2), let's run a special command that will allow the maximum degree of mining randomness, so from Server-1:

```
D:\Program Sources\multichain-windows-1.0-beta-2>
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain20 setruntimeparam miningturnover 1
{"method":"setruntimeparam","params":["miningturnover","1"],"id":1,"chain_name":"chain20"}
D:\Program Sources\multichain-windows-1.0-beta-2>
```

4. The same from Server-2:

```
chain20: setruntimeparam miningturnover 1
{"method":"setruntimeparam","params":["miningturnover","1"],"id":1,"chain_name":"chain20"}
chain20:
```

5. Let's now check the current block height (the latest produced block number) as well as other useful info from Server-2:





```
chain20: getblock 97  
{"method": "getblock", "params": ["97"], "id": 1, "chain_name": "chain20"}  
  
{  
  "hash": "0070d4814126b9bafedc3d375e0ae0105bbba9716ee1f472e819f368f20f1ed0013",  
  "miner": "17ETAjgKRWQTKQ6BxwK5HCWSEKZJLyAokGxqu1",  
  "confirmations": 2,  
  "size": 266,  
  "height": 97,  
  "version": 3,  
  "merkleroot": "29157ed52d24b4c7baa25669f4a2589f94762f5ede5ffad43c8a47748df7f488",  
  "tx": [  
    "29157ed52d24b4c7baa25669f4a2589f94762f5ede5ffad43c8a47748df7f488"  
  ],  
  "time": 1507028111,  
  "nonce": 450,  
  "bits": "2000ffff",  
  "difficulty": 0.00000006,  
  "chainwork": "0000000000000000000000000000000000000000000000000000000000000006200",  
  "previousblockhash": "00717b96e3f98ac649d75d153cd4f75791ac9c0af52d1936e9c7158e5bbf3  
a44",  
  "nextblockhash": "001b0da6acbed4bcee743e9f12f394be05a938c603ed88cc98f3749f492c3f42"  
}  
chain20:
```

## 2. VXLAN

As its name indicates, Virtual eXtensible Local Area Network (VXLAN) is designed to provide the same Ethernet Layer 2 network services as VLAN does today, but with greater extensibility and flexibility. Compared to VLAN, VXLAN offers the following benefits:

Flexible placement of multitenant segments throughout the data center: It provides a solution to extend Layer 2 segments over the underlying shared network infrastructure so that tenant workload can be placed across physical pods in the data center.

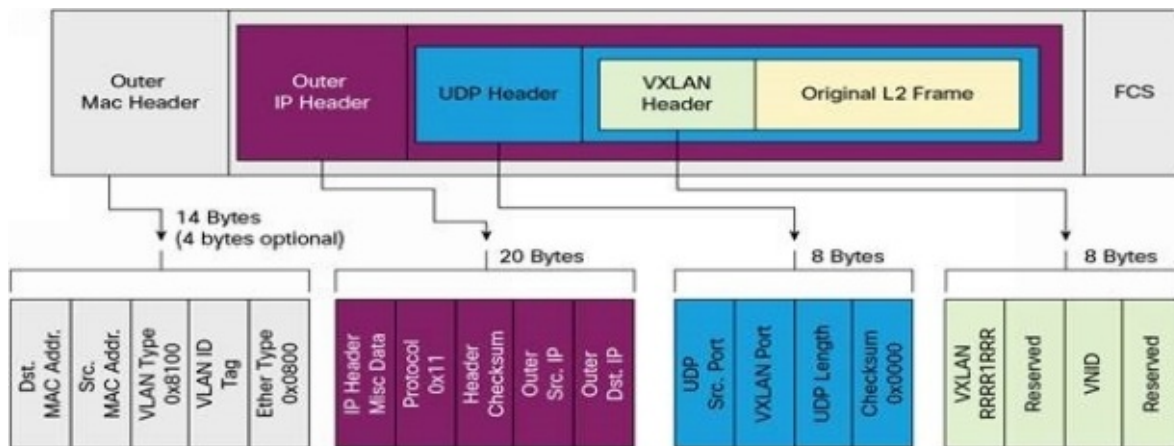
Higher scalability to address more Layer 2 segments: VLANs use a 12-bit VLAN ID to address Layer 2 segments, which results in limiting scalability of only 4094 VLANs. VXLAN uses a 24-bit segment ID known as the VXLAN Network IDentifier (VNID), which enables up to 16 million VXLAN segments to coexist in the same administrative domain.

Better utilization of available network paths in the underlying infrastructure: VLAN uses the Spanning Tree Protocol (STP) for loop prevention, which ends up with not using half of the network links by blocking redundant paths. In contrast, VXLAN packets are transferred through the underlying network based on its Layer 3 header and can take complete advantage of Layer 3 routing, equal-cost multipath (ECMP) routing, and link aggregation protocols to use all available paths.

### 2.1. VXLAN Format, Traffic Flow & ECMP

VXLAN encapsulation adds 50 bytes to the original L2 frame by adding 4 different headers:

- VXLAN header
- UDP header
- Outer IP header
- Outer MAC header



Encapsulated VXLAN packets are forwarded between VTEPs (VXLAN Tunnel End Point) based on the native forwarding decisions of the transport network. And because the VXLAN was originally created to be bounded inside the data center where the underlay transport networks are designed and deployed with multiple redundant paths and take advantage of various multipath load-sharing technologies to distribute traffic loads on all available paths. It is desirable to share the load of the VXLAN traffic in the same fashion in the transport network.

A typical VXLAN transport network is an IP-routing network that uses the standard IP ECMP to balance the traffic load among multiple best paths. To avoid out-of-sequence packet forwarding, flow-based ECMP is commonly deployed. An ECMP flow is defined by the source and destination IP addresses and optionally the source and destination TCP or UDP ports in the IP packet header.

Because all the VXLAN packet flows between a pair of VTEPs have the same outer source and destination IP addresses, and all VTEP devices must use one identical destination UDP port that can be either the Internet Assigned Numbers Authority (IANA)-allocated UDP port 4789 (or a customer-configured port), the only variable element in the ECMP flow definition that can differentiate VXLAN flows from the transport network standpoint is the source UDP port. A similar situation for Link Aggregation Control Protocol (LACP) hashing occurs if the resolved egress interface based on the routing and ECMP decision is a LACP port channel. The LACP uses the VXLAN outer-packet header for link load-share hashing, which results in the source UDP port being the only element that can uniquely identify a VXLAN flow.

## **2.2. VXLAN Modes of Operation**

From the various industry implementations of VXLAN, we can categorize the VXLAN modes of operation into two main categories:

- Control-Plane-Less-VXLAN
- Control-Plane-VXLAN

The differences are mainly in the underlay transport network multicast capability, how to deal with BUM (Broadcast, Unknown Unicast & Multicast) traffic as well as the method of discovery and distribution of MAC addresses.

## **2.3. Control-Plane-Less-VXLAN**

For the Control-Plane-less mode of operation of VXLAN, we have two main sub-modes:

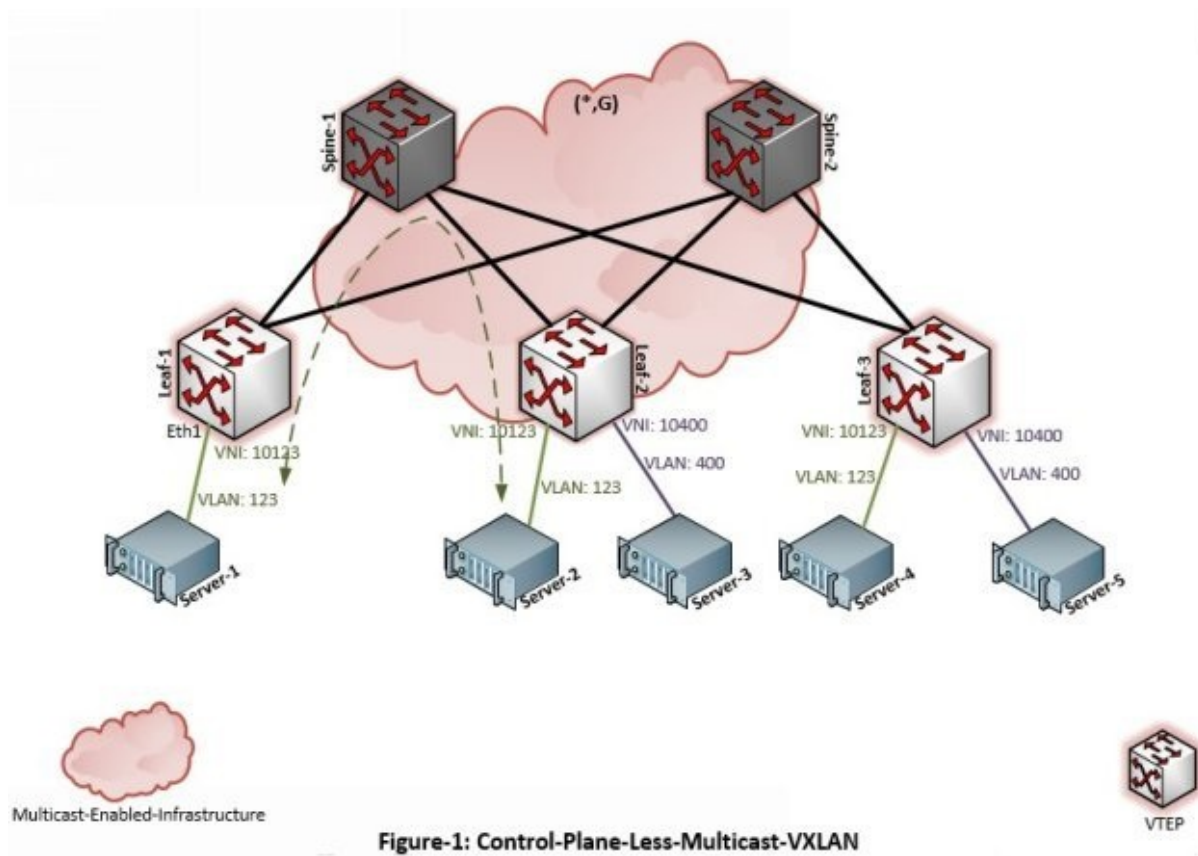
- Control-Plane-Less-Multicast-VXLAN
- Control-Plane-Less-Unicast-VXLAN

### **2.3.1. Control-Plane-Less-Multicast-VXLAN**

As its name implies; there is NO control or signaling established prior to the VXLAN operation.

This mode is according to the original VXLAN draft as per RFC7348.

This mode requires the underlay transport network to fully support the IP Multicast & every VTEP node to join the proper Multicast domain.



In this mode; the BUM (Broadcast, Unknown Unicast & Multicast) traffic is always carried over Multicast.

It's all about the 'Data-Plane' learning (or flow-based learning) that is based on the 'Flood & learn' technique; where the remote VTEPs would know about a MAC address because of the conversational MAC address learning approach:

The destination (receiving) VTEP learns the inner Source MAC of any received VXLAN IP packet (for example a Broadcasted ARP request message carried over Multicast). The source MAC address is then mapped to the Source (Originating) VTEP that originated the VXLAN packet.

The Originating VTEP will learn the remote MAC address to VTEP mapping once it receives the VXLAN encapsulated 'Unicast' ARP reply message from the receiving VTEP.

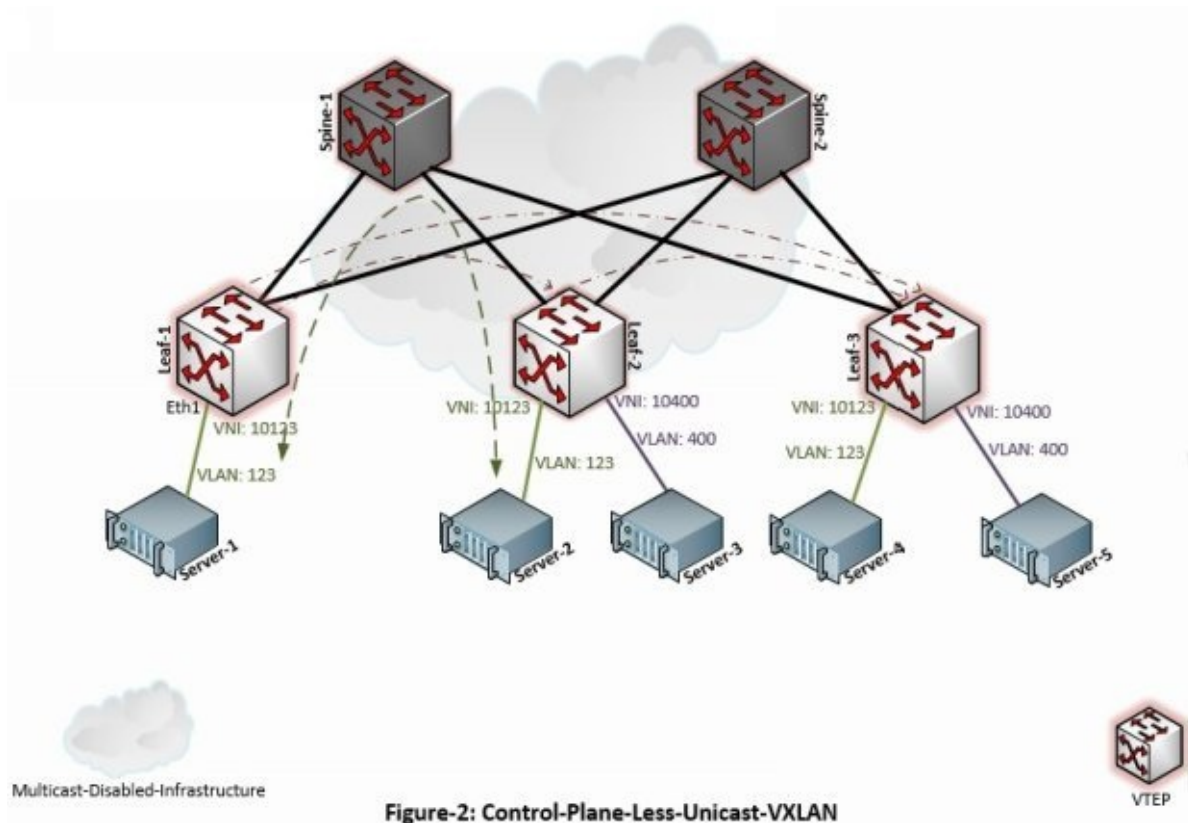
All subsequent traffic to a known MAC address will be Unicast IP encapsulated VXLAN.

### 2.3.2. Control-Plane-Less-Unicast-VXLAN



Exactly like the Control-Plane-Less-Multicast-VXLAN; there is NO control or signaling established prior to the VXLAN operation, instead; a list of all available & participating VTEPs are configured on each VTEP per supported VXLAN.

In this mode; the underlay transport network doesn't need to support IP Multicast.



For the BUM traffic; instead of being 'Multicast' over the underlay transport network as in the previous mode of operation, the 'head-end replication' is used here where the originating VTEP has to replicate the VXLAN packet & sends a copy to every other VTEP participating in this same VXLAN.

The list of VTEPs must be configured (changed & updated) manually on each & every VTEP in the VXLAN domain.

The 'Data-Plane' learning technique is also here in this mode of operation.

## 2.4. Control-Plane-VXLAN

In this mode; there is no need for the IP Multicast in the underlay transport

network; for any traffic that would require to be sent to all VTEPs (like Broadcast or Multicast); the head-end replication will be used instead (as in the previous mode)

Dealing with the Unknown Unicast traffic is what really differentiates this mode of operation from the previous modes. In this mode, a 'Control-Plane' does exist to distribute the MAC-to-VTEP mapping entries between the different VTEPs, hence no need for any data-plane learning technique (like flood & learn).

This control plane piece could be a **Controller** (like VMware NSX, Midokura, Nuage, Openstack...), a signaling protocol (like MP-BGP in the **EV**PN-based VXLAN) or a **Blockchain** as in the proposed **Blockchain-VXLAN** (**BLAN** as will be detailed).

#### 2.4.1. Controller-based-VXLAN

In the Controller-based VXLAN service, Data-Plane (flow-based) learning is optional or even not needed; the controller synchronizes all the MAC addresses as soon as the different switches learn them locally from their local ports.

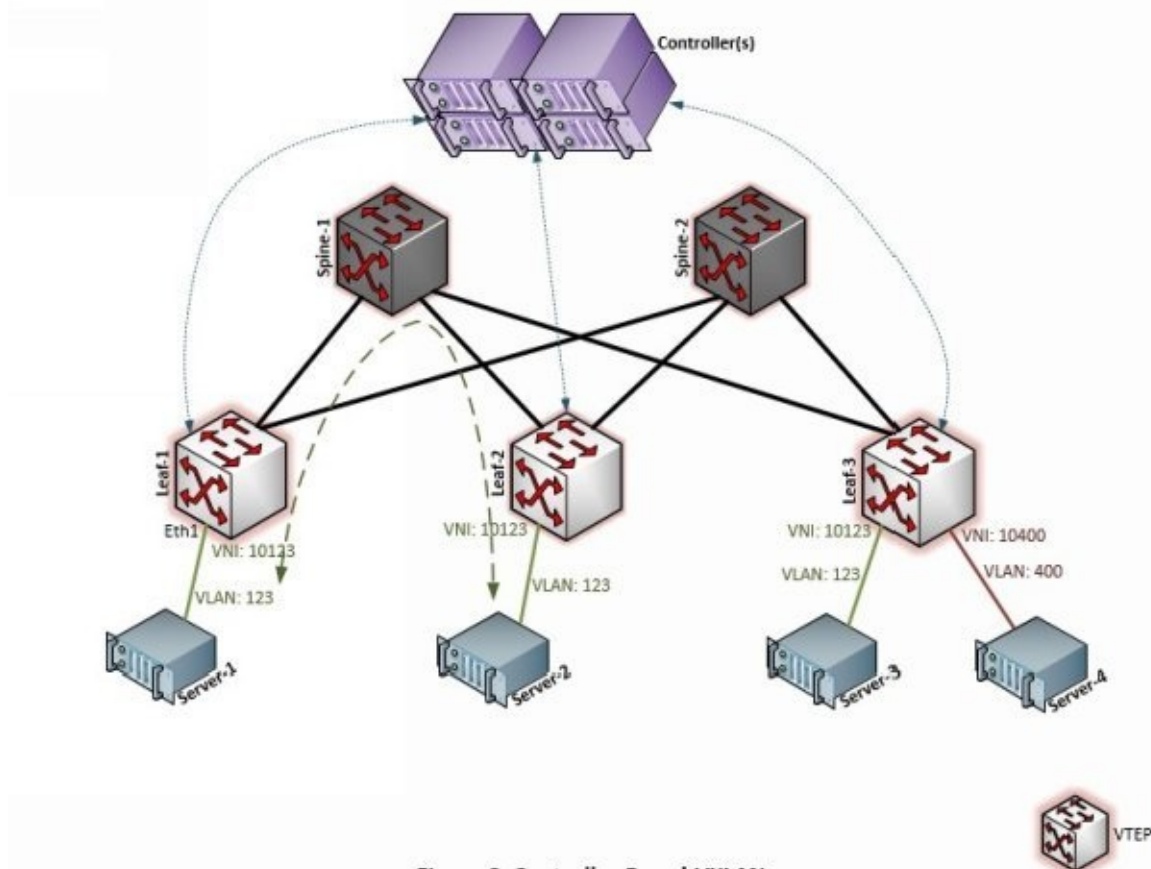


Figure-3: Controller-Based-VXLAN



For example in Figure-3, Leaf-1 learns the MAC address of Server-1 from its local port Eth1. This information is automatically and immediately synchronized with the controller that in turn pushes that info to Leaf-2, Leaf-3 & any other VTEP in the same VXLAN domain.

This VXLAN operation depends on the distribution of all learnt MAC addresses from the different VTEPs via the controller that ‘pushes’ to all VTEPs a complete (and always updated) list of MAC-to-VTEP mapping entries.

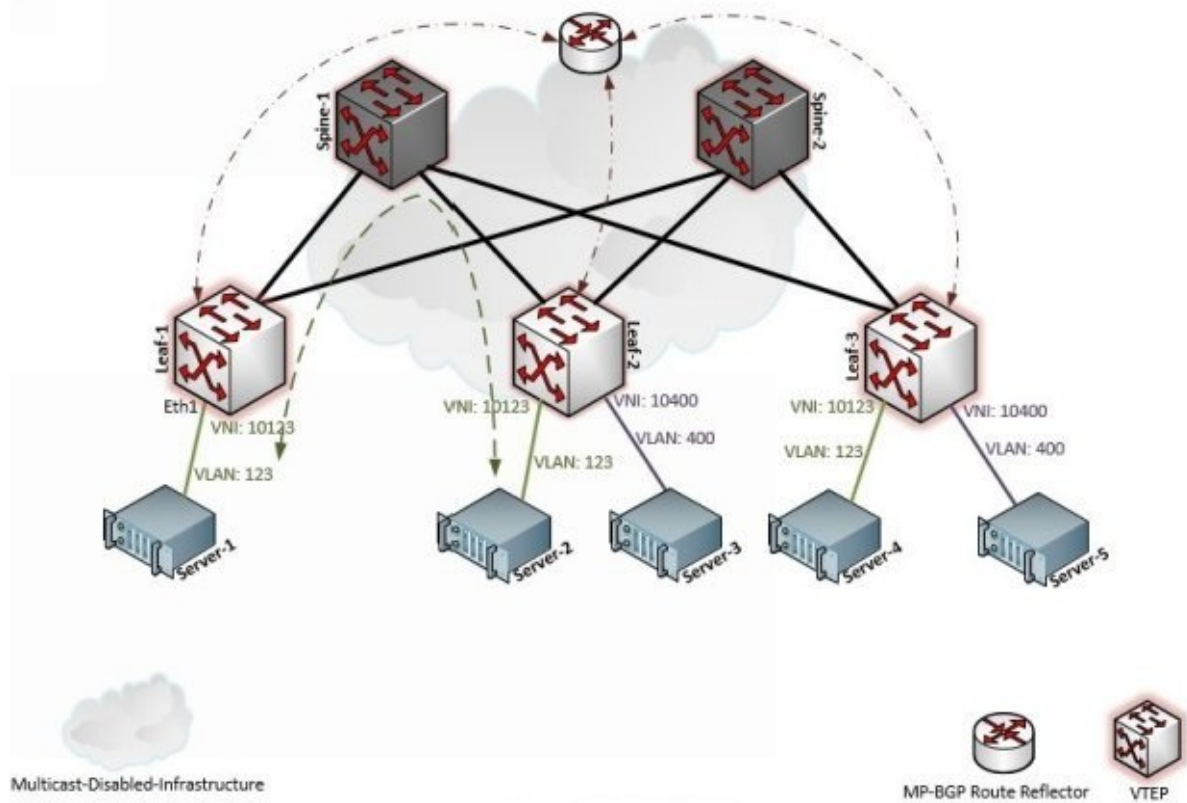
Because of that, for this mode; there would be no Unknown Unicast as the list of all communicating MACs is always present and updated on each VTEP, but in case of an unknown MAC (maybe for a destination outside the local VXLAN domain) & depending on the configuration; the local VTEP can direct it via the default entry towards the VXLAN gateway.

For the other Broadcast & Multicast traffic; the head-end replication is always the solution.

#### **2.4.2. EVPN-VXLAN**

In the EVPN-VXLAN; each VTEP is now a PE (Provider Edge) node & will learn the local MAC addresses associated to its VXLANs from its local ports as usual.

Using MP-BGP EVPN address family; these entries will be propagated between the different PEs (ideally through a set of MP-BGP RR ‘Route Reflectors’)



**Figure-4: EVPN-VXLAN**

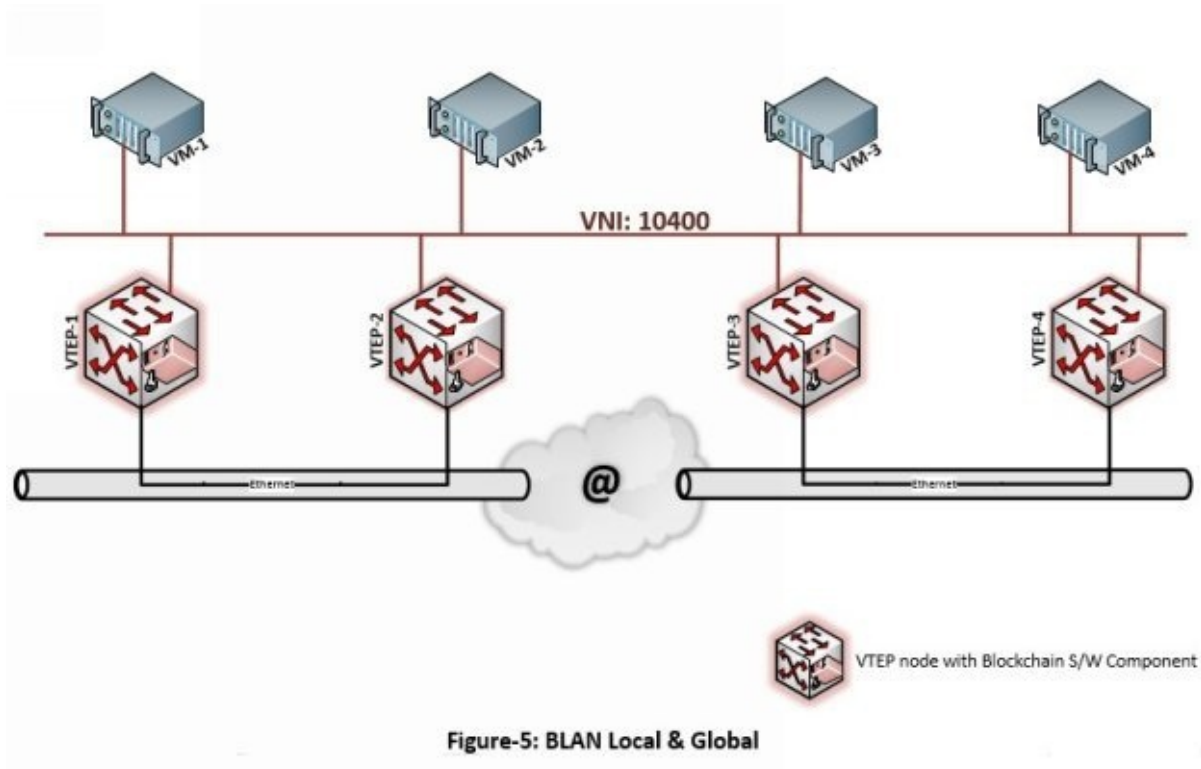
As in the Controller-based VXLAN; there would be no Unknown Unicast as the list of all communicating MACs is on each VTEP, but in case of an unknown MAC; the local VTEP can direct the traffic via the default entry towards the VXLAN gateway.

Again for the other Broadcast & Multicast traffic; the head-end replication is always the solution.

### 2.4.3. Blockchain-VXLAN 'BLAN'

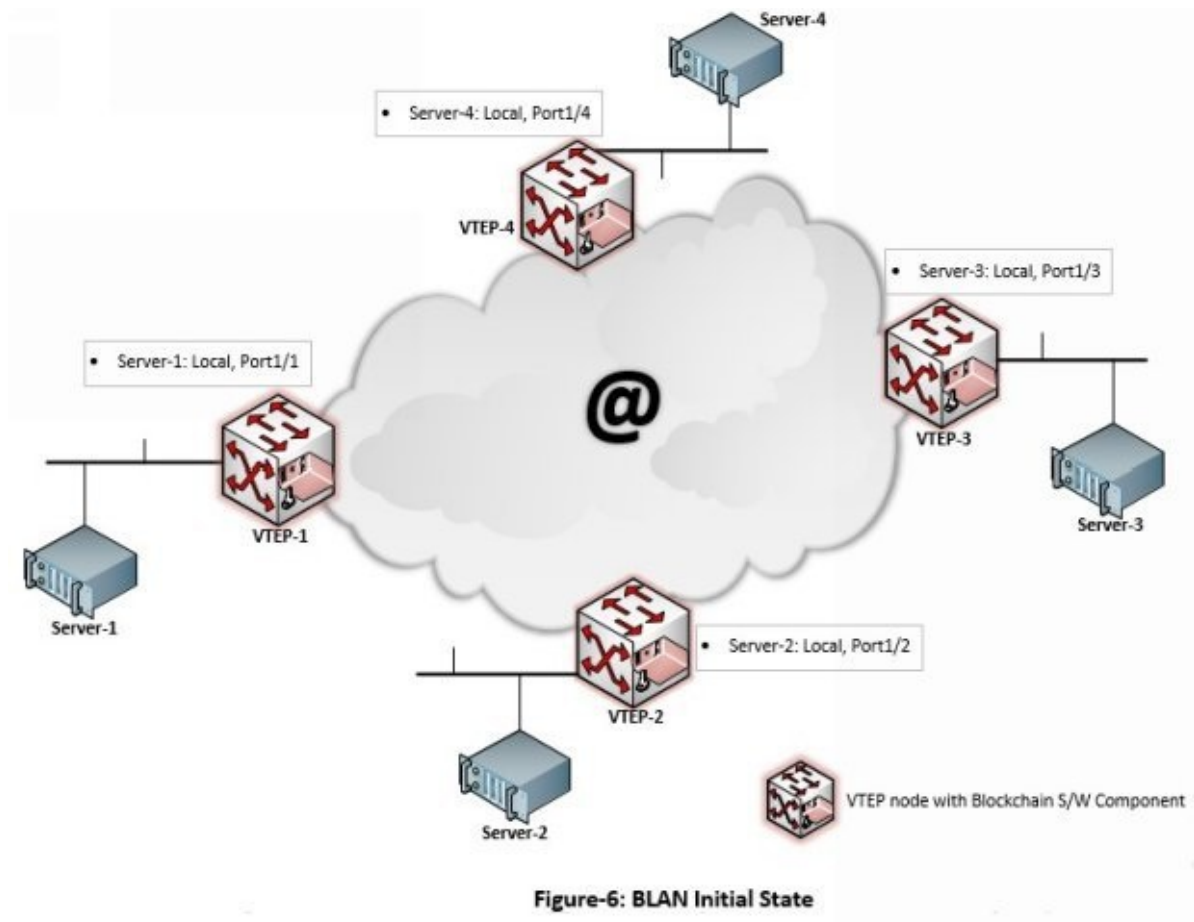
In BLAN; our proposal is that the control plane operation can be achieved via the blockchain technology.

In this mode, the VXLAN VTEPs are also nodes of a public or private blockchain (as in Figure-5) that can span across the public Internet. Note that there is always the option to replace blockchain with a DAPP (Decentralized Application) that runs over a blockchain-based platform, but let's stick to the blockchain option here.



It's worth noting that a private blockchain - as opposed to a public blockchain - is much faster, cheaper and respects the organization's privacy.

The local MAC learning technique is still the same as with any other VXLAN operation; the switches/VTEPs learn the local MAC addresses via their local ports (as the initial state in Figure-6) and then advertise/publish these addresses as reachable through their VTEP IPs over the blockchain (using transactions that are then 'packed' into proper blocks)



If we recall from the ‘Data Storage and Retrieval’ section; we’ve successfully published a stream of Hexadecimal data over the blockchain and we clarified that this stream is not a random one instead it does have a meaning.

What we actually did is that we’ve encoded some info in the form of Hexadecimal data, let’s use any online converter tool that can convert from Hexadecimal to Text to see what was meant by the Data Streams published by some nodes.

From a node, we published:

**‘637573746f6d65722031353520766e6964203130343030’**

## Convert hexadecimal to text

Input data

```
637573746f6d6572203135353520766e6964203130343030
```

Convert

hex numbers to text

Output:

```
customer 155 vnid 10400
```

The output text is: **‘customer 155 vnid 10400’**.

This simple output message could be used to carry some info related to the BLAN operation for a certain customer.

From another node, we published:

**‘637573746f6d6572203135353520766e6964203132353030’**; that simply means: **‘customer 155 vnid 12500’** i.e. another VNID for that same customer.

In the actual implementation, it will not be as straightforward as what we’ve illustrated, instead we can use very complicated encryption techniques to properly encrypt the sensitive info we’re sending over the blockchain.

Also, the published messages won’t be that simple, the messages need to include all information pertaining to a specific client for the proper BLAN operation, let’s consider this use case:

Customer #1555 LAN segment is connected to VTEP-1 (in Figure-6), VTEP-1 needs to participate in VXLAN with VNID 10123; VTEP-1 just learnt the MAC address (00-14-22-01-23-45) from one of the locally attached servers belonging to Customer#1555 (Server-1 attached to Port 1/1).

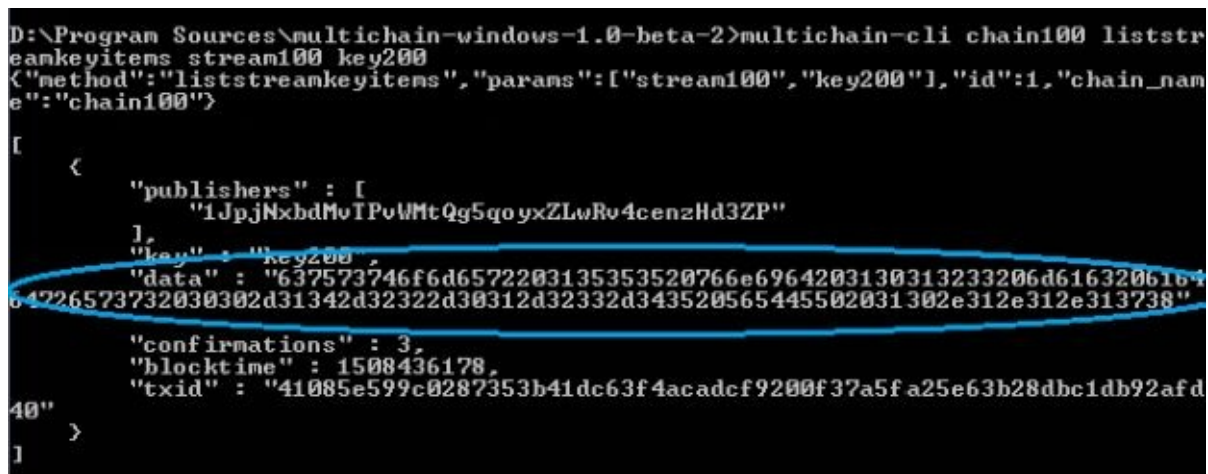
VTEP-1 IP address is 10.1.1.178, this is the IP address that other VTEPs need to

use to reach VTEP-1.

So the message that will be published from that VTEP over the blockchain is typically a MAC-to-VTEP mapping message that also includes the Customer ID as well as the VNID.

In this use case, the message will be: **'customer 1555 vnid 10123 mac address 00-14-22-01-23-45 VTEP 10.1.1.178'**

Its Hexadecimal format that will actually be published over the blockchain is: **'637573746f6d6572203135353520766e6964203130313233206d6163206164647**



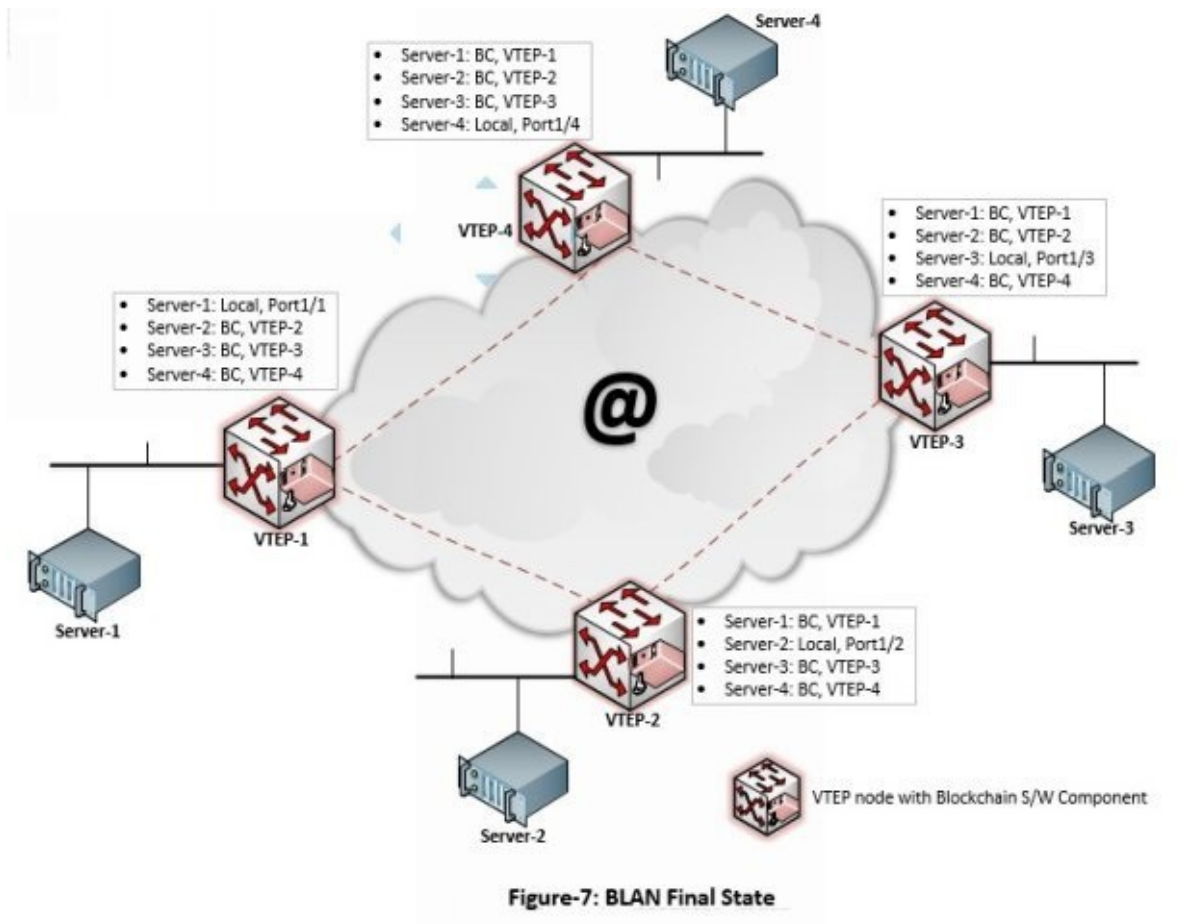
```
D:\Program Sources\multichain-windows-1.0-beta-2>multichain-cli chain100 liststreamkeyitems stream100 key200
{"method": "liststreamkeyitems", "params": ["stream100", "key200"], "id": 1, "chain_name": "chain100"}

[
  {
    "publishers": [
      "1JpJNxbdmvTPvWMtQg5qoyxZLwRv4cenzHd3ZP"
    ],
    "key": "key200",
    "data": "637573746f6d6572203135353520766e6964203130313233206d6163206164647647226573732030302d31342d32322d30312d32332d343520565445502031302e312e312e313738",
    "confirmations": 3,
    "blocktime": 1508436178,
    "txid": "41085e599c0287353b41dc63f4acadc9200f37a5fa25e63b28dbc1db92afd40"
  }
]
```

This message can be seen by all nodes (VTEPs) participating in the blockchain but only those VTEPs that are interested in Customer ID 1555 and VXLAN VNID 10123 will use this message, translate it and add its content to their local copy of the MAC-to-VTEP mappings.

Now because the different MAC-to-VTEP mappings are distributed over the blockchain to all participating nodes/VTEPs (as the final state in Figure-7):

- No Data-Plane learning (Flood & Learn) is required for unknown unicast MAC addresses.
- No IP Multicast underlay required; that is why BLAN can span across the Public INTERNET & not only within the boundary of a Data Center or a Cloud (private cloud or even public cloud).
- As per the distributed nature of blockchain; no significant delay would be expected between the different nodes.
- For the Broadcast & Multicast traffic; the head-end replication is always the solution as in other control-plane-based VXLAN modes.



### **3. BLAN Value Proposition**

As it can operate seamlessly across the public Internet, BLAN can be considered as an appealing WAN transport option in front of traditional WAN technologies like VPN or even the ‘very expensive’ dedicated links.

With BLAN, customers can leverage the public Internet for their WAN traffic so they don’t need to share any traffic with their upstream providers like the case with MPLS VPN service or the modern SD-WAN option.

No special devices are needed to be installed in the customer premises given that the current physical or virtual devices support VXLAN and can accommodate the blockchain software as well.

No Multicast capable underlay transport network is needed that is why BLAN can operate over any IP transport network including the global public Internet.

No Unknown Unicast entry would be found on any VTEPs; it’s either a unicast MAC address match (advertised over the blockchain) or a default entry towards the VXLAN gateway.

Multicast & Broadcast traffic is handled via the head-end replication on the source VTEP to all other known VTEPs in the same VXLAN (the list of VTEPs is known - and always updated - over the blockchain)

BLAN is the transport service of choice for all businesses even the ones dealing with critical high-frequency trading applications



## 4. BLAN Way Forward

Our plan - a short-term one - is to bring the BLAN proposal to reality; the solution needs lots of development and coding.

There are also some challenges that need to be sorted out like the maximum transmission unit, MTU:

Due to the MAC-to-UDP encapsulation performed by VXLAN, it introduces 50-byte overhead to the original frames.

Therefore, the MTU in the transport network needs to be increased by 50 bytes. If the overlays use a 1500-byte MTU, the transport network needs to be configured to accommodate 1550-byte packets at a minimum. Jumbo-frame support in the transport network is required if the overlay applications tend to use larger frame sizes than 1500 bytes. That would be a real challenge when the transport network is extended over the Internet; so techniques like PMTUD would have to take place to avoid any possible fragmentation for example.

As a reader of this BLAN eBook, and hopefully the full series of ‘Blockchain: the Hidden Gem of Data Networking’, you are actually participating along with other emerging technologies enthusiasts in the development needed for this solution; just drop me an email on: [moamin77@gmail.com](mailto:moamin77@gmail.com) and I’ll share with you all details.

Don’t worry! You have nothing to do! You will only enjoy being a shareholder of that project!

## **5. What's next?**

In this first volume of 'Blockchain: The Hidden Gem of Data Networking'; we've seen how blockchain, when integrated with a networking technology like VXLAN, can create a totally different technology (BLAN) that not only enhances the operation of the original technology but also has its unique use cases.

In the coming volumes, we'll use the same approach, i.e. integrating blockchain with other Networking technologies.

Actually I can specify the technologies that we can integrate blockchain with but - although I've never been a marketer - I believe it would be much better if I keep the names for now and talk about them on a case-by-case basis in the coming volumes.

**Thank You!**